Magdalena Larmet
Salesforce
Developer

Park'it

PARK IT

IMPLÉMENTATION D'UNE FONCTIONNALITÉ **JAVA**

Sommaire

1 INTRODUCTION

Rappel du projet et des spécifications.

Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

Résumé du projet

30-10-2022	30-11-2022	A DEFINIR
Date démarrage	Date de fin	Coût du projet

BESOINS EXPRIMÉS

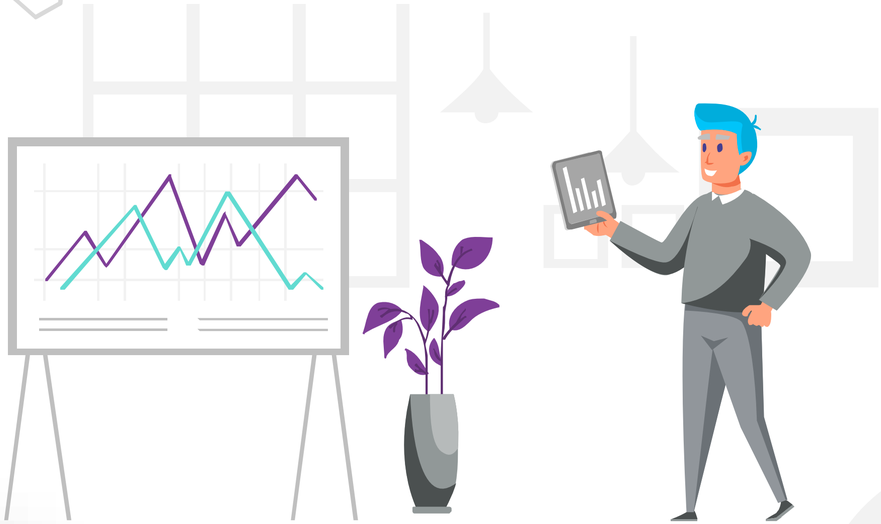
Ajouter une fonctionnalité de stationnement gratuit pour les 30 premières minutes ;
Ajouter une réduction de 5 % pour les utilisateurs récurrents ;
Corriger le code afin qu'il valide tous les tests unitaires ;
Effectuer les tests d'intégration marqués par les commentaires "TODO".

GRANDES LIGNES DU PROJET

La société Move it a récemment démarré un nouveau système de paiement de parking automatisé appelé Park'it.
L'application présente plusieurs bugs qui doivent être corrigés.

PHASES DU PROJET

1. **CRÉATION DU REPOSITORY**
   Mettre en place l'environnement et corriger les bugs de connection.
2. **FONCTION 30 MINUTES GRATUITES**
   Les 30 premières minutes gratuites.
3. **TESTS UNITAIRES**
   Correction du code pour passer les tests unitaires.
4. **TESTS INTEGRATION**
   Créer les tests d'Intégration pour couvrir un maximum de cas de figure.
5. **GÉNÉRATION DES RAPPORTS**
   Exporter les rapports au format HTML.



CAR

Développer un système de parking de véhicule automatisé.

BIKE

2 Le REPOSITORY

Organiser les branches pour faciliter le développement de la solution.

Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

Organisation du repository

Programmation orientée objet Java

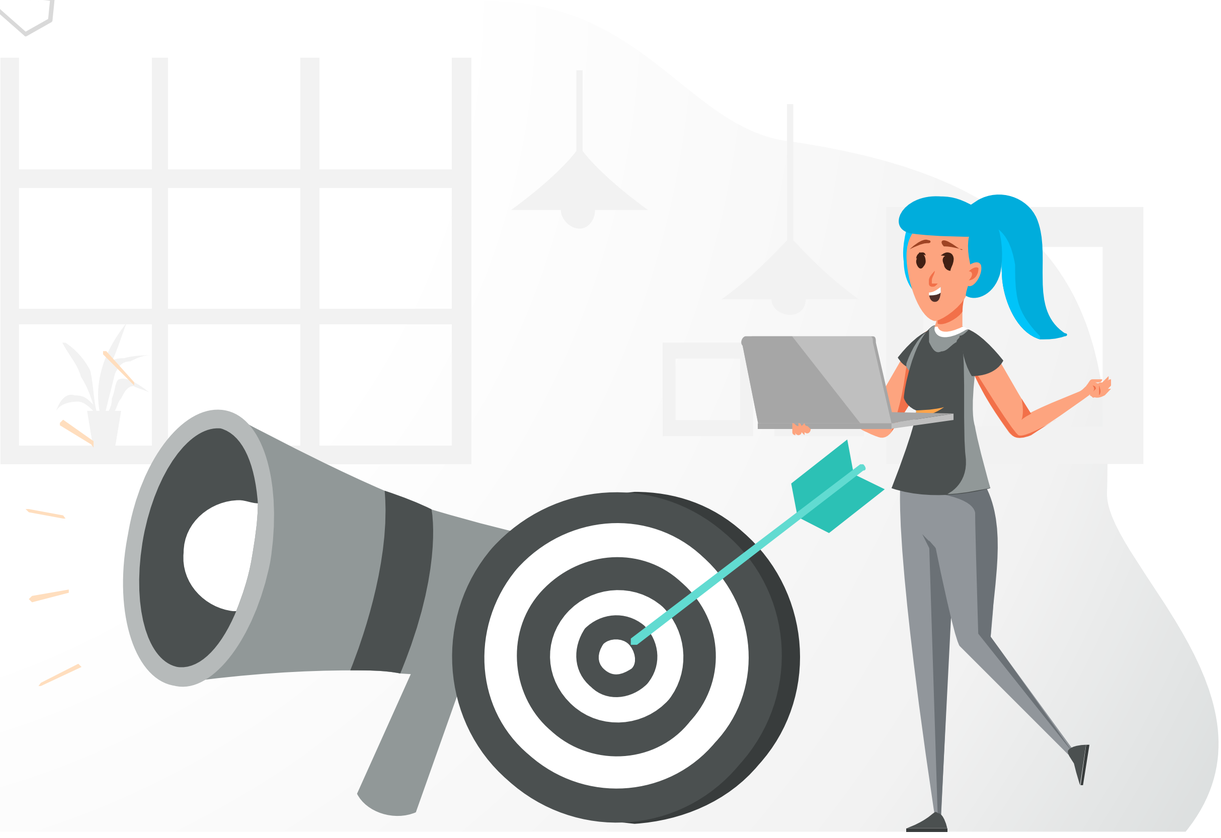
PARKINGSYSTEM

Branch Master

Contient le code final

Branch Developpement

Utilisé pour tester les bouts de code avant integration sur le Master.



3 Les FONCTIONS

Exporter les divers rapports et apporter les corrections nécessaires.

Fonction 30 minutes gratuites

FareCalculatorService

Déduire 30 minutes du temps total avant de calculer le prix

```
long inHour = ticket.getInTime().getTime();
long outHour = ticket.getOutTime().getTime();
double totalInMinutes = (outHour-inHour)/(1000*60);

if (totalInMinutes <= 30.00) {
    finalPrice = 0.0;
    System.out.println("Les 30 first are on us ! You have nothing to pay, See you soon");
    ticket.setPrice(finalPrice);
} else{
    switch (ticket.getParkingSpot().getParkingType()) {
        case CAR: {
            free30MinDiscount(totalInMinutes);
            finalPrice = (totalInMinutes * (Fare.CAR_RATE_PER_HOUR / 60));
            ticket.setPrice(finalPrice);

            break;
        }
        case BIKE: {
            free30MinDiscount(totalInMinutes);
            finalPrice = (totalInMinutes * (Fare.BIKE_RATE_PER_HOUR / 60));
            ticket.setPrice(finalPrice);

            break;
        }
        default:
            throw new IllegalArgumentException("Unknown Parking Type");

    }
```

Récupérer l'heure d'entrée et l'heure de sortie

Calculer le temps de parking et le convertir en minutes

Appliquer la function free30MinDiscount sur tous les tickets de plus de 30 mins

Mettre le temps à 0 pour les tickets de moins de 30 mins

```
2 usages   magda_dev
public void free30MinDiscount(double totalInMinutes){
    totalInMinutes = totalInMinutes - 30.0;
}
```

