

Api3HaniniSProjet21Application.java

```
package be.condorcet.api3haninisprojet2_1;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class Api3HaniniSProjet21Application {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Api3HaniniSProjet21Application.class,  
args);
```

```
    }
```

```
}
```

Adresse.java

```
package be.condorcet.api3haninisprojet2_1.entities;

import jakarta.persistence.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@ToString
@Entity
@Table(name = "ADRESSE", schema = "ORA13", catalog =
"OCRL.CONDORCET.BE")
public class Adresse {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"adresse_generator")
    @SequenceGenerator(name = "adresse_generator", sequenceName =
"ADRESSE_ID_SEQ", allocationSize = 1)
    @Column(name = "IDADRESSE")
    private Integer idadresse;

    @NonNull
    @Column(name = "CP")
    private Integer cp;

    @NonNull
    @Column(name = "LOCALITE")
    private String localite;

    @NonNull
    @Column(name = "RUE")
    private String rue;

    @NonNull
    @Column(name = "NUM")
    private String num;
```

```
}
```

Client.java

```
package be.condorcet.api3haninisprojet2_1.entities;

import jakarta.persistence.*;
import lombok.*;
import com.fasterxml.jackson.annotation.JsonIgnore;
import java.util.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@ToString
@Entity
@Table(name = "CLIENT", schema = "ORA13", catalog = "OCRL.CONDORCET.BE")
public class Client {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"client_generator")
    @SequenceGenerator(name = "client_generator", sequenceName =
"CLIENT_ID_SEQ", allocationSize = 1)
    @Column(name = "IDCLIENT")
    private Integer idclient;

    @NonNull
    @Column(name = "MAIL")
    private String mail;

    @NonNull
    @Column(name = "NOM")
    private String nom;

    @NonNull
    @Column(name = "PRENOM")
    private String prenom;

    @NonNull
    @Column(name = "TEL")
    private String tel;

    @JsonIgnore
    @ToString.Exclude
    @OneToMany(mappedBy = "clientfk")
    private Set<Location> locations;
```



Location.java

```
package be.condorcet.api3haninisprojet2_1.entities;

import java.sql.Date;

import jakarta.persistence.*;
import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@ToString
@Entity
@Table(name = "LOCATION", schema = "ORA13", catalog =
"OCRL.CONDORCET.BE")
public class Location {

    @Getter
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"location_generator")
    @SequenceGenerator(name = "location_generator", sequenceName =
"LOCATION_ID_SEQ", allocationSize = 1)
    @Column(name = "IDLOCATION")
    private Integer idlocation;

    @NonNull
    @Column(name = "DATELOC")
    private Date dateloc;

    @NonNull
    @Column(name = "KMTOTAL")
    private Integer kmtotal;

    @NonNull
    @Column(name = "ACOMPTE")
    private Double acompte;

    @Getter
    @Column(name = "TOTAL")
    private Double total;

    @NonNull
    @ManyToOne
    @JoinColumn(name = "TAXIFK")
    private Taxi taxifk;

    @NonNull
```

```
@ManyToOne
@JoinColumn(name = "CLIENTFK")
private Client clientfk;

@NotNull
@ManyToOne
@JoinColumn(name = "ADRESSEDEPART")
private Adresse addressedepart;

@NotNull
@ManyToOne
@JoinColumn(name = "ADRESSEFIN")
private Adresse adressefin;
```



Taxi.java

```
package be.condorcet.api3haninisprojet2_1.entities;

import jakarta.persistence.*;
import lombok.*;
import com.fasterxml.jackson.annotation.JsonIgnore;

import java.util.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@ToString
@Entity
@Table(name = "TAXI", schema = "ORA13", catalog = "OCRL.CONDORCET.BE")
public class Taxi {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"taxi_generator")
    @SequenceGenerator(name = "taxi_generator", sequenceName =
"TAXI_ID_SEQ", allocationSize = 1)
    @Column(name = "IDTAXI")
    private Integer idtaxi;

    @NonNull
    @Column(name = "IMMATRICULATION")
    private String immatriculation;

    @NonNull
    @Column(name = "CARBURANT")
    private String carburant;

    @NonNull
    @Column(name = "PRIXKM")
    private Double prixkm;

    @JsonIgnore
    @ToString.Exclude
    @OneToMany(mappedBy = "taxifk")
    private List<Location> locations;
```



AdresseRepository.java

```
package be.condorcet.api3haninisprojet2_1.repositories;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface AdresseRepository extends JpaRepository<Adresse,
Integer>{

    List<Adresse> findByLocalite(String localite);
}
```

ClientRepository.java

```
package be.condorcet.api3haninisprojet2_1.repositories;

import java.util.List;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

@Repository
public interface ClientRepository extends JpaRepository<Client, Integer>
{

    Client findByNomAndPrenomAndTel(String nom, String prenom, String
tel);

    @Query(value = "SELECT l FROM Location l WHERE l.clientfk.idclient =
:clientId")
    List<Location> locationsForClient(@Param("clientId") Integer
clientId);
```


LocationRepository.java

```
package be.condorcet.api3haninisprojet2_1.repositories;

import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.entities.Client;

import java.sql.Date;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

@Repository
public interface LocationRepository extends JpaRepository<Location,
Integer>{

    @Query("SELECT DISTINCT l FROM Location l JOIN l.taxifk t WHERE
t.idtaxi = :taxiId AND l.dateloc BETWEEN :startDate AND :endDate")
    List<Location> findLocationsByTaxiIdAndDateRange(@Param("taxiId")
Integer taxiId, @Param("startDate") Date startDate, @Param("endDate")
Date endDate);

    List<Location> findByClientfk(Client cl);

    List<Location> findByTaxifk(Taxi taxi);

    @Query(value ="SELECT l FROM Location l WHERE l.dateloc = :date1")
    List<Location> findByDateloc(@Param("date1") Date date1);
```



TaxiRepository.java

```
package be.condorcet.api3haninisprojet2_1.repositories;

import java.util.List;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.entities.Location;

import org.springframework.stereotype.Repository;

@Repository
public interface TaxiRepository extends JpaRepository<Taxi, Integer> {

    Taxi findByImmatriculation(String immatriculation);

    @Query("SELECT DISTINCT c FROM Client c JOIN c.locations l JOIN l.taxifk t WHERE t.idtaxi = :taxiId")
    List<Client> clientsForTaxi(@Param("taxiId") Integer taxiId);

    @Query(value = "SELECT l FROM Location l WHERE l.taxifk.idtaxi = :taxiId")
    List<Location> locationsForTaxi(@Param("taxiId") Integer taxiId);

    @Query(value = "SELECT SUM(l.kmtotal) FROM Location l WHERE l.taxifk.idtaxi = :taxiId")
    Double totalKilometersForTaxi(@Param("taxiId") Integer taxiId);

    @Query(value = "SELECT SUM(l.total + l.acompte) FROM Location l JOIN Taxi t ON l.taxifk.idtaxi = t.idtaxi WHERE t.idtaxi = :taxiId")
    Double totalCostForTaxi(@Param("taxiId") Integer taxiId);
}
```

