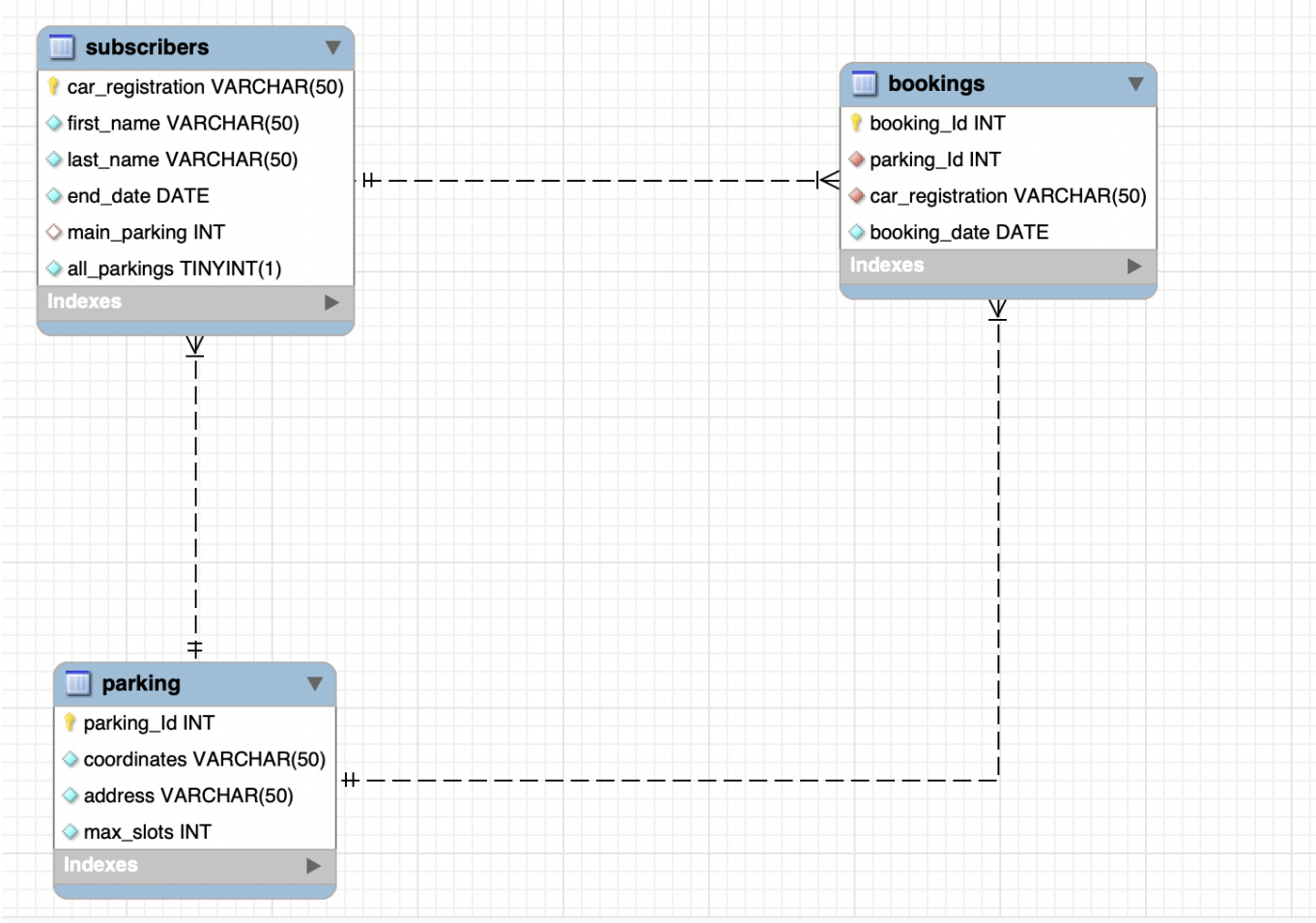
Parking Spot Booking System

Skład: Wojciech Orłowski, Ariel Michalik
Serwer baz danych: MySQL Backend: Java/Hibernate
Temat: System rezerwacji miejsc parkingowych
Opis: System rozwiązuje problem braku miejsc parkingowych dla samochodów osobowych. Osoby płacące abonament dokonują wyboru parkingu, a następnie przydzielane jest im miejsce.