Fonction 5% de réduction

DiscountForReturning

Les étapes

ETAPE 01

Ajout d'un champ boolean ReturningClient à la base de données

ETAPE 03

Création de la fonction

ETAPE 02

Ajout d'un champ boolean ReturningClient à la base de données

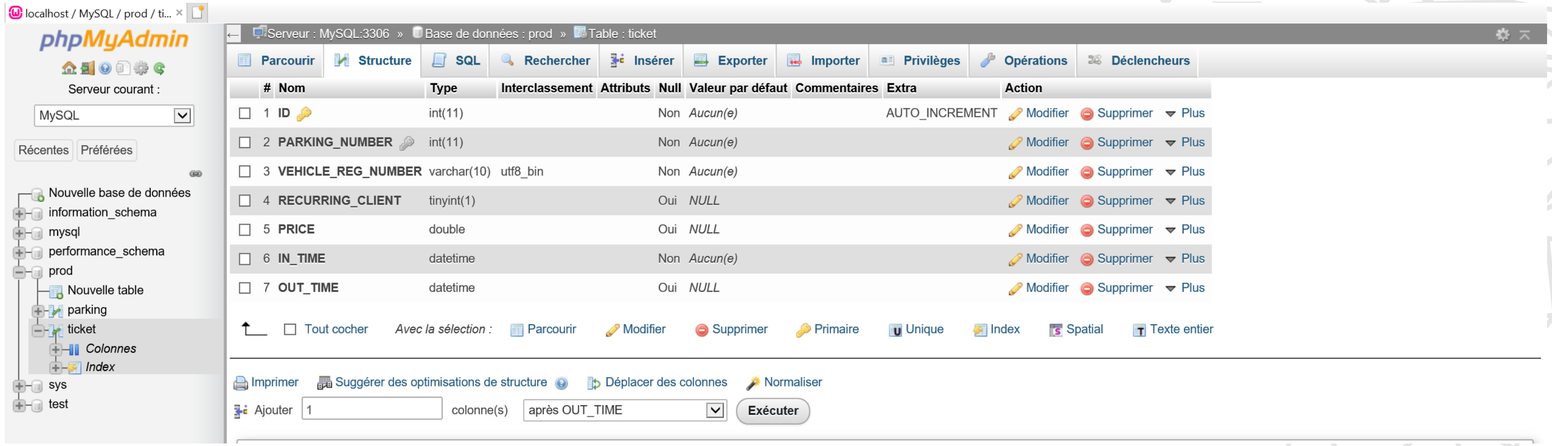
Fonction 5% de réduction

DiscountForReturning

La base de données mysql

1

Ajout d'un champ boolean ReturningClient à la base de données




```
2 usages   magda_dev
public boolean isReturningClient() { return returningClient;}

4 usages   magda_dev
public Boolean setReturningClient(boolean returningClient) { this.returningClient = returningClient;return returningClient;}
```

Fonction 5% de réduction

DiscountForReturning

La base de données et les requêtes

2

Création de la requête

```
package com.parkit.parkingsystem.constants;

avinash-dev *
public class DBConstants {

    1 usage
    public static final String GET_NEXT_PARKING_SPOT = "select min(PARKING_NUMBER) from parking where AVAILABLE = true and TYPE = ?";
    1 usage
    public static final String UPDATE_PARKING_SPOT = "update parking set available = ? where PARKING_NUMBER = ?";
    1 usage
    public static final String IS_RECURRING_CLIENT = "select * from ticket where VEHICLE_REG_NUMBER = ?";

    1 usage
    public static final String SAVE_TICKET = "insert into ticket(PARKING_NUMBER, VEHICLE_REG_NUMBER, PRICE, IN_TIME, OUT_TIME) values(?,?,?,?,?)";
    1 usage
    public static final String UPDATE_TICKET = "update ticket set PRICE=?, OUT_TIME=? where ID=?";
    1 usage
    public static final String GET_TICKET = "select t.PARKING_NUMBER, t.ID, t.PRICE, t.IN_TIME, t.OUT_TIME, p.TYPE from ticket t,parking p where p.parking_number = t.parking_number and t.VEHICLE_
}
```

Fonction 5% de réduction

DiscountForReturning

La formule

3

Création de la fonction

```
    public void discountForReturning (Ticket ticket){

        if (ticket.isReturningClient()) {
            ticket.setPrice(finalPrice-(finalPrice * 0.05));
            System.out.println("Price after discount : " + ticket.getPrice());
        }
    }
```

```
    long inHour = ticket.getInTime().getTime();
    long outHour = ticket.getOutTime().getTime();
    double totalInMinutes = (outHour-inHour)/(double)(1000*60);

    if (totalInMinutes <= 30.00) {
        finalPrice = 0.0;
        System.out.println("Les 30 first are on us ! You have nothing to pay, See you soon");
        ticket.setPrice(finalPrice);
    } else{
        switch (ticket.getParkingSpot().getParkingType()) {
            case CAR: {
                free30MinDiscount(totalInMinutes);
                finalPrice = (totalInMinutes * (Fare.CAR_RATE_PER_HOUR / 60));
                ticket.setPrice(finalPrice);

                break;
            }
            case BIKE: {
                free30MinDiscount(totalInMinutes);
                finalPrice = (totalInMinutes * (Fare.BIKE_RATE_PER_HOUR / 60));
                ticket.setPrice(finalPrice);

                break;
            }
            default:
                throw new IllegalArgumentException("Unknown Parking Type");

        }

    }

    discountForReturning(ticket);
}
```

4 Les TESTS

Exporter les divers rapports et apporter les corrections nécessaires.

Les todos

testParkingACar

Vérifier qu'un ticket est bien enregistré dans la BDD

Vérifier que la place de parking utilisé est en non disponible en BDD

```
1 usage   magda_dev +1 *
@Test
public void testParkingACar(){
    ParkingService parkingService = new ParkingService(inputReaderUtil, parkingSpotDAO, ticketDAO);
    Ticket generatedTicket = parkingService.processIncomingVehicle();
    Ticket ticketInDB = ticketDAO.getTicket(generatedTicket.getVehicleRegNumber());
    Date inTime = new Date();
    inTime.setTime( System.currentTimeMillis());
    assertNotNull(ticketInDB);
    ParkingSpot parkingSpot = ticketInDB.getParkingSpot();
    assertFalse(parkingSpot.isAvailable());
    //TODO: check that a ticket is actually saved in DB and Parking table is updated with availability
}

avinash-dev +1 *
@Test
```

assertNotNull & assertFalse

Les todos

testParkingLotExit

Vérifier que la date de sortie est bien enregistrée en BDD

Vérifier qu'un tarif est correctement généré et enregistré en BDD

```
@Test
public void testParkingLotExit(){
    testParkingACar();
    //checking that outime is null before processing exiting vehicle
    assertNull(ticketDAO.getTicket( vehicleRegNumber: "ABCDEF").getOutTime());

    ParkingService parkingService = new ParkingService(inputReaderUtil, parkingSpotDAO, ticketDAO);
    Ticket generatedTicket = parkingService.processExitingVehicle();
    Ticket ticketInDB = ticketDAO.getTicket(generatedTicket.getVehicleRegNumber());
    //checking that data is populated intot DB
    assertNotNull(ticketInDB);
    assertNotNull(ticketInDB.getOutTime());

    //checking that fare calculation is correct; fare should be 0
    assertEquals( expected: 0, ticketInDB.getPrice());
    //TODO: check that the fare generated and out time are populated correctly in the database
}
```

assertNull
assertNotNull
& assertEquals

Les tests unitaires

processIncomingVehicle

Tester une Exception

```
    @Test
    public void processIncomingVehicleHandlesNoSpotAvailableExceptionTest(){
        when(inputReaderUtil.readSelection()).thenReturn( value: 4);
        try {
            parkingService.getNextParkingNumberIfAvailable();
        } catch (IllegalArgumentException ie){
            Assertions.assertEquals(ie.getMessage(),  actual: "Entered input is invalid");
        }

    }

}
```

Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

Les tests d'intégration

calculateFare

Divers scénarios

```
magda_dev *
@Test
public void calculateFareForReturningCustomersCar(){
    Date inTime = new Date(System.currentTimeMillis() - ( 60 * 60 * 1000) );//1hour parking time should give 24 * parking fare per hour
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.CAR, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    ticket.setReturningClient(true);
    fareCalculatorService.calculateFare(ticket);
    assertEquals( expected: 1.425, ticket.getPrice());
}

magda_dev
@Test
public void calculateFareForReturningCustomersCBike(){
    Date inTime = new Date(System.currentTimeMillis() - ( 60 * 60 * 1000) );//1hour parking time should give 24 * parking fare per hour
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.BIKE, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    ticket.setReturningClient(true);
    fareCalculatorService.calculateFare(ticket);
    assertEquals( expected: 0.95, ticket.getPrice());
}

}
```

Les tests d'intégration

calculateFare

Divers scénarios

magda_dev *

```
@Test
public void calculateFareCarWithLessThan30Min(){
    Date inTime = new Date( System.currentTimeMillis() - (  10 * 60 * 1000) );
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.CAR, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    fareCalculatorService.calculateFare(ticket);
    assertEquals(  expected: 0, ticket.getPrice());
}
```

magda_dev

```
@Test
public void calculateFareBikeWithLessThan30Min(){
    Date inTime = new Date( System.currentTimeMillis() - (  10 * 60 * 1000) );
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.BIKE, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    fareCalculatorService.calculateFare(ticket);
    assertEquals(  expected: 0 , ticket.getPrice());
}
```

Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

Les tests d'intégration

calculateFare

Divers scénarios

```
magda_dev *
@Test
public void calculateFareForReturningCustomersCar(){
    Date inTime = new Date(System.currentTimeMillis() - ( 60 * 60 * 1000) );//1hour parking time should give 24 * parking fare per hour
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.CAR, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    ticket.setReturningClient(true);
    fareCalculatorService.calculateFare(ticket);
    assertEquals( expected: 1.425, ticket.getPrice());
}

magda_dev
@Test
public void calculateFareForReturningCustomersCBike(){
    Date inTime = new Date(System.currentTimeMillis() - ( 60 * 60 * 1000) );//1hour parking time should give 24 * parking fare per hour
    Date outTime = new Date(System.currentTimeMillis());
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.BIKE, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    ticket.setReturningClient(true);
    fareCalculatorService.calculateFare(ticket);
    assertEquals( expected: 0.95, ticket.getPrice());
}

}
```

Les tests d'intégration

calculateFare

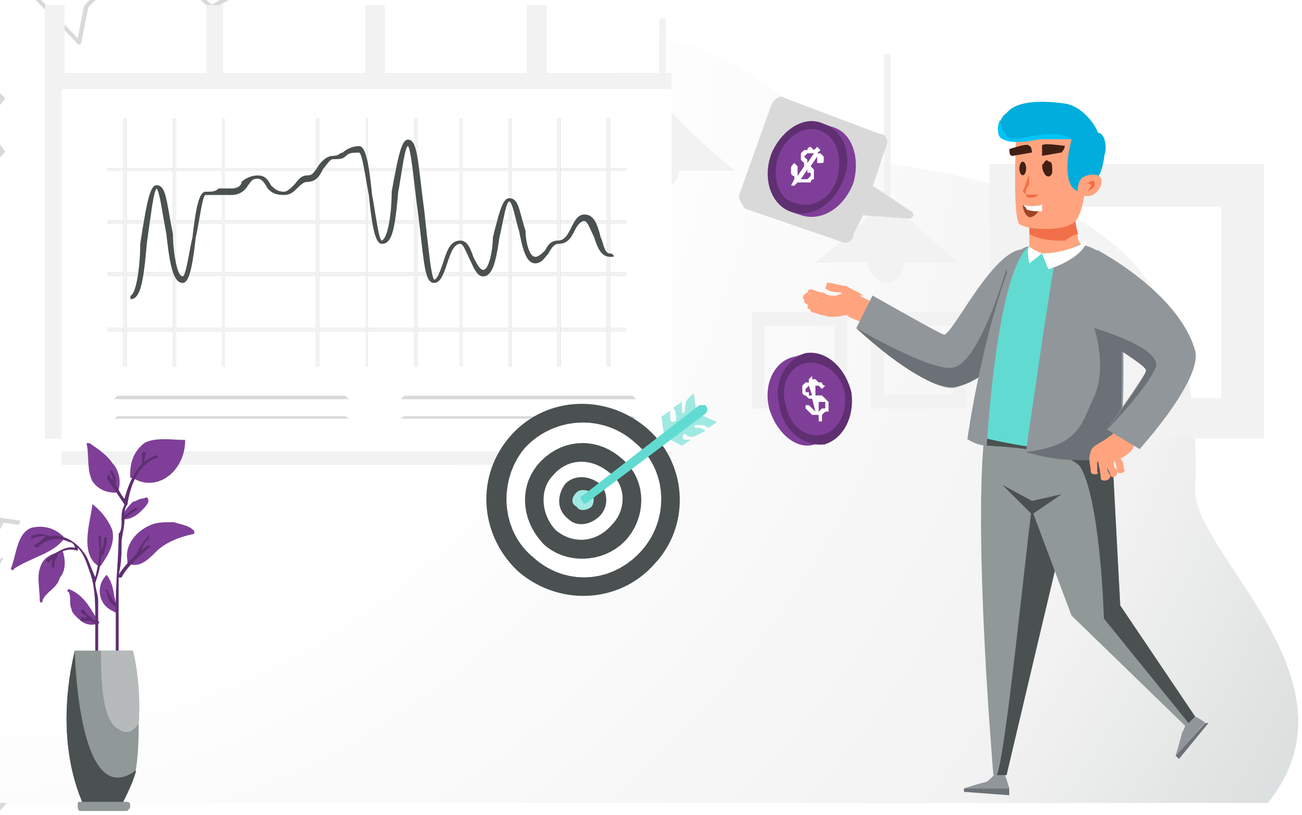
Divers scénarios

```
@Test
public void calculateFareCar(){
    Date inTime = new Date();
    inTime.setTime( System.currentTimeMillis() - (  60 * 60 * 1000) );
    Date outTime = new Date();
    ParkingSpot parkingSpot = new ParkingSpot( number: 1, ParkingType.CAR, isAvailable: false);

    ticket.setInTime(inTime);
    System.out.println(inTime);
    ticket.setOutTime(outTime);
    System.out.println(outTime);
    ticket.setParkingSpot(parkingSpot);
    fareCalculatorService.calculateFare(ticket);
    System.out.println(ticket.getPrice());
    assertEquals( expected: 1*Fare.CAR_RATE_PER_HOUR, ticket.getPrice());
}

 avinash-dev +1
@Test
public void calculateFareBike(){
    Date inTime = new Date();
    inTime.setTime( System.currentTimeMillis() - (  60 * 60 * 1000) ); //CALCUL POUR UNE H DE PARKING
    Date outTime = new Date();
    ParkingSpot parkingSpot = new ParkingSpot( number: 4, ParkingType.BIKE, isAvailable: false);

    ticket.setInTime(inTime);
    ticket.setOutTime(outTime);
    ticket.setParkingSpot(parkingSpot);
    fareCalculatorService.calculateFare(ticket);
    assertEquals( expected: 1* Fare.BIKE_RATE_PER_HOUR , ticket.getPrice());
}
```



5 Les RAPPORTS

Exporter les divers rapports et apporter les corrections nécessaires.

Le rapport Spotbugs

Repérer les bugs et corriger les bugs les plus critiques

Rapport html

SpotBugs Report

Produced using SpotBugs4.4.2.

Project: parkingsystem[parking-system]

Metrics

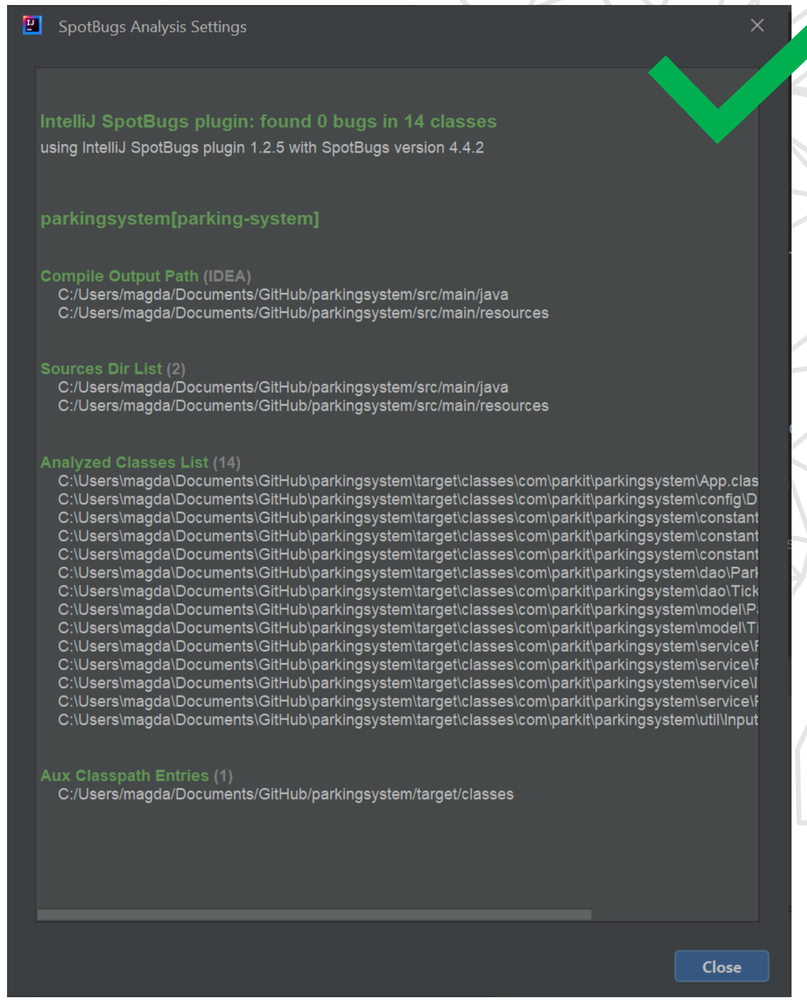
442 lines of code analyzed, in 14 classes, in 7 packages.