AdresseServiceImpl.java

```
package be.condorcet.api3haninisprojet2_1.services.adresse;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.repositories.AdresseRepository;

import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@Transactional(rollbackOn = Exception.class)
public class AdresseServiceImpl implements InterfAdresseService {

    @Autowired
    private AdresseRepository adresseRepository;

    @Override
    public Adresse create(Adresse adresse) throws Exception {
        adresse.setLocalite(adresse.getLocalite().toLowerCase());
        adresseRepository.save(adresse);
        return adresse;
    }

    @Override
    public Adresse read(Integer id) throws Exception {
        Optional<Adresse> ocl= adresseRepository.findById(id);
        return ocl.get();
    }

    @Override
    public Adresse update(Adresse adresse) throws Exception {
        adresse.setLocalite(adresse.getLocalite().toLowerCase());
        adresseRepository.save(adresse);
        return adresse;
    }

    @Override
    public void delete(Adresse adresse) throws Exception {
        adresseRepository.deleteById(adresse.getIdadresse());
    }

    @Override
    public List<Adresse> all() throws Exception {
        return adresseRepository.findAll();
    }
}
```

```
}
```

```
@Override
```

```
public List<Adresse> read(String localite) throws Exception {
```

```
    return adresseRepository.findByLocalite(localite.toLowerCase());
```

```
}
```

```
}
```

InterfAdresseService.java

```
package be.condorcet.api3haninisprojet2_1.services.adresse;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.services.InterfaceService;

import java.util.List;

public interface InterfAdresseService extends InterfaceService<Adresse>
{

    List<Adresse> read(String localite) throws Exception;
```

```
}
```

ClientServiceImpl.java

```
package be.condorcet.api3haninisprojet2_1.services.client;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.repositories.ClientRepository;

import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Locale;
import java.util.Optional;

@Service
@Transactional(rollbackOn = Exception.class)
public class ClientServiceImpl implements InterfClientService {

    @Autowired
    private ClientRepository clientRepository;

    @Override
    public Client create(Client client) throws Exception {
        client.setMail(client.getMail().toLowerCase(Locale.ROOT));
        client.setNom(client.getNom().toLowerCase(Locale.ROOT));
        client.setPrenom(client.getPrenom().toLowerCase(Locale.ROOT));
        clientRepository.save(client);
        return client;
    }

    @Override
    public Client read(Integer id) throws Exception {
        Optional<Client> ocl= clientRepository.findById(id);
        return ocl.get();
    }

    @Override
    public Client update(Client client) throws Exception {

        client.setMail(client.getMail().toLowerCase());
        client.setNom(client.getNom().toLowerCase());
        client.setPrenom(client.getPrenom().toLowerCase());
        clientRepository.save(client);
        return client;
    }

    @Override
    public void delete(Client client) throws Exception {
```

```
        clientRepository.deleteById(client.getIdclient());
```

```
    }
```

```
    @Override
```

```
    public List<Client> all() throws Exception {
```

```
        return clientRepository.findAll();
```

```
    }
```

```
    @Override
```

```
    public Client read(String nom, String prenom, String tel) throws  
Exception {
```

```
        return
```

```
        clientRepository.findByNomAndPrenomAndTel(nom.toLowerCase(),  
prenom.toLowerCase(), tel.toLowerCase());
```

```
    }
```

```
    @Override
```

```
    public List<Location> locationsForClient(Integer idClient) throws  
Exception {
```

```
        return clientRepository.locationsForClient(idClient);
```

```
    }
```

```
}
```

InterfClientService.java

```
package be.condorcet.api3haninisprojet2_1.services.client;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.services.InterfaceService;

import java.util.List;

public interface InterfClientService extends InterfaceService<Client>{

    Client read(String nom, String prenom, String tel) throws Exception;
    List<Location> locationsForClient(Integer idClient) throws
Exception;
```

```
}
```


InterfaceService.java

```
package be.condorcet.api3haninisprojet2_1.services;
```

```
import java.util.List;
```

```
public interface InterfaceService<T> {
```

```
    T create(T t) throws Exception;
```

```
    T read(Integer id) throws Exception;
```

```
    T update(T t) throws Exception;
```

```
    void delete(T t) throws Exception;
```

```
    List<T> all() throws Exception;
```

```
}
```

InterfLocationService.java

```
package be.condorcet.api3haninisprojet2_1.services.location;

import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.services.InterfaceService;

import java.util.List;
import java.sql.Date;

public interface InterfLocationService extends
InterfaceService<Location> {

    List<Location> read(Taxi t) throws Exception;

    List<Location> getLocationsByTaxiIdAndDateRange(Integer id, Date d1,
Date d2) throws Exception;

    List<Location> read(Client cl) throws Exception;
    List<Location> readByDate(Date date1) throws Exception;
```

LocationServiceImpl.java

```
package be.condorcet.api3haninisprojet2_1.services.location;

import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import
be.condorcet.api3haninisprojet2_1.repositories.LocationRepository;

import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.sql.Date;
import java.util.List;

@Service
@Transactional(rollbackOn = Exception.class)
public class LocationServiceImpl implements InterfLocationService{

    @Autowired
    private LocationRepository locationRepository;

    @Override
    public Location create(Location location) throws Exception {
        Location savedLocation = locationRepository.save(location);

        Location locationWithTotal =
locationRepository.findById(savedLocation.getIdlocation())
                .orElseThrow(() -> new RuntimeException("Location not
found"));

        Double total = locationWithTotal.getKmtotal() *
locationWithTotal.getTaxifk().getPrixkm();

        locationWithTotal.setTotal(total);

        locationRepository.save(locationWithTotal);

        return locationWithTotal;
    }

    @Override
    public Location read(Integer id) throws Exception {
        return locationRepository.findById(id).get();
    }

    @Override
    public Location update(Location location) throws Exception {
```

```
        Location existingLocation =  
locationRepository.findById(location.getIdlocation())  
                .orElseThrow(() -> new RuntimeException("Location not  
found"));
```

```
        existingLocation.setKmtotal(location.getKmtotal());
```

```
        Taxi currentTaxi = location.getTaxifk();
```

```
        double newTotal = existingLocation.getKmtotal() *  
currentTaxi.getPrixkm();
```

```
        location.setTotal(newTotal);
```

```
        locationRepository.save(location);
```

```
        return location;
```

```
}
```

```
@Override
```

```
public void delete(Location location) throws Exception {  
    locationRepository.deleteById(location.getIdlocation());
```

```
}
```

```
@Override
```

```
public List<Location> all() throws Exception {  
    return locationRepository.findAll();
```

```
}
```

```
@Override
```

```
public List<Location> read(Taxi t) throws Exception {  
    List<Location> ll = locationRepository.findByTaxifk(t);  
    return ll;
```

```
}
```

```
@Override
```

```
public List<Location> getLocationsByTaxiIdAndDateRange(Integer  
idtaxi, Date d1, Date d2) throws Exception {  
    return
```

```
locationRepository.findLocationsByTaxiIdAndDateRange(idtaxi, d1, d2);
```

```
}
```

```
@Override
```

```
public List<Location> read(Client client) {  
    List<Location> ll = locationRepository.findByClientfk(client);  
    return ll;
```

```
}
```

```
@Override
public List<Location> readByDate(Date datel) throws Exception {
    List<Location> ll = locationRepository.findByDateloc(datel);
    return ll;
}
```

```
}
```