Schemat bazy danych



Entities

Booking

```java
@Entity
@Data
@Table(name = "bookings")
public class Booking {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "booking_id")
    private int bookingId;
```

```
    @Column(name = "parking_Id")
    private int parkingId;



    @Column(name="car_registration")
    private String subscriberCarRegistration;

    @Column(name = "booking_date")
    private String bookingDate;


    public int getParkingId() {
        return parkingId;
    }

    public String getBookingDate() {
        return bookingDate;
    }

    public int getBookingId() {
        return bookingId;
    }

    public String getSubscriberCarRegistration() {
        return subscriberCarRegistration;
    }

}
```

> Parking

```
@Entity
@Data
@Table(name="parking")
public class Parking {

    @Id
    @Column(name="parking_Id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int parkingId;

    @Column(name="coordinates")
    private String coordinates;
    //address like {ul.} Mazowiecka number, city
    @Column(name="address")
    @Pattern(regexp = "\\p{L}+ \\d+, \\p{L}+", message = "enter address as
street number, city")
    private String address;

    @Column(name = "max_slots")
```

```
    @Min(value = 0, message = "maximum number of slots must be non-negative
number")
    private int maxSlots;

}
```

Subscriber

```
@Entity
@Data
@Table(name = "subscribers")

public class Subscriber {

   @Id
   @Column(name = "car_registration", nullable = false)
   @Pattern(regexp = "\\p{L}+\\d+")
   private String carRegistration;


   @Column(name = "first_name")
   @NotBlank(message = "your name can't be empty")
   private String firstName;

   @NotBlank(message = "your surname can't be empty")
   @Column(name = "last_name")
   private String lastName;


   @Column(name = "end_date")
   private Date endDate;

   @Column(name = "main_parking")
   private Integer mainParking;

   @Column(name = "all_parkings")
   private boolean allParkings;

   public @NotBlank(message = "your name can't be empty") String
getFirstName() {
      return firstName;
   }

   public @NotBlank(message = "your surname can't be empty") String
getLastName() {
      return lastName;
   }


   public Integer getMainParking() {
      return mainParking;
```

```
    }

    public boolean isAllParkings() {
       return allParkings;
    }


}
```

Dto - Data Transfer Objects

> Klasy potrzebne do reprezentacji endpointow, nie znajdujące się w bazie danych

> ParkingDetails - dodatkowe informacje o danym parkingu

```
@Data
public class ParkingDetails {

    private int parkingId;
    private String coordinates;
    private String address;
    private int maxSlots;
    private int freeSlots;
    private String percentageParkingLoad;

    public ParkingDetails(Parking parking, int freeSlots) {
        this.parkingId = parking.getParkingId();
        this.coordinates = parking.getCoordinates();
        this.address = parking.getAddress();
        this.maxSlots = parking.getMaxSlots();
        this.freeSlots = freeSlots;
        this.percentageParkingLoad = calculatePercentageParkingLoad();
    }

   private String calculatePercentageParkingLoad() {
      double parkingLoad = ((double)(freeSlots) / maxSlots) * 100;
      double parkingLoad = ((double)(maxSlots - freeSlots) / maxSlots) *
100;
       return String.format("%.0f%%", parkingLoad);
   }

}
```

> ParkingSubscribers - podstawowe informację o parkingu oraz lista subskrybentów z aktywną licencją na ten parking

```
@Data
public class ParkingSubscribers {
   private int parkingId;
   private String coordinates;
   private String address;
   private List<SubscriberDto> subscribers;

}
```

ParkingSummary - szczegółowe informacje o parkingach

```
public class ParkingSummary {
   private Parking parking;
   private int totalCaluclatedDays;
   private int totalParkedSum;
   private float AvarageDailyParkedSum;
   private int maxBookedSlots;
   private float AvarageDailyParkedPercentage;
   private float maxDailyBookedPercentage;

   public ParkingSummary(Parking parking) {
      this.parking = parking;
   }

   public Parking getParking() {
      return parking;
   }

   public int getTotalCaluclatedDays() {
      return totalCaluclatedDays;
   }

   public void setTotalCaluclatedDays(int totalCaluclatedDays) {
      this.totalCaluclatedDays = totalCaluclatedDays;
   }

   public int getTotalParkedSum() {
      return totalParkedSum;
   }

   public void setTotalParkedSum(int totalParkedSum) {
      this.totalParkedSum = totalParkedSum;
   }

   public float getAvarageDailyParkedSum() {
      return AvarageDailyParkedSum;
   }

   public void setAvarageDailyParkedSum(float avarageDailyParkedSum) {
```

```
        AvarageDailyParkedSum = avarageDailyParkedSum;
    }

    public int getMaxBookedSlots() {
        return maxBookedSlots;
    }

    public void setMaxBookedSlots(int maxBookedSlots) {
        this.maxBookedSlots = maxBookedSlots;
    }

    public float getAvarageDailyParkedPercentage() {
        return AvarageDailyParkedPercentage;
    }

    public void setAvarageDailyParkedPercentage(float
avarageDailyParkedPercentage) {
        AvarageDailyParkedPercentage = avarageDailyParkedPercentage;
    }

    public float getMaxDailyBookedPercentage() {
        return maxDailyBookedPercentage;
    }

    public void setMaxDailyBookedPercentage(float maxDailyBookedPercentage)
{
        this.maxDailyBookedPercentage = maxDailyBookedPercentage;
    }
}
```

> SubscriberDto - podstawowe informacje o subskrybencie

```
@AllArgsConstructor
@Data
public class SubscriberDto {
    private String carRegistration;
    private String firstName;
    private String lastName;
}
```