Metric	Total	Density*
High Priority Warnings	1	2.26
Medium Priority Warnings	9	20.36
Total Warnings	**10**	**22.62**

(Defects per Thousand lines of non-commenting source statements)*

Summary

Warning Type	Number
Experimental Warnings	9
Internationalization Warnings	1
Total	**10**



Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

Le rapport Spotbugs

Repérer les bugs et corriger les bugs les plus critiques

Rapport détaillé

Experimental Warnings

Warning	Priority	Details
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.ParkingSpotDAO.getNextAvailableSlot(ParkingType) may fail to clean up java.sql.ResultSet on checked exception In file ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java, line 26 28 33 34 37 38 40 In class com.parkit.parkingsystem.dao.ParkingSpotDAO In method com.parkit.parkingsystem.dao.ParkingSpotDAO.getNextAvailableSlot(ParkingType) Reference type java.sql.ResultSet 1 instances of obligation remaining Obligation to clean up resource created at ParkingSpotDAO.java:[line 26] is not discharged Path continues at ParkingSpotDAO.java:[line 28] Path continues at ParkingSpotDAO.java:[line 33] Path continues at ParkingSpotDAO.java:[line 34] Path continues at ParkingSpotDAO.java:[line 37] Path continues at ParkingSpotDAO.java:[line 38] Path continues at ParkingSpotDAO.java:[line 40] Remaining obligations: {Statement x 1,ResultSet x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.ParkingSpotDAO.getNextAvailableSlot(ParkingType) may fail to clean up java.sql.Statement on checked exception In file ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java, line 24 25 26 28 33 34 37 38 40 In class com.parkit.parkingsystem.dao.ParkingSpotDAO In method com.parkit.parkingsystem.dao.ParkingSpotDAO.getNextAvailableSlot(ParkingType) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at ParkingSpotDAO.java:[line 24] is not discharged Path continues at ParkingSpotDAO.java:[line 25] Path continues at ParkingSpotDAO.java:[line 26] Path continues at ParkingSpotDAO.java:[line 28] Path continues at ParkingSpotDAO.java:[line 33] Path continues at ParkingSpotDAO.java:[line 34] Path continues at ParkingSpotDAO.java:[line 37] Path continues at ParkingSpotDAO.java:[line 38] Path continues at ParkingSpotDAO.java:[line 40] Remaining obligations: {Statement x 1,ResultSet x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.isRecurringClient(String) may fail to clean up java.sql.ResultSet on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 108 110 122 123 125 126 127 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.isRecurringClient(String) Reference type java.sql.ResultSet 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 108] is not discharged Path continues at TicketDAO.java:[line 110] Path continues at TicketDAO.java:[line 122] Path continues at TicketDAO.java:[line 123] Path continues at TicketDAO.java:[line 125] Path continues at TicketDAO.java:[line 126] Path continues at TicketDAO.java:[line 127] Remaining obligations: {Statement x 1,ResultSet x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.isRecurringClient(String) may fail to clean up java.sql.Statement on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 107 108 110 122 123 125 126 127 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.isRecurringClient(String) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 107] is not discharged Path continues at TicketDAO.java:[line 108] Path continues at TicketDAO.java:[line 110] Path continues at TicketDAO.java:[line 122] Path continues at TicketDAO.java:[line 123] Path continues at TicketDAO.java:[line 125] Path continues at TicketDAO.java:[line 126] Path continues at TicketDAO.java:[line 127] Remaining obligations: {Statement x 1,ResultSet x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.saveTicket(Ticket) may fail to clean up java.sql.Statement on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 30 33 41 42 44 45 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.saveTicket(Ticket) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 30] is not discharged Path continues at TicketDAO.java:[line 33] Path continues at TicketDAO.java:[line 41] Path continues at TicketDAO.java:[line 42] Path continues at TicketDAO.java:[line 44]
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.ParkingSpotDAO.updateParking(ParkingSpot) may fail to clean up java.sql.Statement on checked exception In file ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java ParkingSpotDAO.java, line 48 49 54 55 56 58 56 In class com.parkit.parkingsystem.dao.ParkingSpotDAO In method com.parkit.parkingsystem.dao.ParkingSpotDAO.updateParking(ParkingSpot) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at ParkingSpotDAO.java:[line 48] is not discharged Path continues at ParkingSpotDAO.java:[line 49] Path continues at ParkingSpotDAO.java:[line 54] Path continues at ParkingSpotDAO.java:[line 55] Path continues at ParkingSpotDAO.java:[line 56] Path continues at ParkingSpotDAO.java:[line 58] Path continues at ParkingSpotDAO.java:[line 56] Remaining obligations: {Statement x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.getTicket(String) may fail to clean up java.sql.ResultSet on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 57 58 73 74 76 77 78 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.getTicket(String) Reference type java.sql.ResultSet 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 57] is not discharged Path continues at TicketDAO.java:[line 58] Path continues at TicketDAO.java:[line 73] Path continues at TicketDAO.java:[line 74] Path continues at TicketDAO.java:[line 76] Path continues at TicketDAO.java:[line 77] Path continues at TicketDAO.java:[line 78] Remaining obligations: {Statement x 1,ResultSet x 1}
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.getTicket(String) may fail to clean up java.sql.Statement on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 54 56 57 58 73 74 76 77 78 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.getTicket(String) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 54] is not discharged Path continues at TicketDAO.java:[line 56] Path continues at TicketDAO.java:[line 57] Path continues at TicketDAO.java:[line 58] Path continues at TicketDAO.java:[line 73] Path continues at TicketDAO.java:[line 74]
Method may fail to clean up stream or resource on checked exception	Medium	com.parkit.parkingsystem.dao.TicketDAO.updateTicket(Ticket) may fail to clean up java.sql.Statement on checked exception In file TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java TicketDAO.java, line 85 86 92 93 95 96 97 In class com.parkit.parkingsystem.dao.TicketDAO In method com.parkit.parkingsystem.dao.TicketDAO.updateTicket(Ticket) Reference type java.sql.Statement 1 instances of obligation remaining Obligation to clean up resource created at TicketDAO.java:[line 85] is not discharged Path continues at TicketDAO.java:[line 86] Path continues at TicketDAO.java:[line 92] Path continues at TicketDAO.java:[line 93] Path continues at TicketDAO.java:[line 95] Path continues at TicketDAO.java:[line 96] Path continues at TicketDAO.java:[line 97] Remaining obligations: {Statement x 1}

Internationalization Warnings

Warning	Priority	Details
Reliance on default encoding	High	Found reliance on default encoding in com.parkit.parkingsystem.util.InputReaderUtil.<static initializer for InputReaderUtil>(): new java.util.Scanner(InputStream) In file InputReaderUtil.java InputReaderUtil.java, line 10 10 In class com.parkit.parkingsystem.util.InputReaderUtil In method com.parkit.parkingsystem.util.InputReaderUtil.<static initializer for InputReaderUtil>() Called method new java.util.Scanner(InputStream) At InputReaderUtil.java:[line 10] At InputReaderUtil.java:[line 10]