InterfTaxiService.java

```
package be.condorcet.api3haninisprojet2_1.services.taxi;

import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.services.InterfaceService;
import be.condorcet.api3haninisprojet2_1.entities.Client;

import java.util.List;

public interface InterfTaxiService extends InterfaceService<Taxi>{

    Taxi getTaxiByImmatriculation(String immatriculation) throws
Exception;

    List<Client> clientsForTaxi(Integer idTaxi) throws Exception;

    List<Location> locationsForTaxi(Integer idTaxi) throws Exception;

    Double totalKilometersForTaxi(Integer idTaxi) throws Exception;

    Double totalCostForTaxi(Integer idTaxi) throws Exception;
```

```
}
```

TaxiServiceImpl.java

```
package be.condorcet.api3haninisprojet2_1.services.taxi;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import
be.condorcet.api3haninisprojet2_1.repositories.LocationRepository;
import be.condorcet.api3haninisprojet2_1.repositories.TaxiRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@Transactional(rollbackOn = Exception.class)
public class TaxiServiceImpl implements InterfTaxiService{

    @Autowired
    private TaxiRepository taxiRepository;

    @Autowired
    private LocationRepository locationRepository;

    @Override
    public Taxi create(Taxi taxi) throws Exception {
        taxiRepository.save(taxi);
        return taxi;
    }

    @Override
    public Taxi read(Integer id) throws Exception {
        Optional<Taxi> ocl= taxiRepository.findById(id);
        return ocl.get();
    }

    public Taxi update(Taxi taxi) throws Exception {
        read(taxi.getIdtaxi());
        taxiRepository.save(taxi);

        List<Location> locations =
locationRepository.findByTaxifk(taxi);

        for (Location location : locations) {
            double newTotal = location.getKmtotal() * taxi.getPrixkm();
            location.setTotal(newTotal);
        }
    }
}
```

```
locationRepository.saveAll(locations);  
return taxi;
```

```
}
```

```
@Override  
public void delete(Taxi taxi) throws Exception {  
    taxiRepository.deleteById(taxi.getIdtaxi());  
}
```

```
}
```

```
@Override  
public List<Taxi> all() throws Exception {  
    List<Taxi> taxis=taxiRepository.findAll();  
    return taxis;  
}
```

```
}
```

```
@Override  
public Taxi getTaxiByImmatriculation(String immatriculation) throws  
Exception {  
    return taxiRepository.findByImmatriculation(immatriculation);  
}
```

```
}
```

```
@Override  
public List<Client> clientsForTaxi(Integer idTaxi) throws Exception {  
    return taxiRepository.clientsForTaxi(idTaxi);  
}
```

```
}
```

```
@Override  
public List<Location> locationsForTaxi(Integer idTaxi) throws  
Exception {  
    return taxiRepository.locationsForTaxi(idTaxi);  
}
```

```
}
```

```
@Override  
public Double totalKilometersForTaxi(Integer idTaxi) throws  
Exception {  
    return taxiRepository.totalKilometersForTaxi(idTaxi);  
}
```

```
}
```

```
@Override  
public Double totalCostForTaxi(Integer idTaxi) throws Exception {  
    return taxiRepository.totalCostForTaxi(idTaxi);  
}
```

```
}
```

```
}
```


RestAdresse.java

```
package be.condorcet.api3haninisprojet2_1.webservices;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.services.adresse.InterfAdresseService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin(origins = "*", allowedHeaders = "*", exposedHeaders = "*")
@RestController
@RequestMapping("/adresses")
public class RestAdresse {
    @Autowired
    private InterfAdresseService AdresseServiceImpl;

    //-----Retrouver l'adresse correspondant à un id
    //-----
    @RequestMapping(value =("/{id}", method = RequestMethod.GET)
    public ResponseEntity<Adresse> getAdresse(@PathVariable(value =
"id") int id) throws Exception {
        System.out.println("recherche de l'adresse avec l'id " + id);
        Adresse adresse = AdresseServiceImpl.read(id);
        return new ResponseEntity<>(adresse, HttpStatus.OK);
    }

    //-----Retrouver les adresses portant une localité
    //-----
    @RequestMapping(value = "/localite/{localite}", method =
RequestMethod.GET)
    public ResponseEntity<List<Adresse>>
listAdressesLocalite(@PathVariable(value = "localite") String localite)
throws Exception {
        System.out.println("recherche de la localite " + localite);
        List<Adresse> adresses;
        adresses = AdresseServiceImpl.read(localite);
        return new ResponseEntity<>(adresses, HttpStatus.OK);
    }

    //-----Créer un adresse-----
    //-----
    @RequestMapping(value = "/create", method = RequestMethod.POST)
    public ResponseEntity<Adresse> createAdresse(@RequestBody Adresse
adresse) throws Exception {
```

```

        System.out.println("Cr ation de l'adresse " + adresse);
        AdresseServiceImpl.create(adresse);
        return new ResponseEntity<>(adresse, HttpStatus.OK);
    }

    //-----Mettre   jour une adresse avec un id donn -----
    -----
    @RequestMapping(value = "/" + id, method = RequestMethod.PUT)
    public ResponseEntity<Adresse> majAdresse(@PathVariable(value =
"id") int id, @RequestBody Adresse nouvAdresse) throws Exception {
        System.out.println("maj de adresse id = " + id);
        nouvAdresse.setIdadresse(id);
        Adresse adresse = AdresseServiceImpl.update(nouvAdresse);
        return new ResponseEntity<>(adresse, HttpStatus.OK);
    }

    //-----Effacer une adresse avec un id donn -----
    -----
    @RequestMapping(value = "/" + id, method = RequestMethod.DELETE)
    public ResponseEntity<Void> deleteAdresse(@PathVariable(value =
"id") int id) throws Exception {
        System.out.println("effacement du adresse d'id " + id);
        Adresse adresse = AdresseServiceImpl.read(id);
        AdresseServiceImpl.delete(adresse);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    //-----Retrouver tous les adresses -----
    -----
    @RequestMapping(value = "/all", method = RequestMethod.GET)
    public ResponseEntity<List<Adresse>> listAdresse() throws Exception
{
        System.out.println("recherche de tous les adresses");
        return new ResponseEntity<>(AdresseServiceImpl.all(),
HttpStatus.OK);
    }

    //-----G rer les erreurs-----
    -----
    @ExceptionHandler({Exception.class})
    public ResponseEntity<Void> handleIOException(Exception ex) {
        System.out.println("erreur : " + ex.getMessage());
        return ResponseEntity.notFound().header("error",
ex.getMessage()).build();
    }
}