Dao - Data Access Objects

> Interfejsy zawierające implementacje podstawowych metod sqlowych typu select do znalezienia danego przez obiektu w repozytorium

> BookingRepository

```
public interface BookingRepository extends JpaRepository<Booking, Integer>
{
    Booking findBookingByBookingId(int id);
    List<Booking> findBookingBySubscriberCarRegistration(String
CarRegistration);

}
```

> ParkingRepository

```
public interface ParkingRepository extends JpaRepository<Parking, Integer>
{
   Parking findByParkingId(int id);

}
```

> SubscriberRepository

```
public interface SubscriberRepository extends JpaRepository<Subscriber,
String> {

}
```

Services

> BookingService - zawiera implementacje funkcji związanych z rezerwacji

```
@Service
public class BookingService {
   private final BookingRepository bookingRepository;
   private final ParkingService parkingService;

   private final SubscriberService subscriberService;
   private final ParkingRepository parkingRepository;
   private final SubscriberRepository subscriberRepository;


   @Autowired
   public BookingService(BookingRepository bookingRepository,
ParkingService parkingService, SubscriberService subscriberService,
ParkingRepository parkingRepository, SubscriberRepository
subscriberRepository) {
```

```
        this.bookingRepository = bookingRepository;
        this.parkingService = parkingService;
        this.subscriberService = subscriberService;
        this.parkingRepository = parkingRepository;
        this.subscriberRepository = subscriberRepository;
    }

    public ResponseEntity<Booking> findBookingById(int bookingId){
        try {
            Booking foundBooking =
bookingRepository.findBookingByBookingId(bookingId);
            if (foundBooking != null) {
                return ResponseEntity.ok(foundBooking);
            } else {
                return ResponseEntity.notFound().build();
            }
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }


    public void saveBooking(Booking bookingToSave) throws
ParkingNotFoundException, SubscriberNotFoundException{
        String carRegistration =
bookingToSave.getSubscriberCarRegistration();
        int parkingId = bookingToSave.getParkingId();
        String bookingDate = bookingToSave.getBookingDate();
        if(!subscriberService.subscriberExist(carRegistration)) {
            throw new SubscriberNotFoundException();
        }

        if (!parkingService.parkingExist(parkingId)) {
            throw new ParkingNotFoundException();
        }


        Optional<Subscriber> optSubscriber =
subscriberRepository.findById(carRegistration);
        Subscriber subscriber =  optSubscriber.get();
        Parking parking = parkingRepository.findByParkingId(parkingId);



        if (!subscriberService.hasLicense(carRegistration, parkingId)) {
            throw new IllegalStateException("Subskrybent o numerze
rejestracyjnym " + carRegistration +
                    " nie posiada licencji na parking: " + parkingId);
        }
        if(!subscriberService.isSubscriptionActive(subscriber, bookingDate)){
            throw new IllegalStateException("Subskrybent nie posiada aktywnej
subskrypcji na ten dzień");
        }
```

```
      if
(!parkingService.listAvailableParkingsList(bookingDate).contains(parking))
{
         throw new IllegalStateException("W dniu " + bookingDate +
                 " nie mamy już miejsca na parking " + parkingId);
      }

      bookingRepository.save(bookingToSave);
   }



}
```