Magdalena Larmet | DEVELOPPEUR **SALESFORCE**

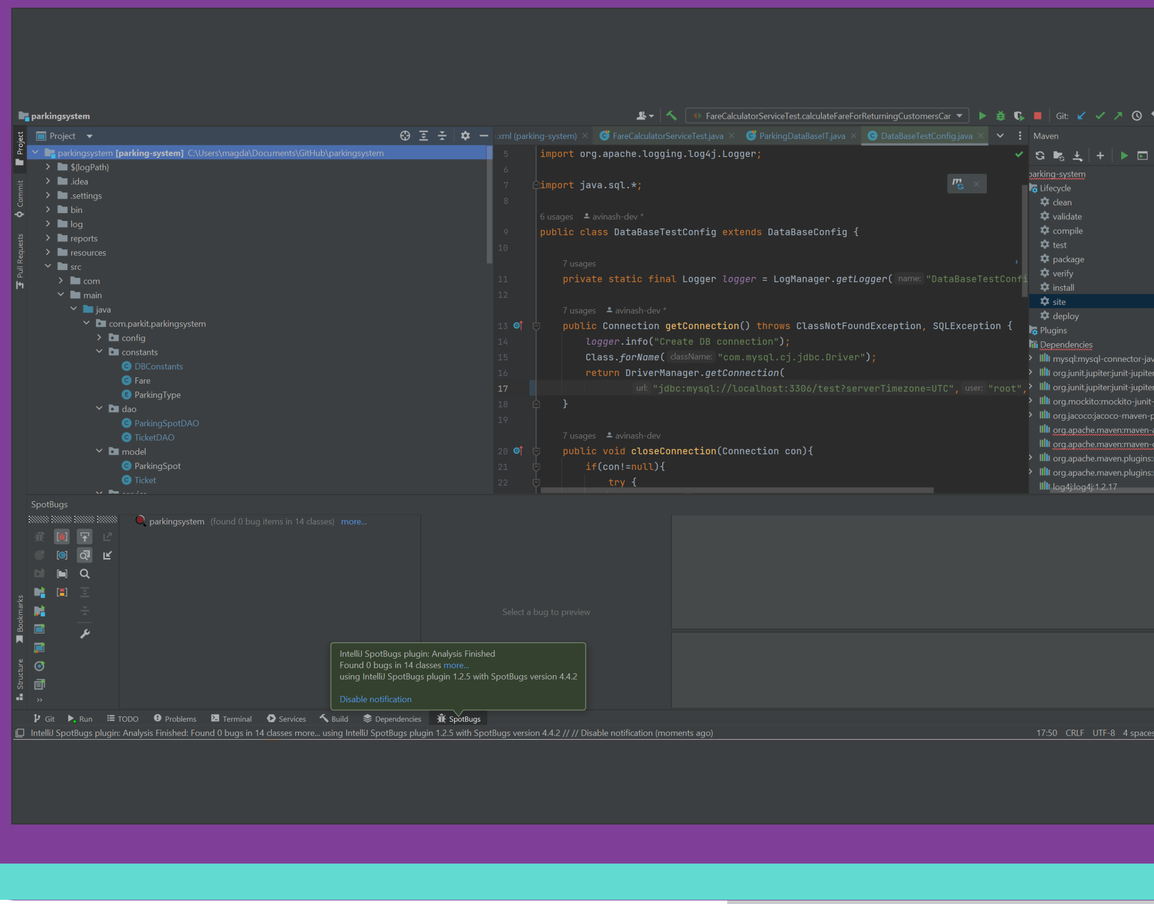
Le rapport Spotbugs

Repérer les bugs et corriger les bugs les plus critiques

Rapport détaillé

Une application sans bugs et facile à maintenir

Les bugs ont été corrigés et le programme est désormais sans bug apparent. Il suffira de relancer des tests si on intègre une autre fonctionnalité.



Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport global html

Couverture globale de **71%**

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	71,4% (10/14)	74,6% (50/67)	71,4% (225/315)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem	0% (0/1)	0% (0/3)	0% (0/4)
com.parkit.parkingsystem.config	100% (1/1)	100% (6/6)	73,9% (17/23)
com.parkit.parkingsystem.constants	33,3% (1/3)	33,3% (1/3)	60% (3/5)
com.parkit.parkingsystem.dao	100% (2/2)	100% (10/10)	85,3% (93/109)
com.parkit.parkingsystem.model	100% (2/2)	80% (20/25)	68,6% (24/35)
com.parkit.parkingsystem.service	75% (3/4)	75% (12/16)	70,5% (86/122)
com.parkit.parkingsystem.util	100% (1/1)	25% (1/4)	11,8% (2/17)

generated on 2022-12-05 19:53

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem

App

Coverage Summary for Package: com.parkit.parkingsystem

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem	100% (1/1)	66.7% (2/3)	75% (3/4)

Class	Class, %	Method, %	Line, %
App	100% (1/1)	66.7% (2/3)	75% (3/4)

generated on 2022-11-30 12:08

Coverage Summary for Class: App (com.parkit.parkingsystem)

Class	Class, %	Method, %	Line, %
App	100% (1/1)	66.7% (2/3)	75% (3/4)

```
 1 package com.parkit.parkingsystem;
 2
 3 import com.parkit.parkingsystem.service.InteractiveShell;
 4 import org.apache.logging.log4j.LogManager;
 5 import org.apache.logging.log4j.Logger;
 6
 7 public class App {
 8     private static final Logger logger = LogManager.getLogger("App");
 9     public static void main(String args[]){
10         logger.info("Initializing Parking System");
11         InteractiveShell.loadInterface();
12     }
13 }
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.config

DataBaseConfig

Coverage Summary for Package: com.parkit.parkingsystem.config

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.config	100% (1/1)	100% (6/6)	73,9% (17/23)

Class	Class, %	Method, %	Line, %
DataBaseConfig	100% (1/1)	100% (6/6)	73,9% (17/23)

Coverage Summary for Class: DataBaseConfig (com.parkit.parkingsystem.config)

Class	Class, %	Method, %	Line, %
DataBaseConfig	100% (1/1)	100% (6/6)	73,9% (17/23)

```
1  package com.parkit.parkingsystem.config;
2
3  import edu.umd.cs.findbugs.annotations.SuppressFBWarnings;
4  import org.apache.logging.log4j.LogManager;
5  import org.apache.logging.log4j.Logger;
6
7  import java.sql.*;
8
9  public class DataBaseConfig {
10
11     private static final Logger logger = LogManager.getLogger("DataBaseConfig");
12
13     @SuppressFBWarnings("DMI_EMPTY_DB_PASSWORD")
14     public Connection getConnection() throws ClassNotFoundException, SQLException {
15         logger.info("Create DB connection");
16         Class.forName("com.mysql.cj.jdbc.Driver");
17         return DriverManager.getConnection(
18                 "jdbc:mysql://localhost:3306/prod?serverTimezone=UTC","root","");
19     }
20
21     public void closeConnection(Connection con){
22         if(con!=null){
23             try {
24                 con.close();
25                 logger.info("Closing DB connection");
26             } catch (SQLException e) {
27                 logger.error("Error while closing connection",e);
28             }
29         }
30     }
31
32     public void closePreparedStatement(PreparedStatement ps) {
33         if(ps!=null){
34             try {
35                 ps.close();
36                 logger.info("Closing Prepared Statement");
37             } catch (SQLException e) {
38                 logger.error("Error while closing prepared statement",e);
39             }
40         }
41     }
42
43     public void closeResultSet(ResultSet rs) {
44         if(rs!=null){
45             try {
46                 rs.close();
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.constants

ParkingType

Coverage Summary for Package: com.parkit.parkingsystem.constants

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.constants	33.3% (1/3)	33.3% (1/3)	60% (3/5)

Class	Class, %	Method, %	Line, %
DBConstants	0% (0/1)	0% (0/1)	0% (0/1)
Fare	0% (0/1)	0% (0/1)	0% (0/1)
ParkingType	100% (1/1)	100% (1/1)	100% (3/3)

Coverage Summary for Class: ParkingType (com.parkit.parkingsystem.constants)

Class	Class, %	Method, %	Line, %
ParkingType	100% (1/1)	100% (1/1)	100% (3/3)

```
1 package com.parkit.parkingsystem.constants;
2
3 public enum ParkingType {
4     CAR,
5     BIKE
6 }
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.dao

ParkingSpotDAO

Coverage Summary for Package: com.parkit.parkingsystem.dao

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.dao	100% (2/2)	90% (9/10)	71.7% (76/106)

Class	Class, %	Method, %	Line, %
ParkingSpotDAO	100% (1/1)	100% (4/4)	80.6% (25/31)
TicketDAO	100% (1/1)	83.3% (5/6)	68% (51/75)

```java
package com.parkit.parkingsystem.dao;

import com.parkit.parkingsystem.config.DataBaseConfig;
import com.parkit.parkingsystem.constants.DBConstants;
import com.parkit.parkingsystem.constants.ParkingType;
import com.parkit.parkingsystem.model.ParkingSpot;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class ParkingSpotDAO {
    private static final Logger logger = LogManager.getLogger("ParkingSpotDAO");

    public DataBaseConfig dataBaseConfig = new DataBaseConfig();

    public int getNextAvailableSlot(ParkingType parkingType){
        Connection con = null;
        int result=-1;
        try {
            con = dataBaseConfig.getConnection();
            PreparedStatement ps = con.prepareStatement(DBConstants.GET_NEXT_PARKING_SPOT);
            ps.setString(1, parkingType.toString());
            ResultSet rs = ps.executeQuery();

            if(rs.next()){
                result = rs.getInt(1);
            }
            dataBaseConfig.closeResultSet(rs);
            dataBaseConfig.closePreparedStatement(ps);
        }catch (Exception ex){
            logger.error("Error fetching next available slot",ex);
        }finally {

            dataBaseConfig.closeConnection(con);
        }

        return result;
    }

    public boolean updateParking(ParkingSpot parkingSpot){
        //update the availability for that parking slot
        Connection con = null;
        try {
            con = dataBaseConfig.getConnection();
            PreparedStatement ps = con.prepareStatement(DBConstants.UPDATE_PARKING_SPOT);
            ps.setBoolean(1, parkingSpot.isAvailable());
            ps.setInt(2, parkingSpot.getId());
            int updateRowCount = ps.executeUpdate();
            dataBaseConfig.closePreparedStatement(ps);
            return (updateRowCount == 1);
        }catch (Exception ex){
            logger.error("Error updating parking info",ex);
            return false;
        }finally {
            dataBaseConfig.closeConnection(con);
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.dao

TicketDAO

Coverage Summary for Package: com.parkit.parkingsystem.dao

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.dao	100% (2/2)	90% (9/10)	71.7% (76/106)