```

RestClient.java

```
package be.condorcet.api3haninisprojet2_1.webservices;

import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import
be.condorcet.api3haninisprojet2_1.services.client.InterfClientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin(origins = "*", allowedHeaders = "*", exposedHeaders = "*")
@RestController
@RequestMapping("/clients")
public class RestClient {
    @Autowired
    private InterfClientService clientServiceImpl;

    //-----Retrouver le client correspondant à un id
    donnÃ©-----
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public ResponseEntity<Client> getClient(@PathVariable(value = "id")
int id) throws Exception {
        System.out.println("recherche du client d' id " + id);
        Client client = clientServiceImpl.read(id);
        return new ResponseEntity<>(client, HttpStatus.OK);
    }

    //-----Retrouver le client correspondant à un triplet
    (nom,prÃ©nom,tel) unique donnÃ©-----
    -----
    @RequestMapping(value = "{nom}/{prenom}/{tel}", method =
RequestMethod.GET)
    public ResponseEntity<Client> getClientUnique(@PathVariable(value =
"nom") String nom,
                                                    @PathVariable(value =
"prenom") String prenom,
                                                    @PathVariable(value =
"tel") String tel) throws Exception {
        System.out.println("recherche du client " + nom + " " + prenom +
" " + tel);
        Client client = clientServiceImpl.read(nom, prenom, tel);
        return new ResponseEntity<>(client, HttpStatus.OK);
    }
}
```

```

//-----Cr er un client-----
-----

@RequestMapping(value = "/create", method = RequestMethod.POST)
public ResponseEntity<Client> createClient(@RequestBody Client
client){
    try
        clientServiceImpl.create(client);
        return new ResponseEntity<>(client, HttpStatus.OK);
    catch (Exception e)
        e.printStackTrace();
        return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
    }

}

//-----Mettre   jour un client d'un id donn -----
-----

@RequestMapping(value = "/" + id, method = RequestMethod.PUT)
public ResponseEntity<Client> majClient(@PathVariable(value = "id")
int id, @RequestBody Client nouvcli) throws Exception {
    System.out.println("maj de client id = " + id);
    nouvcli.setIdclient(id);
    Client clact = clientServiceImpl.update(nouvcli);
    return new ResponseEntity<>(clact, HttpStatus.OK);
}

//-----Effacer un client d'un id donn -----
-----

@RequestMapping(value = "/" + id, method = RequestMethod.DELETE)
public ResponseEntity<Void> deleteClient(@PathVariable(value = "id")
int id) throws Exception {
    System.out.println("effacement du client d'id " + id);
    Client client = clientServiceImpl.read(id);
    clientServiceImpl.delete(client);
    return new ResponseEntity<>(HttpStatus.OK);
}

//-----Retrouver tous les clients -----
-----

@RequestMapping(value = "/all", method = RequestMethod.GET)
public ResponseEntity<List<Client>> listClient() throws Exception {
    System.out.println("recherche de tous les clients");
    return new ResponseEntity<>(clientServiceImpl.all(),
HttpStatus.OK);
}

```

```

//-----GÃ©rer les erreurs-----
-----

@ExceptionHandler({Exception.class})
public ResponseEntity<Void> handleIOException(Exception ex) {
    System.out.println("erreur : " + ex.getMessage());
    return ResponseEntity.notFound().header("error",
ex.getMessage()).build();
}

//-----Locations pour un client-----
-----

@RequestMapping(value="/identifiantloc/{id}",method =
RequestMethod.GET)
public ResponseEntity<List<Location>>
locationsForClient(@PathVariable(value = "id") int id) throws Exception
{
    List<Location> locations =
clientServiceImpl.locationsForClient(id);
    return new ResponseEntity<>(locations, HttpStatus.OK);
}
}

```

RestLocation.java

```
package be.condorcet.api3haninisprojet2_1.webservices;

import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import
be.condorcet.api3haninisprojet2_1.services.taxi.InterfTaxiService;
import
be.condorcet.api3haninisprojet2_1.services.client.InterfClientService;
import
be.condorcet.api3haninisprojet2_1.services.location.InterfLocationService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.sql.Date;

@CrossOrigin(origins = "*", allowedHeaders = "*", exposedHeaders = "*")
@RestController
@RequestMapping("/locations")
public class RestLocation {

    @Autowired
    private InterfLocationService locationServiceImpl;
    @Autowired
    private InterfTaxiService taxiServiceImpl;
    @Autowired
    private InterfClientService clientServiceImpl;

    //-----Retrouver la location correspondant à un n°
    donnÃ©-----
    @RequestMapping(value =("/{id}", method = RequestMethod.GET)
    public ResponseEntity<Location> getLocation(@PathVariable(value =
"id") int id) throws Exception {
        System.out.println("recherche de la location n° " + id);
        Location loc = locationServiceImpl.read(id);
        return new ResponseEntity<>(loc, HttpStatus.OK);
    }

    //-----Retrouver la location correspondant à un taxi
    donnÃ©-----
    @RequestMapping(value = "/idtaxi={id}", method = RequestMethod.GET)
    public ResponseEntity<List<Location>>
getLocationTaxi(@PathVariable(value = "id") int id) throws Exception {
```

```

        System.out.println("recherche des locations du taxi d'id " +
id);
        Taxi taxi = taxiServiceImpl.read(id);
        List<Location> lloc = locationServiceImpl.read(taxi);
        return new ResponseEntity<>(lloc, HttpStatus.OK);
    }

    //-----Retrouver la location correspondant à un
client donné-----
    @RequestMapping(value = "/idclient={id}", method =
RequestMethod.GET)
    public ResponseEntity<List<Location>>
getLocationClient(@PathVariable(value = "id") int id) throws Exception {
        System.out.println("recherche des locations du client d'id " +
id);
        Client cl = clientServiceImpl.read(id);
        List<Location> lloc = locationServiceImpl.read(cl);
        return new ResponseEntity<>(lloc, HttpStatus.OK);
    }

    //-----Retrouver la location correspondant à une
période et un taxi-----
    @RequestMapping(value =("/{id}/{d1}/{d2}", method =
RequestMethod.GET)
    public ResponseEntity<List<Location>>
getLocationBetweenDatesAndTaxi(
        @PathVariable(value = "id") int id,
        @PathVariable(value = "d1") Date d1,
        @PathVariable(value = "d2") Date d2
        ) throws Exception {
        System.out.println("Recherche des locations du taxi d'id " + id
+ " dans la période du " + d1 + " au " + d2);
        List<Location> locations =
locationServiceImpl.getLocationsByTaxiIdAndDateRange(id,d1, d2);

        return new ResponseEntity<>(locations, HttpStatus.OK);
    }

    //-----Créer une location-----
    @RequestMapping(value = "/create", method = RequestMethod.POST)
    public ResponseEntity<Location> createLocation(@RequestBody Location
loc) {
        try{
            System.out.println("Création de la location du taxi " +
loc.getTaxiId());
            System.out.println(loc);