ParkingService - zawiera implementacje funkcji związanych z parkingami

```
@Service
public class ParkingService {

   private ParkingRepository parkingRepository;
   private BookingRepository bookingRepository;
   private final SubscriberRepository subscriberRepository;
   private final SubscriberService subscriberService;


   @Autowired
   public ParkingService(ParkingRepository parkingRepository,
BookingRepository bookingRepository,
                         SubscriberRepository subscriberRepository,
SubscriberService subscriberService) {
      this.parkingRepository = parkingRepository;
      this.bookingRepository = bookingRepository;
      this.subscriberRepository = subscriberRepository;
      this.subscriberService = subscriberService;
   }


   public boolean parkingExist(int parkingId){
      return parkingRepository.existsById(parkingId);
   }


   private Map<Parking, Integer> calculateBookedSlots(String date){
      List<Parking> parkings = parkingRepository.findAll();
      List<Booking> bookings = bookingRepository.findAll();
      //przechowuje liczbe <Parking, liczba zajetych miejsc>
      Map<Parking, Integer> parkingMap = new HashMap<>();
      for (Parking parking: parkings){
```

```
            parkingMap.put(parking, 0);
        }
        for(Booking booking: bookings){
            if(booking.getBookingDate().equals(date)){
                int parkingId = booking.getParkingId();
                Parking bookedParking =
parkingRepository.findByParkingId(parkingId);
                parkingMap.put(bookedParking, parkingMap.get(bookedParking) +
1);
            }
        }
        return parkingMap;

    }

    public List<ParkingDetails> listAvailableParkings(String date){
        Map<Parking, Integer> parkingMap = calculateBookedSlots(date);
        List<ParkingDetails> availableParkingsNow = new ArrayList<>();

        for(Map.Entry<Parking, Integer> entry: parkingMap.entrySet()){
            Parking potentialParking = entry.getKey();
            int bookedSlots = entry.getValue();
            int freeSlots = potentialParking.getMaxSlots() - bookedSlots;
            if(freeSlots > 0){
                ParkingDetails parkingDetails = new
ParkingDetails(potentialParking, freeSlots);
                availableParkingsNow.add(parkingDetails);
            }}

        return availableParkingsNow;
    }

    public List<Parking> listAvailableParkingsList(String date){
        Map<Parking, Integer> parkingMap = calculateBookedSlots(date);
        List<Parking> availableParkingsNow = new ArrayList<>();

        for(Map.Entry<Parking, Integer> entry: parkingMap.entrySet()){
            Parking potentialParking = entry.getKey();
            int bookedSlots = entry.getValue();
            int freeSlots = potentialParking.getMaxSlots() - bookedSlots;
            if(freeSlots > 0){
                availableParkingsNow.add(potentialParking);
            }}

        return availableParkingsNow;
    }

    public int numberOfFreeSlots(int parkingId, String date){
        Map<Parking, Integer> parkingMap = calculateBookedSlots(date);
        Parking parking = parkingRepository.findByParkingId(parkingId);
        int bookedSlots = parkingMap.get(parking);

        return parking.getMaxSlots() - bookedSlots;
    }
```

```
    public List<ParkingSubscribers> listAllParkingSubscribers(String date) {

        List<Subscriber> subscribers =
subscriberService.activeLicenseSubscribers(date);

        List<Parking> parkings = parkingRepository.findAll();
        List<ParkingSubscribers> parkingSubscribersList = new ArrayList<>();


        for (Parking parking : parkings) {
            int parkingId = parking.getParkingId();

            ParkingSubscribers parkingSubscribers = new ParkingSubscribers();
            parkingSubscribers.setParkingId(parkingId);
            parkingSubscribers.setCoordinates(parking.getCoordinates());
            parkingSubscribers.setAddress(parking.getAddress());


            List<SubscriberDto> subscriberDtoList =
getSubscriberDtoList(subscribers, parkingId);
            parkingSubscribers.setSubscribers(subscriberDtoList);

            parkingSubscribersList.add(parkingSubscribers);
        }
        return parkingSubscribersList;
    }

    private static List<SubscriberDto> getSubscriberDtoList(List<Subscriber>
subscribers, int parkingId) {
        List<SubscriberDto> subscriberDtoList = new ArrayList<>();
        for (Subscriber subscriber : subscribers) {
            boolean hasAllParkingSubscription = subscriber.isAllParkings();
            int subscriberMainParking = subscriber.getMainParking();

            if(hasAllParkingSubscription || subscriberMainParking ==
parkingId){
                SubscriberDto subscriberDto = new
SubscriberDto(subscriber.getCarRegistration(),
                        subscriber.getFirstName(), subscriber.getLastName());
                subscriberDtoList.add(subscriberDto);
            }
        }
        return subscriberDtoList;
    }

}
```