Class	Class, %	Method, %	Line, %
ParkingSpotDAO	100% (1/1)	100% (4/4)	80.6% (25/31)
TicketDAO	100% (1/1)	83.3% (5/6)	68% (51/75)

```
18
19 public class TicketDAO {
20
21     private static final Logger logger = LogManager.getLogger("TicketDAO");
22
23     public DataBaseConfig dataBaseConfig = new DataBaseConfig();
24
25     public boolean saveTicket(Ticket ticket){
26         Connection con = null;
27         Boolean res = false;
28         try {
29             con = dataBaseConfig.getConnection();
30             PreparedStatement ps = con.prepareStatement(DBConstants.SAVE_TICKET);
31             //ID, PARKING_NUMBER, VEHICLE_REG_NUMBER, PRICE, IN_TIME, OUT_TIME)
32             //ps.setInt(1,ticket.getId());
33             ps.setInt(1,ticket.getParkingSpot().getId());
34             ps.setString(2, ticket.getVehicleRegNumber());
35             ps.setBoolean(3, ticket.isReturningClient());
36             ps.setDouble(3, ticket.getPrice());
37             ps.setTimestamp(4, new Timestamp(ticket.getInTime().getTime()));
38             ps.setTimestamp(5, (ticket.getOutTime() == null)?null: (new Timestamp(ticket.getOutTime().getTime())) );
39             res = ps.execute();
40             ps.close();
41         }catch (Exception ex){
42             logger.error("Error fetching next available slot",ex);
43         }finally {
44             dataBaseConfig.closeConnection(con);
45             return res;
46         }
47     }
48
49     public Ticket getTicket(String vehicleRegNumber) {
50         Connection con = null;
51         Ticket ticket = null;
52         try{
53             con = dataBaseConfig.getConnection();
54             PreparedStatement ps = con.prepareStatement(DBConstants.GET_TICKET);
55             //ID, PARKING_NUMBER, VEHICLE_REG_NUMBER, PRICE, IN_TIME, OUT_TIME)
56             ps.setString(1,vehicleRegNumber);
57             ResultSet rs = ps.executeQuery();
58             if(rs.next()){
59                 ticket = new Ticket();
60                 ParkingSpot parkingSpot = new ParkingSpot(rs.getInt(1), ParkingType.valueOf(rs.getString(6)),false);
61                 ticket.setParkingSpot(parkingSpot);
62                 ticket.setId(rs.getInt(2));
63                 ticket.setVehicleRegNumber(vehicleRegNumber);
64                 ticket.setReturningClient(true);
65                 ticket.setPrice(rs.getDouble(3));
66                 ticket.setInTime(rs.getTimestamp(4));
67                 ticket.setOutTime(rs.getTimestamp(5));
68
69             }
70
71             dataBaseConfig.closeResultSet(rs);
72             dataBaseConfig.closePreparedStatement(ps);
73         }catch (Exception ex){
74             logger.error("Error fetching next available slot",ex);
75         }finally {
76             dataBaseConfig.closeConnection(con);
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.model

ParkingSpot

Coverage Summary for Package: com.parkit.parkingsystem.model

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.model	100% (2/2)	86.4% (19/22)	78.6% (22/28)

Class	Class, %	Method, %	Line, %
ParkingSpot	100% (1/1)	71.4% (5/7)	61.5% (8/13)
Ticket	100% (1/1)	93.3% (14/15)	93.3% (14/15)

Coverage Summary for Class: ParkingSpot (com.parkit.parkingsystem.model)

Class	Class, %	Method, %	Line, %
ParkingSpot	100% (1/1)	71.4% (5/7)	61.5% (8/13)

```
1  package com.parkit.parkingsystem.model;
2
3  import com.parkit.parkingsystem.constants.ParkingType;
4
5  public class ParkingSpot {
6      private int number;
7      private ParkingType parkingType;
8      private boolean isAvailable;
9
10     public ParkingSpot(int number, ParkingType parkingType, boolean isAvailable) {
11         this.number = number;
12         this.parkingType = parkingType;
13         this.isAvailable = isAvailable;
14     }
15
16     public int getId() {
17         return number;
18     }
19
20     public ParkingType getParkingType() {
21         return parkingType;
22     }
23
24     public boolean isAvailable() {
25         return isAvailable;
26     }
27
28     public void setAvailable(boolean available) {
29         isAvailable = available;
30     }
31
32     @Override
33     public boolean equals(Object o) {
34         if (this == o) return true;
35         if (o == null || getClass() != o.getClass()) return false;
36         ParkingSpot that = (ParkingSpot) o;
37         return number == that.number;
38     }
39
40     @Override
41     public int hashCode() {
42         return number;
43     }
44 }
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.model

Ticket

Coverage Summary for Package: com.parkit.parkingsystem.model

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.model	100% (2/2)	86.4% (19/22)	78.6% (22/28)

Class	Class, %	Method, %	Line, %
ParkingSpot	100% (1/1)	71.4% (5/7)	61.5% (8/13)
Ticket	100% (1/1)	93.3% (14/15)	93.3% (14/15)

Coverage Summary for Class: Ticket (com.parkit.parkingsystem.model)

Class	Class, %	Method, %	Line, %
Ticket	100% (1/1)	93.3% (14/15)	93.3% (14/15)

```
1  package com.parkit.parkingsystem.model;
2
3  import edu.umd.cs.findbugs.annotations.SuppressFBWarnings;
4
5  import java.util.Calendar;
6  import java.util.Date;
7
8  @SuppressFBWarnings("EI_EXPOSE_REP")
9  public class Ticket {
10     private int id;
11     private ParkingSpot parkingSpot;
12     private String vehicleRegNumber;
13     private double price;
14     private Date inTime;
15     private Date outTime;
16     private boolean returningClient;
17
18     public int getId() {
19         return id;
20     }
21
22     public void setId(int id) {
23         this.id = id;
24     }
25
26     public ParkingSpot getParkingSpot() {
27         return parkingSpot;
28     }
29
30
31     public void setParkingSpot(ParkingSpot parkingSpot) {
32         this.parkingSpot = parkingSpot;
33     }
34
35     public String getVehicleRegNumber() {
36         return vehicleRegNumber;
37     }
38
39     public void setVehicleRegNumber(String vehicleRegNumber) {
40         this.vehicleRegNumber = vehicleRegNumber;
41     }
42
43     public double getPrice() {
44         return price;
45     }
46
47     public void setPrice(double price) {
48         this.price = price;
49     }
50
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.service

FareCalculatorService

Coverage Summary for Package: com.parkit.parkingsystem.service

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.service	100% (4/4)	93.8% (15/16)	84.4% (103/122)

Class	Class, %	Method, %	Line, %
FareCalculatorService	100% (2/2)	100% (5/5)	96.6% (28/29)
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)
ParkingService	100% (1/1)	100% (7/7)	89.6% (60/67)

Coverage Summary for Class: FareCalculatorService (com.parkit.parkingsystem.service)

Class	Method, %	Line, %
FareCalculatorService	100% (4/4)	96.4% (27/28)
FareCalculatorService$1	100% (1/1)	100% (1/1)
Total	100% (5/5)	96.6% (28/29)

```
1  package com.parkit.parkingsystem.service;
2
3  import com.parkit.parkingsystem.constants.Fare;
4  import com.parkit.parkingsystem.model.Ticket;
5  import edu.umd.cs.findbugs.annotations.SuppressFBWarnings;
6
7  public class FareCalculatorService {
8      public double finalPrice;
9
10     @SuppressFBWarnings("RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT")
11     public void calculateFare(Ticket ticket) {
12         if ((ticket.getOutTime() == null) || (ticket.getOutTime().before(ticket.getInTime()))) {
13             throw new IllegalArgumentException("Out time provided is incorrect:" + ticket.getOutTime().toString());
14         }
15
16         if((ticket.getParkingSpot().getParkingType()== null)){
17             throw new NullPointerException("Parking type is incorrect:" + ticket.getOutTime().toString());
18         }
19
20         long inHour = ticket.getInTime().getTime();
21         long outHour = ticket.getOutTime().getTime();
22         double totalInMinutes = (outHour-inHour)/(double)(1000*60);
23
24         if (totalInMinutes <= 30.00) {
25             finalPrice = 0.0;
26             System.out.println("Les 30 first are on us ! You have nothing to pay, See you soon");
27             ticket.setPrice(finalPrice);
28         } else{
29             switch (ticket.getParkingSpot().getParkingType()) {
30                 case CAR: {
31                     free30MinDiscount(totalInMinutes);
32                     finalPrice = (totalInMinutes * (Fare.CAR_RATE_PER_HOUR / 60));
33                     ticket.setPrice(finalPrice);
34
35                     break;
36                 }
37                 case BIKE: {
38                     free30MinDiscount(totalInMinutes);
39                     finalPrice = (totalInMinutes * (Fare.BIKE_RATE_PER_HOUR / 60));
40                     ticket.setPrice(finalPrice);
41
42                     break;
43                 }
44                 default:
45                     throw new IllegalArgumentException("Unknown Parking Type");
46
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.service

InteractiveShell

Coverage Summary for Package: com.parkit.parkingsystem.service

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.service	100% (4/4)	93.8% (15/16)	84.4% (103/122)

Class	Class, %	Method, %	Line, %
FareCalculatorService	100% (2/2)	100% (5/5)	96.6% (28/29)
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)
ParkingService	100% (1/1)	100% (7/7)	89.6% (60/67)

Coverage Summary for Class: InteractiveShell (com.parkit.parkingsystem.service)