```

```

        locationServiceImpl.create(loc);
        return new ResponseEntity<>(loc, HttpStatus.OK);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

//-----Retrouver toutes les locations -----
-----
    @RequestMapping(value = "/all", method = RequestMethod.GET)
    public ResponseEntity<List<Location>> listLocation() throws
Exception {
        System.out.println("recherche de toutes les locations");
        return new ResponseEntity<>(locationServiceImpl.all(),
HttpStatus.OK);
    }

//-----Mettre à jour une location-----
-----
    @RequestMapping(value = "/" + "{id}", method = RequestMethod.PUT)
    public ResponseEntity<Location> majLocationTaxi(@PathVariable(value
= "id") int id, @RequestBody Location newloc) throws Exception {
        System.out.println("maj de la location n° " + id);
        Location locact = locationServiceImpl.update(newloc);
        return new ResponseEntity<>(locact, HttpStatus.OK);
    }

//-----Effacer une location d'un id donné-----
-----
    @RequestMapping(value = "/" + "{id}", method = RequestMethod.DELETE)
    public ResponseEntity<Void> deleteLocation(@PathVariable(value =
"id") int id) throws Exception {
        System.out.println("effacement de la location n°" + id);
        Location loc = locationServiceImpl.read(id);
        locationServiceImpl.delete(loc);
        return new ResponseEntity<>(HttpStatus.OK);
    }

//-----Retrouver les locations pour une date-----
-----
    @RequestMapping(value = "/datelocation/{datel}", method =
RequestMethod.GET)
    public ResponseEntity<List<Location>>
getLocationDate(@PathVariable(value = "datel") Date datel) throws
Exception {
        System.out.println("recherche des locations de la date " +
datel);

```



```
List<Location> lloc = locationServiceImpl.readByDate(date1);  
return new ResponseEntity<>(lloc, HttpStatus.OK);
```

```
}
```

```
//-----GÃ©rer les erreurs-----
```

```
-----
```

```
@ExceptionHandler({Exception.class})
```

```
public ResponseEntity<Void> handleIOException(Exception ex) {
```

```
    System.out.println("erreur : " + ex.getMessage());
```

```
    return ResponseEntity.notFound().header("error",  
ex.getMessage()).build();
```

```
}
```

```
}
```

RestTaxi.java

```
package be.condorcet.api3haninisprojet2_1.webservices;

import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import
be.condorcet.api3haninisprojet2_1.services.taxi.InterfTaxiService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;

import java.util.ArrayList;
import java.util.List;

@CrossOrigin(origins = "*", allowedHeaders = "Content-Type",
exposedHeaders = "*")
@RestController
@RequestMapping("/taxis")
public class RestTaxi {
    @Autowired
    private InterfTaxiService TaxiServiceImpl;

    //-----Retrouver le taxi correspondant à un id
    //-----
    @RequestMapping(value = "/" + "id", method = RequestMethod.GET)
    public ResponseEntity<Taxi> getTaxi(@PathVariable(value = "id") int
id) throws Exception {
        System.out.println("recherche de le taxi avec l'id " + id);
        Taxi taxi = TaxiServiceImpl.read(id);
        return new ResponseEntity<>(taxi, HttpStatus.OK);
    }

    @RequestMapping(value="/immatriculation/" + "immatriculation",method =
RequestMethod.GET)
    public ResponseEntity<Taxi>
getTaxiByImmatriculation(@PathVariable(value = "immatriculation") String
immatriculation) throws Exception {
        Taxi
taxi=TaxiServiceImpl.getTaxiByImmatriculation(immatriculation);
        return new ResponseEntity<>(taxi, HttpStatus.OK);
    }

    //-----Clients pour un taxi-----
    //-----
    @RequestMapping(value="/identifiant/" + "id",method =
RequestMethod.GET)
    public ResponseEntity<List<Client>>
```

```

clientsForTaxi(@PathVariable(value = "id") int id) throws Exception {
    List<Client> clients =TaxiServiceImpl.clientsForTaxi(id);
    return new ResponseEntity<>(clients, HttpStatus.OK);
}

//-----Location pour un taxi-----
-----

@RequestMapping(value="/identifiantloc/{id}",method =
RequestMethod.GET)
public ResponseEntity<List<Location>>
locationsForTaxi(@PathVariable(value = "id") int id) throws Exception {
    List<Location> locations =TaxiServiceImpl.locationsForTaxi(id);
    return new ResponseEntity<>(locations, HttpStatus.OK);
}

//-----Nb total km pour un taxi-----
-----

@RequestMapping(value="/identifiantkmtot",method =
RequestMethod.POST)
public ResponseEntity<List<Double>>
totalKilometersForTaxi(@RequestBody List<Integer> idtaxis) throws
Exception {
    List<Double> kmtots = new ArrayList<>();
    for(int idtaxi: idtaxis){
        Double totalkm
=TaxiServiceImpl.totalKilometersForTaxi(idtaxi);
        kmtots.add(totalkm);
    }

    return new ResponseEntity<>(kmtots, HttpStatus.OK);
}

//-----Nb total km pour un taxi-----
-----

@RequestMapping(value="/identifiantmontant",method =
RequestMethod.POST)
public ResponseEntity<List<Double>> totalCostForTaxi(@RequestBody
List<Integer> idtaxis) throws Exception {
    List<Double> monttots = new ArrayList<>();
    for(int idtaxi: idtaxis){
        Double montttot=TaxiServiceImpl.totalCostForTaxi(idtaxi);
        monttots.add(montttot);
    }

    return new ResponseEntity<>(monttots, HttpStatus.OK);
}

//-----Cr  er un taxi-----
-----

@RequestMapping(value = "/create", method = RequestMethod.POST)

```

```

public ResponseEntity<Taxi> createTaxi(@RequestBody Taxi taxi){
    try
        TaxiServiceImpl.create(taxi);
        return new ResponseEntity<>(taxi, HttpStatus.OK);
    }catch (Exception e){
        e.printStackTrace();
        return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
    }

}

//-----Mettre à jour un taxi avec un id donné-----
-----
@RequestMapping(value = "/" + id, method = RequestMethod.PUT)
public ResponseEntity<Taxi> majTaxi(@PathVariable(value = "id") int
id, @RequestBody Taxi nouvTaxi) throws Exception {
    System.out.println("maj de taxi id = " + id);
    nouvTaxi.setIdtaxi(id);
    Taxi taxi = TaxiServiceImpl.update(nouvTaxi);
    return new ResponseEntity<>(taxi, HttpStatus.OK);
}

//-----Effacer un taxi avec un id donné-----
-----
@RequestMapping(value = "/" + id, method = RequestMethod.DELETE)
public ResponseEntity<Void> deleteTaxi(@PathVariable(value = "id")
int id) throws Exception {
    System.out.println("effacement du taxi d'id " + id);
    Taxi taxi = TaxiServiceImpl.read(id);
    TaxiServiceImpl.delete(taxi);
    return new ResponseEntity<>(HttpStatus.OK);
}

//-----Retrouver tous les taxis -----
-----
@RequestMapping(value = "/all", method = RequestMethod.GET)
public ResponseEntity<List<Taxi>> listTaxi() throws Exception {
    System.out.println("recherche de tous les taxis");
    return new ResponseEntity<>(TaxiServiceImpl.all(),
HttpStatus.OK);
}

//-----Gérer les erreurs-----
-----
@ExceptionHandler({Exception.class})
public ResponseEntity<Void> handleIOException(Exception ex) {
    System.out.println("erreur : " + ex.getMessage());
    return ResponseEntity.notFound().header("error",