SubscriberService - zawiera implementacje funkcji związanych z subsrybentami

```
@Service
public class SubscriberService {


    private final SubscriberRepository subscriberRepository;
    private final BookingRepository bookingRepository;

    @Autowired
    public SubscriberService(SubscriberRepository subscriberRepository,
BookingRepository bookingRepository) {
        this.subscriberRepository = subscriberRepository;
        this.bookingRepository = bookingRepository;
    }

    public List<Subscriber> activeLicenseSubscribers(String date){
        List<Subscriber> allSubscribers = subscriberRepository.findAll();
        List<Subscriber> activeLicenseSubscribers = new ArrayList<>();

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");
        LocalDate currentDate = LocalDate.parse(date, formatter);
        for(Subscriber subscriber: allSubscribers){
            String endDate =  subscriber.getEndDate().toString();
            endDate = endDate.substring(0,10);
            LocalDate licenseEnd = LocalDate.parse(endDate, formatter);
            if(licenseEnd.isAfter(currentDate) ||
licenseEnd.isEqual(currentDate)){
                activeLicenseSubscribers.add(subscriber);
            }
        }
        return activeLicenseSubscribers;

    }


    public boolean subscriberExist(String subscriberCarRegistration){
        return subscriberRepository.existsById(subscriberCarRegistration);
    }

    public boolean isSubscriptionActive(Subscriber subscriber, String date)
{
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");
        LocalDate currentDate = LocalDate.parse(date, formatter);
        String endDate =  subscriber.getEndDate().toString();
        endDate = endDate.substring(0,10);
        LocalDate licenseEnd = LocalDate.parse(endDate, formatter);
        if(licenseEnd.isAfter(currentDate) ||
licenseEnd.isEqual(currentDate)){
           return true;
        }
        return false;
```

```
    }


    public boolean hasLicense(String subscriberCarRegistration, int
parkingId){
        Optional<Subscriber> OptionalSubscriber =
subscriberRepository.findById(subscriberCarRegistration);


        if(!OptionalSubscriber.isPresent()){
//            throw new SubscriberNotFoundException("Subskrybent o podanym
numerze rejestracyjnm nie istnieje");
            return false;

        }

        Subscriber subscriber = OptionalSubscriber.get();
        if(subscriber.isAllParkings()){
            return true;
        }


        if(subscriber.getMainParking() == null){
            return false;
        }

        return subscriber.getMainParking() == parkingId;

    }


    public String deleteSubscriber(String subscriberCarRegistration){
        if(!subscriberExist(subscriberCarRegistration)){
           throw new SubscriberNotFoundException();

        }
        Optional<Subscriber> subscriberToDelete =
subscriberRepository.findById(subscriberCarRegistration);
        Subscriber subscriber = subscriberToDelete.get();
        List<Booking> bookingList =
bookingRepository.findBookingBySubscriberCarRegistration(subscriberCarRegis
tration);

        bookingRepository.deleteAll(bookingList);
        subscriberRepository.delete(subscriber);

        return "Usunięto subskrybenta o rejestracji: " +
subscriberCarRegistration + " oraz " + bookingList.toArray().length + "
jego rezerwacji";

    }


    public Subscriber getSubscriber(String subscriberCarRegistration){
```

```
        if(!subscriberExist(subscriberCarRegistration)){
            throw new SubscriberNotFoundException();
        }
        Optional<Subscriber> subscriber =
subscriberRepository.findById(subscriberCarRegistration);
        return subscriber.orElseThrow(SubscriberNotFoundException::new);
    }


    public Subscriber putSubscriber(String subscriberCarRegistration,
Subscriber newSubscriberData){
        if(!subscriberExist(subscriberCarRegistration)){
            throw new SubscriberNotFoundException();
        }

        return subscriberRepository.findById(subscriberCarRegistration)
                .map(subscriber -> {

subscriber.setFirstName(newSubscriberData.getFirstName());

subscriber.setLastName(newSubscriberData.getLastName());

subscriber.setMainParking(newSubscriberData.getMainParking());
                    subscriber.setEndDate(newSubscriberData.getEndDate());

subscriber.setAllParkings(newSubscriberData.isAllParkings());
                    return subscriberRepository.save(subscriber);
                }).orElseThrow(SubscriberNotFoundException::new);
    }


    public Subscriber patchSubscriber(String carRegistration, Subscriber
newSubscriberData) {
        return subscriberRepository.findById(carRegistration)
                .map(subscriber -> {
                    if (newSubscriberData.getFirstName() != null) {

subscriber.setFirstName(newSubscriberData.getFirstName());
                    }
                    if (newSubscriberData.getLastName() != null) {

subscriber.setLastName(newSubscriberData.getLastName());
                    }
                    if (newSubscriberData.getEndDate() != null) {

subscriber.setEndDate(newSubscriberData.getEndDate());
                    }
                    if (newSubscriberData.getMainParking() != null) {

subscriber.setMainParking(newSubscriberData.getMainParking());
                    }

subscriber.setAllParkings(newSubscriberData.isAllParkings());
                    return subscriberRepository.save(subscriber);
```

```
                })
                .orElseThrow(SubscriberNotFoundException::new);
    }


}
```

ParkingSummaryService - zawiera implementacje funkcji związanych z podsumowaniami

```
@Service
public class ParkingSummaryService {
    private final ParkingRepository parkingRepository;
    private final ParkingService parkingService;
    @Autowired
    public ParkingSummaryService(ParkingRepository parkingRepository,
ParkingService parkingService) {
        this.parkingRepository = parkingRepository;
        this.parkingService = parkingService;
    }

    private int max(int a, int b){
        if(a>b){
            return a;
        }
        return b;
    }

    private float[] calculateStats(String date1, String date2, Parking
parking){
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");
        LocalDate startDate = LocalDate.parse(date1, formatter);
        LocalDate endDate = LocalDate.parse(date2, formatter);
        int booked_counter = 0;
        int days = 0;
        int max_count = 0;
        for (LocalDate date = startDate; !date.isAfter(endDate); date =
date.plusDays(1)) {
           int booked = parking.getMaxSlots() -
parkingService.numberOfFreeSlots(
                   parking.getParkingId(), date.format(formatter));
//
System.out.println(parkingService.numberOfFreeSlots(parking.getParkingId(),
date.format(formatter)));
            booked_counter+= booked;
            max_count = max(max_count, booked);
            days+=1;

        }
        float daily_percentage = 0;
```

```
        float max_daily_percentage = 0;
        if(days!=0 && parking.getMaxSlots()!=0){
             daily_percentage = (float) booked_counter/(days *
parking.getMaxSlots());
             max_daily_percentage = (float) max_count/parking.getMaxSlots()
* 100;
        }

        float[] cntdays = new float[5];
        cntdays[0] = booked_counter;
        cntdays[1] = days;
        cntdays[2] = max_count;
        cntdays[3] = daily_percentage;
        cntdays[4] = max_daily_percentage;

        return cntdays;

    }


    public List<ParkingSummary> parkingSummaries(String date1, String
date2) throws Exception{
        DateValidation dateValidation = DateValidation.getInstance();
        if(!dateValidation.validateDate(date1) ||
!dateValidation.validateDate(date2)){
            throw new WrongDateFormatException();
        }

        List<Parking> listOfParkings = parkingRepository.findAll();
        List<ParkingSummary> parkingSummaries = new LinkedList<>();
        for(Parking parking: listOfParkings){
            ParkingSummary parkingToAdd = new ParkingSummary(parking);
             float[] cntdays = calculateStats(date1,date2,parking);
             int bookedCounter = (int) cntdays[0];
             int days = (int) cntdays[1];
             int maxBooked = (int) cntdays[2];
             float dailyPercentage = cntdays[3];
             float maxDailyPercentage = cntdays[4];
             float avarageDailyParkedSum = (float) bookedCounter/days;
             parkingToAdd.setAvarageDailyParkedSum(avarageDailyParkedSum);
             parkingToAdd.setTotalParkedSum(bookedCounter);
             parkingToAdd.setTotalCaluclatedDays(days);
             parkingToAdd.setMaxDailyBookedPercentage(maxDailyPercentage);
             parkingToAdd.setMaxBookedSlots(maxBooked);
             parkingToAdd.setAvarageDailyParkedPercentage(dailyPercentage);

             parkingSummaries.add(parkingToAdd);

        }

        return parkingSummaries;

    }
}
```