Class	Class, %	Method, %	Line, %
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)

```
1  package com.parkit.parkingsystem.service;
2
3  import com.parkit.parkingsystem.dao.ParkingSpotDAO;
4  import com.parkit.parkingsystem.dao.TicketDAO;
5  import com.parkit.parkingsystem.util.InputReaderUtil;
6  import org.apache.logging.log4j.LogManager;
7  import org.apache.logging.log4j.Logger;
8
9  public class InteractiveShell {
10
11     private static final Logger logger = LogManager.getLogger("InteractiveShell");
12
13     public static void loadInterface(){
14         logger.info("App initialized!!!");
15         System.out.println("Welcome to Parking System!");
16
17         boolean continueApp = true;
18         InputReaderUtil inputReaderUtil = new InputReaderUtil();
19         ParkingSpotDAO parkingSpotDAO = new ParkingSpotDAO();
20         TicketDAO ticketDAO = new TicketDAO();
21         ParkingService parkingService = new ParkingService(inputReaderUtil, parkingSpotDAO, ticketDAO);
22
23         while(continueApp){
24             loadMenu();
25             int option = inputReaderUtil.readSelection();
26             switch(option){
27                 case 1: {
28                     parkingService.processIncomingVehicle();
29
30                     break;
31                 }
32                 case 2: {
33                     parkingService.processExitingVehicle();
34                     break;
35                 }
36                 case 3: {
37                     System.out.println("Exiting from the system!");
38                     continueApp = false;
39                     break;
40                 }
41                 default: System.out.println("Unsupported option. Please enter a number corresponding to the provided menu");
42             }
43         }
44     }
45
46     private static void loadMenu(){
47         System.out.println("Please select an option. Simply enter the number to choose an action");
48         System.out.println("1 New Vehicle Entering - Allocate Parking Space");
49         System.out.println("2 Vehicle Exiting - Generate Ticket Price");
50         System.out.println("3 Shutdown System");
51     }
52
53 }
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.service

InteractiveShell

Coverage Summary for Package: com.parkit.parkingsystem.service

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.service	100% (4/4)	93.8% (15/16)	84.4% (103/122)

Class	Class, %	Method, %	Line, %
FareCalculatorService	100% (2/2)	100% (5/5)	96.6% (28/29)
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)
ParkingService	100% (1/1)	100% (7/7)	89.6% (60/67)

Coverage Summary for Class: InteractiveShell (com.parkit.parkingsystem.service)

Class	Class, %	Method, %	Line, %
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)

```
1  package com.parkit.parkingsystem.service;
2
3  import com.parkit.parkingsystem.dao.ParkingSpotDAO;
4  import com.parkit.parkingsystem.dao.TicketDAO;
5  import com.parkit.parkingsystem.util.InputReaderUtil;
6  import org.apache.logging.log4j.LogManager;
7  import org.apache.logging.log4j.Logger;
8
9  public class InteractiveShell {
10
11     private static final Logger logger = LogManager.getLogger("InteractiveShell");
12
13     public static void loadInterface(){
14         logger.info("App initialized!!!");
15         System.out.println("Welcome to Parking System!");
16
17         boolean continueApp = true;
18         InputReaderUtil inputReaderUtil = new InputReaderUtil();
19         ParkingSpotDAO parkingSpotDAO = new ParkingSpotDAO();
20         TicketDAO ticketDAO = new TicketDAO();
21         ParkingService parkingService = new ParkingService(inputReaderUtil, parkingSpotDAO, ticketDAO);
22
23         while(continueApp){
24             loadMenu();
25             int option = inputReaderUtil.readSelection();
26             switch(option){
27                 case 1: {
28                     parkingService.processIncomingVehicle();
29
30                     break;
31                 }
32                 case 2: {
33                     parkingService.processExitingVehicle();
34                     break;
35                 }
36                 case 3: {
37                     System.out.println("Exiting from the system!");
38                     continueApp = false;
39                     break;
40                 }
41                 default: System.out.println("Unsupported option. Please enter a number corresponding to the provided menu");
42             }
43         }
44     }
45
46     private static void loadMenu(){
47         System.out.println("Please select an option. Simply enter the number to choose an action");
48         System.out.println("1 New Vehicle Entering - Allocate Parking Space");
49         System.out.println("2 Vehicle Exiting - Generate Ticket Price");
50         System.out.println("3 Shutdown System");
51     }
52
53 }
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.service

ParkingService

Coverage Summary for Package: com.parkit.parkingsystem.service

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.service	100% (4/4)	93.8% (15/16)	84.4% (103/122)

Class	Class, %	Method, %	Line, %
FareCalculatorService	100% (2/2)	100% (5/5)	96.6% (28/29)
InteractiveShell	100% (1/1)	75% (3/4)	57.7% (15/26)
ParkingService	100% (1/1)	100% (7/7)	89.6% (60/67)

Coverage Summary for Class: ParkingService (com.parkit.parkingsystem.service)

Class	Class, %	Method, %	Line, %
ParkingService	100% (1/1)	100% (7/7)	89.6% (60/67)

```
1  package com.parkit.parkingsystem.service;
2
3  import com.parkit.parkingsystem.constants.ParkingType;
4  import com.parkit.parkingsystem.dao.ParkingSpotDAO;
5  import com.parkit.parkingsystem.dao.TicketDAO;
6  import com.parkit.parkingsystem.model.ParkingSpot;
7  import com.parkit.parkingsystem.model.Ticket;
8  import com.parkit.parkingsystem.util.InputReaderUtil;
9  import edu.umd.cs.findbugs.annotations.SuppressFBWarnings;
10 import org.apache.logging.log4j.LogManager;
11 import org.apache.logging.log4j.Logger;
12
13 import java.util.ArrayList;
14 import java.util.Date;
15 import java.util.List;
16
17 public class ParkingService {
18
19     private static final Logger logger = LogManager.getLogger("ParkingService");
20
21     private static final FareCalculatorService fareCalculatorService = new FareCalculatorService();
22
23     private final InputReaderUtil inputReaderUtil;
24     private final ParkingSpotDAO parkingSpotDAO;
25     private final TicketDAO ticketDAO;
26
27     public ParkingService(InputReaderUtil inputReaderUtil, ParkingSpotDAO parkingSpotDAO, TicketDAO ticketDAO){
28         this.inputReaderUtil = inputReaderUtil;
29         this.parkingSpotDAO = parkingSpotDAO;
30         this.ticketDAO = ticketDAO;
31     }
32
33     @SuppressFBWarnings("DLS_DEAD_LOCAL_STORE")
34     public void processIncomingVehicle() {
35         try{
36             ParkingSpot parkingSpot = getNextParkingNumberIfAvailable();
37             if(parkingSpot !=null && parkingSpot.getId() > 0){
38                 String vehicleRegNumber = getVehicleRegNumber();
39                 parkingSpot.setAvailable(false);
40                 parkingSpotDAO.updateParking(parkingSpot);//allot this parking space and mark its availability as false
41                 Date inTime = new Date();
42                 ticketDAO.isRecurringClient(vehicleRegNumber);
43                 List<String> vehicles = new ArrayList<>();
44
45                 Ticket ticket = new Ticket();
46                 //ID, PARKING_NUMBER, VEHICLE_REG_NUMBER, RECURRING_CLIENT, PRICE, IN_TIME, OUT_TIME)
47                 //ticket.setID(ticketID);
48
49                 ticket.setParkingSpot(parkingSpot);
50                 ticket.setVehicleRegNumber(vehicleRegNumber);
51                 ticket.setPrice(0);
52                 ticket.setInTime(inTime);
53                 ticket.setOutTime(null);
54                 ticketDAO.saveTicket(ticket);
55                 System.out.println("Generated Ticket and saved in DB");
56                 System.out.println("Please park your vehicle in spot number:"+parkingSpot.getId());
```

Le rapport de couverture Jacoco

Couvrir un maximum de cas de figure pour sécuriser l'application

Rapport parkingsystem.util

InputReaderUtil

Coverage Summary for Package: com.parkit.parkingsystem.util

Package	Class, %	Method, %	Line, %
com.parkit.parkingsystem.util	100% (1/1)	75% (3/4)	23.5% (4/17)

Class	Class, %	Method, %	Line, %
InputReaderUtil	100% (1/1)	75% (3/4)	23.5% (4/17)

Coverage Summary for Class: InputReaderUtil (com.parkit.parkingsystem.util)