```

```
ex.getMessage()).build();
```

```
}
```

```
}
```

Api3HaniniSProjet21ApplicationTests.java

```
package be.condorcet.api3haninisprojet2_1;
```

```
import org.junit.jupiter.api.Test;
```

```
import org.springframework.boot.test.context.SpringBootTest;
```

```
@SpringBootTest
```

```
class Api3HaniniSProjet21ApplicationTests {
```

```
    @Test
```

```
    void contextLoads() {
```

```
    }
```

```
}
```

AdresseServiceImplTest.java

```
package be.condorcet.api3haninisprojet2_1.services.adresse;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class AdresseServiceImplTest {

    @Autowired
    private InterfAdresseService adresseServiceImpl;

    private Adresse adresse;

    @BeforeEach
    void setUp() throws Exception {
        adresse = new Adresse(1200, "Baudour", "Rue des parapluies",
"5");
        adresse = adresseServiceImpl.create(adresse);
    }

    @AfterEach
    void tearDown() throws Exception {
        if (adresse != null && adresse.getIdadresse() != null) {
            adresseServiceImpl.delete(adresse);
            assertThrows(Exception.class, () ->
adresseServiceImpl.read(adresse.getIdadresse()),
"L'adresse ne devrait plus Ãªtre accessible aprÃªs
suppression");
        }
    }

    @Test
    void create() {
        assertNotNull(adresse.getIdadresse(), "L'ID de l'adresse devrait
Ãªtre gÃ©nÃ©rÃ© aprÃªs la crÃ©ation");
        assertAll("VÃ©rification de la crÃ©ation de l'adresse",
        () -> assertEquals(0, adresse.getIdadresse(), "L'ID
de l'adresse ne devrait pas Ãªtre zÃ©ro"),
        () -> assertEquals(1200, adresse.getCp(), "Le code
postal est incorrect"),
```

```

        () -> assertEquals("Baudour", adresse.getLocalite(), "La
localit   est incorrecte"),
        () -> assertEquals("Rue des parapluies",
adresse.getRue(), "La rue est incorrecte"),
        () -> assertEquals("5", adresse.getNum(), "Le num  ro
est incorrect")
    );
}

```

```

@Test
void read() throws Exception {
    Adresse adresse2 =
adresseServiceImpl.read(adresse.getIdadresse());
    assertNotNull(adresse2, "L'adresse lue ne devrait pas   tre
null");

    assertAll("V  rification des donn  es de l'adresse lue",
        () -> assertEquals(adresse.getCp(), adresse2.getCp(),
"Le code postal est incorrect"),
        () -> assertEquals(adresse.getLocalite(),
adresse2.getLocalite(), "La localit   est incorrecte"),
        () -> assertEquals(adresse.getRue(), adresse2.getRue(),
"La rue est incorrecte"),
        () -> assertEquals(adresse.getNum(), adresse2.getNum(),
"Le num  ro est incorrect")
    );
}

```

```

@Test
void rechLocalite() throws Exception {
    List<Adresse> adresses = adresseServiceImpl.read("Baudour");
    assertTrue(adresses.stream().anyMatch(a ->
"Baudour".equals(a.getLocalite()))),
        "Aucune adresse trouv  e pour la localit  
sp  cifi  e");
}

```

```

@Test
void all() throws Exception {
    List<Adresse> adresses = adresseServiceImpl.all();
    assertFalse(adresses.isEmpty(), "Aucune adresse trouv  e");
}

```

```

@Test
void update() throws Exception {
    adresse.setCp(1000);
    adresse.setLocalite("Bruxelles-Ville");
    adresse.setRue("Rue de la Loi");
    adresse.setNum("1");
}

```



```

        Adresse updatedAdresse = adresseServiceImpl.update(adresse);

        assertAll("Vérification de la mise à jour de l'adresse",
            () -> assertEquals(1000, updatedAdresse.getCp(), "Le
code postal après mise à jour est incorrect"),
            () -> assertEquals("Bruxelles-Ville",
updatedAdresse.getLocalite(), "La localité après mise à jour est
incorrecte"),
            () -> assertEquals("Rue de la Loi",
updatedAdresse.getRue(), "La rue après mise à jour est incorrecte"),
            () -> assertEquals("1", updatedAdresse.getNum(), "Le
numéro après mise à jour est incorrect")
        );
    }

```

```

@Test
void delete() throws Exception {
    adresseServiceImpl.delete(adresse);
    assertThrows(Exception.class, () ->
adresseServiceImpl.read(adresse.getIdadresse()),
        "L'adresse devrait être supprimée et inaccessible");
}

```

ClientServiceImplTest.java

```
package be.condorcet.api3haninisprojet2_1.services.client;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import
be.condorcet.api3haninisprojet2_1.services.adresse.AdresseServiceImpl;
import
be.condorcet.api3haninisprojet2_1.services.location.LocationServiceImpl;
import be.condorcet.api3haninisprojet2_1.services.taxi.TaxiServiceImpl;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

import java.sql.Date;
import java.time.LocalDate;
import java.util.List;

@SpringBootTest
class ClientServiceImplTest {

    @Autowired
    private ClientServiceImpl clientService;

    private Client client;

    @Autowired
    private LocationServiceImpl locationService;

    @Autowired
    private TaxiServiceImpl taxiServiceImpl;

    @Autowired
    private AdresseServiceImpl adresseService;

    private Location location;

    private Taxi taxi;
    private Adresse adDebut;
    private Adresse adFin;

    @BeforeEach
    void setUp() throws Exception {
```

```

        client = new Client("testmail@test.com", "TestNom",
"TestPrenom", "123");
        client = clientService.create(client);

        taxi = new Taxi("TEST-A12", "Essence", 10.0);
        taxiServiceImpl.create(taxi);

        adDebut = new Adresse(7000, "Mons", "Rue des arbres", "1A");
        adresseService.create(adDebut);

        adFin = new Adresse(7300, "Saint-Ghislain", "Rue des rochers",
"34");
        adresseService.create(adFin);

        Date date = Date.valueOf(LocalDate.now().toString());
        location = new Location(date, 30, 25.0, taxi, client, adDebut,
adFin);
        locationService.create(location);
    }

    @AfterEach
    void tearDown() throws Exception {
        if (client != null && client.getIdclient() != null) {
            clientService.delete(client);
            assertThrows(Exception.class, () ->
clientService.read(client.getIdclient()),
                "Client should be deleted and not retrievable");
        }
        locationService.delete(location);
        taxiServiceImpl.delete(taxi);
        adresseService.delete(adDebut);
        adresseService.delete(adFin);
    }

    @Test
    void create() {
        assertNotNull(client.getIdclient(), "Client ID should not be
null after creation");
        assertAll("Client creation verification",
            () -> assertEquals(0, client.getIdclient(), "Client
ID should be generated"),
            () -> assertEquals("testmail@test.com",
client.getMail(), "Email should match"),
            () -> assertEquals("TestNom", client.getNom(), "Name
should match"),
            () -> assertEquals("TestPrenom", client.getPrenom(),
"First name should match"),
            () -> assertEquals("123", client.getTel(), "Phone number
should match")
        );
    }

```

```

    );
}

@Test
void read() throws Exception {
    Client retrievedClient =
clientService.read(client.getIdclient());
    assertNotNull(retrievedClient, "Client should be retrievable by
ID");