Controllers

> DataBookingController - zawiera endpointy dotyczące rezerwacji

```
@RestController
@RequestMapping("/bookings")
public class DataBookingController {
    private final BookingRepository bookingRepository;
    private final ParkingService parkingService;
    private final SubscriberService subscriberService;
    private final ParkingRepository parkingRepository;

    private final BookingService bookingService;

    @Autowired
    public DataBookingController(BookingRepository bookingRepository,
ParkingService parkingService,
                                 SubscriberService subscriberService,
ParkingRepository parkingRepository, BookingService bookingService) {
        this.bookingRepository = bookingRepository;
        this.parkingService = parkingService;
        this.subscriberService = subscriberService;
        this.parkingRepository = parkingRepository;
        this.bookingService = bookingService;
    }


    @PostMapping()
    public ResponseEntity<String> saveBooking(@Validated @RequestBody
Booking bookingToSave) {
        try {
            bookingService.saveBooking(bookingToSave);
            return ResponseEntity.ok("Utworzono rezerwację na dzień: " +
bookingToSave.getBookingDate());
        } catch (SubscriberNotFoundException | ParkingNotFoundException e)
{
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        } catch (IllegalStateException e) {
            return
ResponseEntity.status(HttpStatus.CONFLICT).body(e.getMessage());
        }catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił błąd
podczas tworzenia rezerwacji");
        }

    }
```

```
    @GetMapping()
    public ResponseEntity<List<Booking>> bookingList(){
        List<Booking> bookingList = bookingRepository.findAll();
        return ResponseEntity.ok(bookingList);
    }




}
```

DataRestParkingController - zawiera endpointy dotyczące parkingów

```
@RestController
public class DataRestParkingController {
    private final ParkingService parkingService;
    private final ParkingRepository parkingRepository;
    private final ParkingSummaryService parkingSummaryService;


    @Autowired
    public DataRestParkingController(ParkingRepository parkingRepository,
BookingRepository bookingRepository, ParkingService parkingService,
ParkingSummaryService parkingSummaryService) {
        this.parkingService = parkingService;
        this.parkingRepository = parkingRepository;
        this.parkingSummaryService = parkingSummaryService;
    }

    @GetMapping("/getAllAvailableParkingsNow")
    public ResponseEntity<List<ParkingDetails>> getAllAvailableParking() {
        LocalDate today = LocalDate.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");
        String formattedDate = today.format(formatter);

        return new ResponseEntity<>
(parkingService.listAvailableParkings(formattedDate), HttpStatus.OK);
    }


    @GetMapping("/getAllAvailableParkings/{date}")
    public ResponseEntity<?> getAllAvailableParking(@PathVariable String
date) {

        DateValidation dateValidation = DateValidation.getInstance();
        if (!dateValidation.validateDate(date)) {
            return new ResponseEntity<>("Niepoprawny format daty",
HttpStatus.BAD_REQUEST);
```

```
        }

        List<ParkingDetails> availableParkings =
parkingService.listAvailableParkings(date);
        return new ResponseEntity<>(availableParkings, HttpStatus.OK);
    }


    @GetMapping("/getReport/{date1}/{date2}")
    public ResponseEntity<Object> getParkingSummary(@PathVariable String
date1, @PathVariable String date2) {
        try {
            List<ParkingSummary> parkingSummaries =
parkingSummaryService.parkingSummaries(date1, date2);
            return ResponseEntity.ok(parkingSummaries);
        } catch (WrongDateFormatException e) {
            return ResponseEntity.badRequest().body("Niepoprawny format
daty. Wymagany format: yyyy-MM-dd");
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił
nieoczekiwany błąd podczas" +
                    " pobierania podsumowania parkingu: " + e.getMessage()
);
        }
    }


    @GetMapping("/listAllParkingSubscribers/{date}")
    public ResponseEntity<?> listAllParkingSubscribers(@PathVariable String
date){

        DateValidation dateValidation = DateValidation.getInstance();
        if (!dateValidation.validateDate(date)) {
            return new ResponseEntity<>("Niepoprawny format daty",
HttpStatus.BAD_REQUEST);
        }

        List<ParkingSubscribers> parkingSubscribersList =
parkingService.listAllParkingSubscribers(date);
        return new ResponseEntity<>(parkingSubscribersList, HttpStatus.OK);

    }




}
```