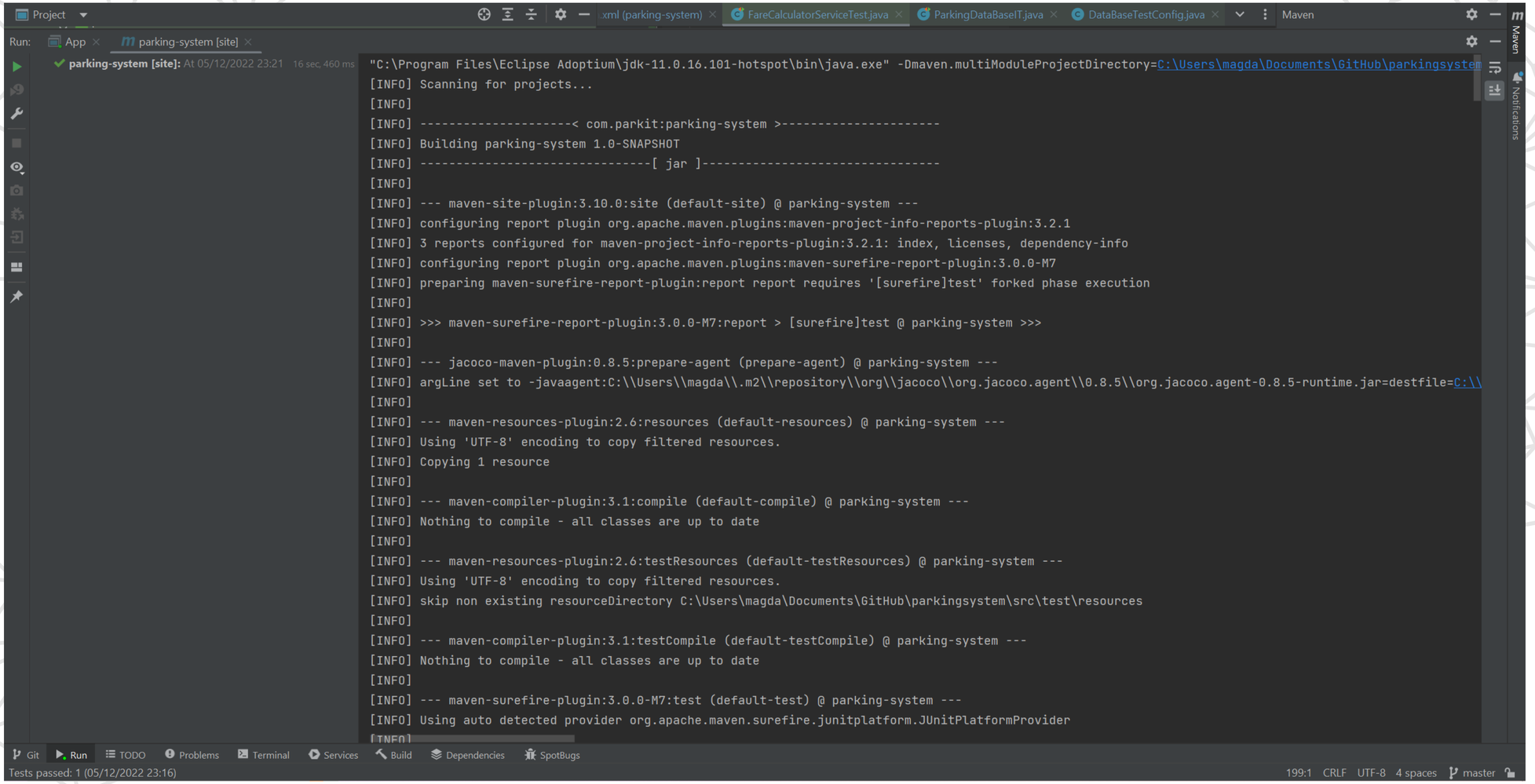
Class	Method, %	Line, %
InputReaderUtil	75% (3/4)	23.5% (4/17)
InputReaderUtil$MockitoMock$2074447737		
InputReaderUtil$MockitoMock$2074447737$auxiliary$ozrZ1prh		
InputReaderUtil$MockitoMock$2074447737$auxiliary$REjiT7pd		
InputReaderUtil$MockitoMock$2074447737$auxiliary$VrAOj8ih		
InputReaderUtil$MockitoMock$2074447737$auxiliary$YSCQvXu5		
Total	75% (3/4)	23.5% (4/17)

```
1  package com.parkit.parkingsystem.util;
2
3  import org.apache.logging.log4j.LogManager;
4  import org.apache.logging.log4j.Logger;
5
6  import java.util.Scanner;
7
8  public class InputReaderUtil {
9
10     private static Scanner scan = new Scanner(System.in);
11     private static final Logger logger = LogManager.getLogger("InputReaderUtil");
12
13     public int readSelection() {
14         try {
15             int input = Integer.parseInt(scan.nextLine());
16             return input;
17         }catch(Exception e){
18             logger.error("Error while reading user input from Shell", e);
19             System.out.println("Error reading input. Please enter valid number for proceeding further");
20             return -1;
21         }
22     }
23
24     public String readVehicleRegistrationNumber() throws Exception {
25         try {
26             String vehicleRegNumber= scan.nextLine();
27             if(vehicleRegNumber == null || vehicleRegNumber.trim().length()==0) {
28                 throw new IllegalArgumentException("Invalid input provided");
29             }
30             return vehicleRegNumber;
31         }catch(Exception e){
32             logger.error("Error while reading user input from Shell", e);
33             System.out.println("Error reading input. Please enter a valid string for vehicle registration number");
34             throw e;
35         }
36     }
37
38
39 }
```

Le rapport Surefire

Rapport d'exécution

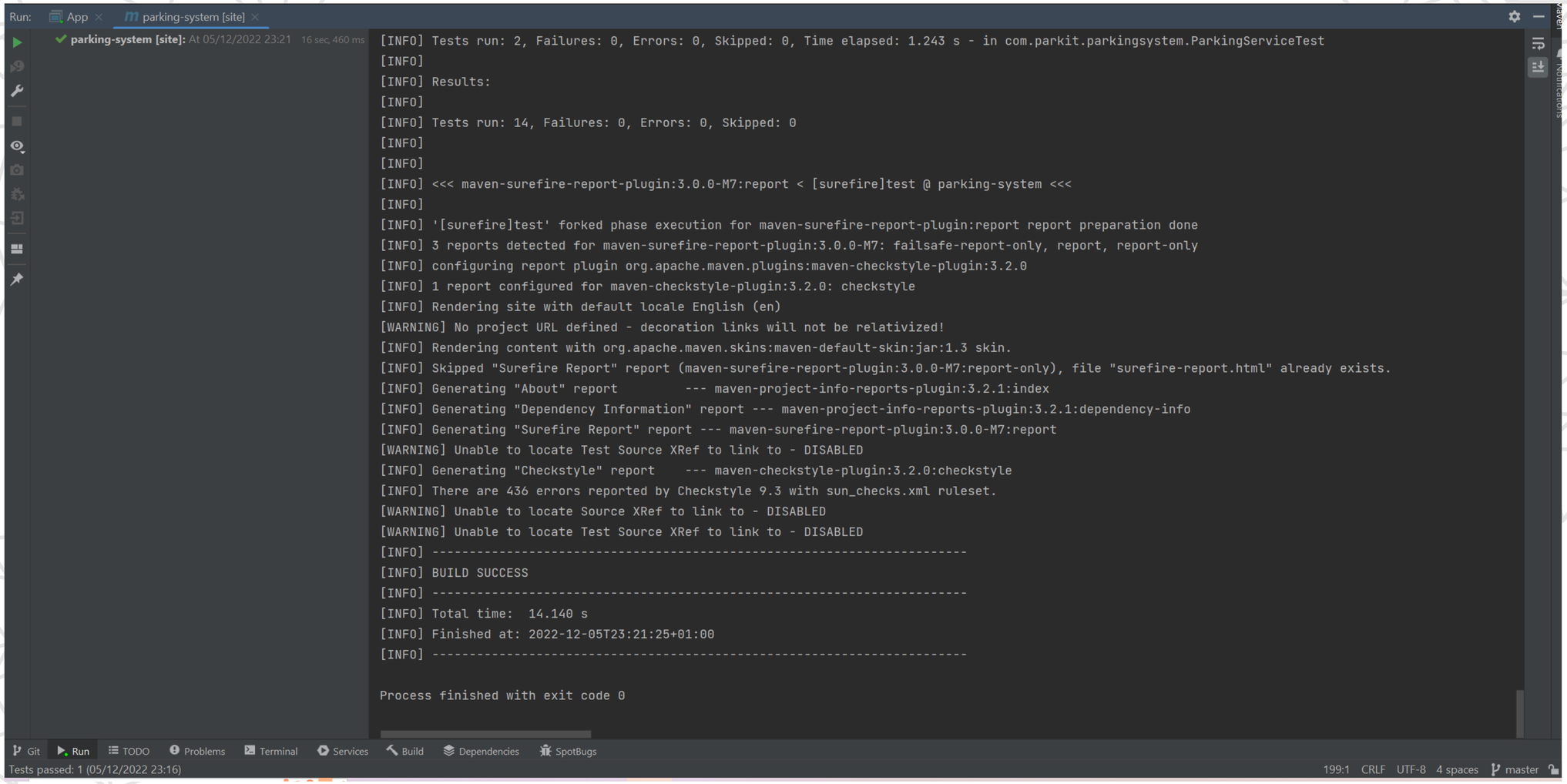
Sur le programme et les tests



Le rapport Surefire

Rapport d'exécution

Sur le programme et les tests



Magdalena Larmet

Salesforce
Developer