```

```

        assertAll("Client retrieval verification",
            () -> assertEquals(client.getMail(),
retrievedClient.getMail(), "Emails should match"),
            () -> assertEquals(client.getNom(),
retrievedClient.getNom(), "Names should match"),
            () -> assertEquals(client.getPrenom(),
retrievedClient.getPrenom(), "First names should match"),
            () -> assertEquals(client.getTel(),
retrievedClient.getTel(), "Phone numbers should match")
        );
}

```

```

@Test
void update() throws Exception {
    client.setMail("newemail@test.com");
    client.setNom("NewNom");
    client.setPrenom("NewPrenom");
    client.setTel("456");
    client = clientService.update(client);

    assertNotNull(client, "Updated client should not be null");

    assertAll("Client update verification",
        () -> assertEquals("newemail@test.com",
client.getMail(), "Email should be updated"),
        () -> assertEquals("NewNom", client.getNom(), "Name
should be updated"),
        () -> assertEquals("NewPrenom", client.getPrenom(),
"First name should be updated"),
        () -> assertEquals("456", client.getTel(), "Phone number
should be updated")
    );
}

```

```

@Test
void delete() throws Exception {
    clientService.delete(client);
    assertThrows(Exception.class, () ->
clientService.read(client.getIdclient()),

```

```

        "Deleted client should not be retrievable");
        client = null;
    }

    @Test
    void all() throws Exception {
        List<Client> clients = clientService.all();
        assertFalse(clients.isEmpty(), "Client list should not be
empty");
        boolean containsClient = clients.stream().anyMatch(c ->
c.getIdclient().equals(client.getIdclient()));

        assertTrue(containsClient, "Client list should contain the
created client with matching ID");
    }

    @Test
    void readByFullnameAndPhoneNumber() throws Exception {

        Client retrievedClient = clientService.read("TestNom",
"TestPrenom", "123");

        assertNotNull(retrievedClient, "Client should be retrievable by
full name and phone number");

        assertAll("Client retrieval by full name and phone number
verification",
            () -> assertEquals("testmail@test.com",
retrievedClient.getMail(), "Emails should match"),
            () -> assertEquals("TestNom", retrievedClient.getNom(),
"Names should match"),
            () -> assertEquals("TestPrenom",
retrievedClient.getPrenom(), "First names should match"),
            () -> assertEquals("123", retrievedClient.getTel(),
"Phone numbers should match")
        );
    }

    @Test
    void locationsForClient() throws Exception {
        List<Location> locs =
clientService.locationsForClient(client.getIdclient());
        assertNotNull(locs, "List of locations should not be null.");
        assertFalse(locs.isEmpty(), "List of locations for the taxi
should not be empty.");
    }

```


LocationServiceImplTest.java

```
package be.condorcet.api3haninisprojet2_1.services.location;

import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import
be.condorcet.api3haninisprojet2_1.services.adresse.AdresseServiceImpl;
import
be.condorcet.api3haninisprojet2_1.services.client.ClientServiceImpl;
import be.condorcet.api3haninisprojet2_1.services.taxi.TaxiServiceImpl;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.sql.Date;
import java.time.LocalDate;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class LocationServiceImplTest {

    @Autowired
    private LocationServiceImpl locationService;

    @Autowired
    private TaxiServiceImpl taxiService;

    @Autowired
    private AdresseServiceImpl adresseService;

    @Autowired
    private ClientServiceImpl clientService;

    private Location location;
    private Client client;
    private Taxi taxi;
    private Adresse adDebut;
    private Adresse adFin;

    @BeforeEach
    void setUp() throws Exception {
```

```

        adDebut = new Adresse(7000, "Mons", "Rue des arbres", "1A");
        adFin = new Adresse(7300, "Saint-Ghislain", "Rue des rochers",
"34");

        adresseService.create(adDebut);
        adresseService.create(adFin);

        taxi = new Taxi("T-000-EST", "ESSENCE", 10.0);
        taxiService.create(taxi);

        client = new Client("clienttest@gmail.com", "TestNom",
"TestPrenom", "048476378");
        clientService.create(client);

        Date date=Date.valueOf(LocalDate.now().toString());
        location = new Location(date, 30, 25.0, taxi, client, adDebut,
adFin);
        locationService.create(location);
    }

    @AfterEach
    void tearDown() throws Exception {
        locationService.delete(location);
        clientService.delete(location.getClientfk());
        taxiService.delete(location.getTaxifk());
        adresseService.delete(location.getAdressedepart());
        adresseService.delete(location.getAdressefin());
    }

    @Test
    void create() {
        Date date=Date.valueOf(LocalDate.now().toString());
        assertNotEquals(0, location.getIdlocation(), "Location ID not
incremented");
        assertEquals(date, location.getDateloc(), "Location date not set
correctly");
        assertEquals(location.getTaxifk().getIdtaxi(), taxi.getIdtaxi(),
"Taxi not set correctly");
        assertEquals(location.getClientfk().getIdclient(),
client.getIdclient(), "Client not set correctly");
    }

    @Test
    void read() throws Exception {
        Location fetchedLocation =
locationService.read(location.getIdlocation());
        assertEquals(location.getIdlocation(),
fetchedLocation.getIdlocation(), "Fetched location ID does not match");
    }

```



```

@Test
void update() throws Exception {
    location.setKmtotal(50);
    Location updatedLocation = locationService.update(location);

    assertEquals(50, updatedLocation.getKmtotal(), "Km total not
updated correctly");
}

@Test
void delete() throws Exception {
    locationService.delete(location);

    assertThrows(Exception.class, () ->
locationService.read(location.getIdlocation()), "Location not deleted");
}

@Test
void all() throws Exception {
    List<Location> locations = locationService.all();
    assertEquals(0, locations.size(), "No locations found in the
database");
}

@Test
void readByDatesAndTaxi() throws Exception {

    Date start =
Date.valueOf(location.getDateloc().toLocalDate().minusDays(10));
    Date end =
Date.valueOf(location.getDateloc().toLocalDate().plusDays(10));

    List<Location> locations =
locationService.getLocationsByTaxiIdAndDateRange(taxi.getIdtaxi(),
start, end);
    assertEquals(0, locations.size(), "No locations found between
the given dates for the specified taxi");
}

@Test
void readByDate() throws Exception {

    Date datel = Date.valueOf(location.getDateloc().toLocalDate());
    List<Location> locations = locationService.readByDate(datel);
    assertEquals(0, locations.size(), "No locations found between
the given date");
}

@Test

```

```
void readByTaxi() throws Exception {
```

```
    List<Location> locations = locationService.read(taxi);  
    assertEquals(0, locations.size(), "No locations found for the  
given taxi");  
}
```

```
@Test
```

```
void readByClient() throws Exception {
```

```
    List<Location> locations = locationService.read(client);  
    assertEquals(0, locations.size(), "No locations found for the  
given client");  
}
```

```
}
```