> DataSubscriberController - zawiera endpointy dotyczące subskrybentów

```
@RestController
@RequestMapping("/subscribers")
public class DataSubscriberController {
    private final SubscriberRepository subscriberRepository;
    private final SubscriberService subscriberService;


    public DataSubscriberController(SubscriberRepository
subscriberRepository, SubscriberService subscriberService) {
        this.subscriberRepository = subscriberRepository;
        this.subscriberService = subscriberService;
    }
    @DeleteMapping("/{carRegistration}")
    public ResponseEntity<String> deleteSubscriber(@PathVariable String
carRegistration){
        try{
            String response =
subscriberService.deleteSubscriber(carRegistration);
            return ResponseEntity.ok(response);
        }
        catch (SubscriberNotFoundException e){
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        } catch(Exception e){
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił
nieoczekwiany błąd");
        }
    }

    @GetMapping("/{carRegistration}")
    public ResponseEntity<?> getSubscriber(@PathVariable String
carRegistration){
        try{
            Subscriber subscriber =
subscriberService.getSubscriber(carRegistration);
            return ResponseEntity.ok(subscriber);
        }
        catch (SubscriberNotFoundException e){
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }catch(Exception e){
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił
nieoczekwiany błąd");
        }
    }


    @PutMapping("/{carRegistration}")
    public ResponseEntity<?> putSubscriber(@PathVariable String
```

```
carRegistration, @RequestBody Subscriber updatedSubscriber){
        try{
            Subscriber subscriber =
subscriberService.putSubscriber(carRegistration, updatedSubscriber);
            return ResponseEntity.ok(subscriber);
        }catch (SubscriberNotFoundException e){
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }catch(Exception e){
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił
nieoczekwiany błąd");
        }
    }


    @PatchMapping("/{carRegistration}")
    public ResponseEntity<?> patchSubscriber(@PathVariable String
carRegistration, @RequestBody Subscriber updatedSubscriber){
        try{
            Subscriber subscriber =
subscriberService.patchSubscriber(carRegistration, updatedSubscriber);
            return ResponseEntity.ok(subscriber);
        }catch (SubscriberNotFoundException e){
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }catch(Exception e){
            e.printStackTrace();
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Wystąpił
nieoczekwiany błąd");
        }
    }
}
```

Wyjatki

> LicenseExpiredException - rzuca wyjątek jeśli licencja na parking danego subskrybenta jest nie aktywna

```
public class LicenseExpiredException extends Exception{
    public LicenseExpiredException() {
        super("Licencja podanego subskrybenta przedawniła się");
    }
}
```

> ParkingNotFoundException - rzuca wyjątek jeśli szukany parking nie istnieje

```
public class ParkingNotFoundException extends Exception{
    public ParkingNotFoundException(){super("Nie znaleziono podanego
parkingu");}

}
```

> SubscriberNotFoundException - rzuca wyjątek jeśli dany subskrybent nie istnieje

```
public class SubscriberNotFoundException extends RuntimeException {
    public SubscriberNotFoundException() {
        super("Nie znaleziono subskrybenta o podanej rejestracji");

    }
}
```

> WrongDateFormatException - rzuca wyjątek jeśli data ma zły format

```
public class WrongDateFormatException extends Exception {
    public WrongDateFormatException() {
        super("Podano zły format daty");
    }
}
```

Walidacja danych

> CoordinatesValidation - sprawdza czy podane współrzędne są prawidłowe

```
public class CoordinatesValidation {
    private static final Pattern COORDINATES_PATTERN =
Pattern.compile("\\d{2}(\\.\\d+)?, \\d{2}(\\.\\d+)?");

    public static boolean isValidCoordinates(String coords){
        Matcher matcher = COORDINATES_PATTERN.matcher(coords);
        return matcher.matches();
    }

}
```