TaxiServiceImplTest.java

```
package be.condorcet.api3haninisprojet2_1.services.taxi;

import be.condorcet.api3haninisprojet2_1.entities.Adresse;
import be.condorcet.api3haninisprojet2_1.entities.Client;
import be.condorcet.api3haninisprojet2_1.entities.Location;
import be.condorcet.api3haninisprojet2_1.entities.Taxi;
import
be.condorcet.api3haninisprojet2_1.services.adresse.AdresseServiceImpl;
import
be.condorcet.api3haninisprojet2_1.services.client.ClientServiceImpl;
import
be.condorcet.api3haninisprojet2_1.services.location.LocationServiceImpl;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.time.LocalDate;
import java.util.List;
import java.sql.Date;

import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class TaxiServiceImplTest {

    @Autowired
    private LocationServiceImpl locationService;

    @Autowired
    private TaxiServiceImpl taxiServiceImpl;

    @Autowired
    private AdresseServiceImpl adresseService;

    @Autowired
    private ClientServiceImpl clientService;

    private Location location;
    private Client client;
    private Taxi taxi;
    private Adresse adDebut;
    private Adresse adFin;

    @BeforeEach
    void setUp() throws Exception {
```

```

        taxi = new Taxi("TEST-A12", "Essence", 10.0);
        taxiServiceImpl.create(taxi);

        adDebut = new Adresse(7000, "Mons", "Rue des arbres", "1A");
        adresseService.create(adDebut);

        adFin = new Adresse(7300, "Saint-Ghislain", "Rue des rochers",
"34");
        adresseService.create(adFin);

        client = new Client("clienttest@gmail.com", "TestNom",
"TestPrenom", "048476378");
        clientService.create(client);

        Date date = Date.valueOf(LocalDate.now().toString());
        location = new Location(date, 30, 25.0, taxi, client, adDebut,
adFin);
        locationService.create(location);
    }

    @AfterEach
    void tearDown() throws Exception {
        locationService.delete(location);
        clientService.delete(client);
        taxiServiceImpl.delete(taxi);
        adresseService.delete(adDebut);
        adresseService.delete(adFin);
    }

    @Test
    void create() throws Exception {
        Taxi newTaxi = new Taxi("CREATE-TEST", "Diesel", 15.0);
        taxiServiceImpl.create(newTaxi);

        Taxi retrievedTaxi =
taxiServiceImpl.getTaxiByImmatriculation("CREATE-TEST");
        assertNotNull(retrievedTaxi, "Taxi should be created and
retrieved.");
        assertEquals("CREATE-TEST", retrievedTaxi.getImmatriculation(),
"Immatriculation should match.");

        taxiServiceImpl.delete(newTaxi);
    }

    @Test
    void delete() throws Exception {
        Taxi newTaxi = new Taxi("DELETE-TEST", "Electric", 20.0);
        taxiServiceImpl.create(newTaxi);

```

```

        int taxiId = newTaxi.getIdtaxi();
        taxiServiceImpl.delete(newTaxi);

        assertThrows(Exception.class, () ->
taxiServiceImpl.read(taxiId, "Taxi should be deleted.");
    }

    @Test
    void read() throws Exception {
        Taxi retrievedTaxi = taxiServiceImpl.read(taxi.getIdtaxi());
        assertNotNull(retrievedTaxi, "Taxi should be retrievable by
ID.");
        assertEquals(taxi.getIdtaxi(), retrievedTaxi.getIdtaxi(), "Taxi
ID should match.");
    }

    @Test
    void update() throws Exception {
        taxi.setImmatriculation("T-002-EST");
        taxi.setCarburant("Diesel");
        taxi.setPrixkm(2.0);
        Taxi updatedTaxi = taxiServiceImpl.update(taxi);

        assertEquals("T-002-EST", updatedTaxi.getImmatriculation(),
"Immatriculation should be updated.");
        assertEquals("Diesel", updatedTaxi.getCarburant(), "Carburant
should be updated.");
        assertEquals(2.0, updatedTaxi.getPrixkm(), "Prix/km should be
updated.");
    }

    @Test
    void all() throws Exception {
        List<Taxi> taxis = taxiServiceImpl.all();
        assertNotNull(taxis, "List of taxis should not be null.");
        assertFalse(taxis.isEmpty(), "List of taxis should not be
empty.");
    }

    @Test
    void clientsForTaxi() throws Exception {
        List<Client> clients =
taxiServiceImpl.clientsForTaxi(taxi.getIdtaxi());
        assertNotNull(clients, "List of clients should not be null.");
        assertFalse(clients.isEmpty(), "List of clients for the taxi
should not be empty.");
    }

```

```

@Test
void readByImmatriculation() throws Exception {
    Taxi retrievedTaxi =
taxiServiceImpl.getTaxiByImmatriculation(taxi.getImmatriculation());
    assertNotNull(retrievedTaxi, "Taxi should be retrievable by
Immatriculation.");
    assertEquals(taxi.getImmatriculation(),
retrievedTaxi.getImmatriculation(), "Taxi immatriculation should
match.");
}

@Test
void locationsForTaxi() throws Exception {
    List<Location> locs =
taxiServiceImpl.locationsForTaxi(taxi.getIdtaxi());
    assertNotNull(locs, "List of locations should not be null.");
    assertFalse(locs.isEmpty(), "List of locations for the taxi
should not be empty.");
}

@Test
void totKmTaxi() throws Exception {
    Location location1 = new
Location(Date.valueOf(LocalDate.now().minusDays(1)), 10, 15.0, taxi,
client, adDebut, adFin);
    Location location2 = new
Location(Date.valueOf(LocalDate.now().minusDays(2)), 20, 25.0, taxi,
client, adDebut, adFin);

    locationService.create(location1);
    locationService.create(location2);

    double expectedTotalKm = location.getKmtotal() +
location1.getKmtotal() + location2.getKmtotal();

    Double retrievedTotalKm =
taxiServiceImpl.totalKilometersForTaxi(taxi.getIdtaxi());

    assertNotNull(retrievedTotalKm, "Total kilometers should not be
null.");
    assertEquals(expectedTotalKm, retrievedTotalKm, "Total
kilometers should match the expected value.");

    locationService.delete(location1);
    locationService.delete(location2);
}

@Test
void totalCostForTaxi() throws Exception {

```

```
        Location location1 = new
Location(Date.valueOf(LocalDate.now().minusDays(1)), 10, 100.0, taxi,
client, adDebut, adFin);
        Location location2 = new
Location(Date.valueOf(LocalDate.now().minusDays(2)), 20, 150.0, taxi,
client, adDebut, adFin);

        location1.setTotal(100.0);
        location1.setAcompte(10.0);

        location2.setTotal(150.0);
        location2.setAcompte(20.0);

        locationService.create(location1);
        locationService.create(location2);

        double expectedTotalCost = (location.getTotal() +
location.getAcompte()) +
                (location1.getTotal() + location1.getAcompte()) +
                (location2.getTotal() + location2.getAcompte());

        Double retrievedTotalCost =
taxiServiceImpl.totalCostForTaxi(taxi.getIdtaxi());

        assertNotNull(retrievedTotalCost, "Total cost should not be
null.");
        assertEquals(expectedTotalCost, retrievedTotalCost, "Total cost
should match the expected value.");

        locationService.delete(location1);
        locationService.delete(location2);
    }
}
```

```
}
```