> DateValidation - sprawdza czy data ma prawidłowy format

```
```

```
public class DateValidation {
    private static DateValidation instance;


    private DateValidation() {
    }

    public boolean validateDate(String date){
        String regex = "^\\d{4}-\\d{2}-\\d{2}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(date);

        return matcher.matches();
    }

    public static DateValidation getInstance(){
        if(instance == null){
            instance = new DateValidation();
        }
        return instance;
    }

}
```

Aplikacja uruchomieniowa

> ParkingSystemApplication

```
@SpringBootApplication
public class ParkingSystemApplication {

    public static void main(String[] args) {
        SpringApplication.run(ParkingSystemApplication.class, args);
    }

}
```

Dane wejściowe

> Dostępne parkingi oraz subskrybenci Dane są usuwane i wczytywane na nowo z każdym uruchomieniem aplikacji

```
use `parking-system`;
SET FOREIGN_KEY_CHECKS=0;
TRUNCATE TABLE bookings;
TRUNCATE TABLE parking;
```

```
TRUNCATE TABLE subscribers;
SET FOREIGN_KEY_CHECKS=1;


INSERT INTO parking (coordinates, address, max_slots)
VALUES
    ('52.22, 21.012', 'Mazowiecka 1, Warszawa', 40),
    ('50.06, 19.94', 'ul. Grodzka 10, Kraków', 10),
    ('50.06, 18.94', 'ul. Kawiory 10, Kraków', 8),
    ('51.10, 17.03', 'Rynek 2, Wrocław', 15),
    ('53.13, 23.16', 'ul. Lipowa 15, Białystok', 30),
    ('54.35, 18.64', 'Dluga 5, Gdańsk', 20);

INSERT INTO subscribers (car_registration, first_name, last_name, end_date,
main_parking, all_parkings)
VALUES
    ('ABC123', 'Jan', 'Kowalski', '2024-12-31', 1, TRUE),
    ('DEF456', 'Anna', 'Nowak', '2024-11-30', 2, FALSE),
    ('GHI789', 'Piotr', 'Wiśniewski', '2024-10-31', 3, TRUE),
    ('JKL101', 'Maria', 'Wójcik', '2024-09-30', 4, FALSE),
    ('MNO112', 'Krzysztof', 'Kowalczyk', '2024-08-31', 5, TRUE),
    ('PQR131', 'Zofia', 'Kamińska', '2024-07-31', 6, FALSE),
    ('STU415', 'Tomasz', 'Lewandowski', '2024-06-30', 1, TRUE),
    ('VWX161', 'Katarzyna', 'Zielińska', '2024-06-30', 2, FALSE),
    ('YZA718', 'Paweł', 'Szymański', '2024-07-30', 3, TRUE),
    ('BCD192', 'Magdalena', 'Woźniak', '2024-08-31', 4, FALSE),
    ('EFG213', 'Michał', 'Dąbrowski', '2024-08-29', 5, TRUE),
    ('HIJ324', 'Agnieszka', 'Kozłowska', '2024-09-30', 6, FALSE),
    ('KLM435', 'Rafał', 'Jankowski', '2024-12-30', 1, TRUE),
    ('NOP546', 'Joanna', 'Wiśniewska', '2024-11-29', 2, FALSE),
    ('QRS657', 'Maciej', 'Wróblewski', '2024-10-28', 3, TRUE),
    ('TUV768', 'Ewa', 'Kamiński', '2024-09-27', 4, FALSE),
    ('WXY879', 'Adam', 'Zieliński', '2024-08-26', 5, TRUE),
    ('ZAB980', 'Magda', 'Szymczak', '2024-07-25', 6, FALSE),
    ('BCD101', 'Marek', 'Lewandowski', '2024-06-24', 1, TRUE),
    ('DEF212', 'Izabela', 'Kamińska', '2024-09-23', 2, FALSE),
    ('GHI323', 'Karol', 'Kozłowski', '2024-08-22', 3, TRUE),
    ('JKL434', 'Patrycja', 'Jankowska', '2024-09-21', 4, FALSE),
    ('MNO545', 'Dariusz', 'Kowal', '2024-09-20', 5, TRUE),
    ('PQR656', 'Aleksandra', 'Mazur', '2024-09-19', 6, FALSE),
    ('STU767', 'Wojciech', 'Nowicki', '2024-12-18', 1, TRUE),
    ('VWX878', 'Natalia', 'Kwiatkowska', '2024-11-17', 2, FALSE),
    ('YZA989', 'Andrzej', 'Wysocki', '2024-10-16', 3, TRUE),
    ('BCD100', 'Alicja', 'Głowacka', '2024-09-15', 4, FALSE),
    ('EFG211', 'Roman', 'Król', '2024-08-14', 5, TRUE),
    ('HIJ322', 'Agnieszka', 'Adamska', '2024-07-13', 6, FALSE),
    ('KLM433', 'Jerzy', 'Pawlak', '2024-06-12', 1, TRUE),
    ('NOP544', 'Łukasz', 'Kaczmarek', '2024-11-11', 2, FALSE),
    ('QRS655', 'Renata', 'Borkowska', '2024-11-10', 3, TRUE),
    ('TUV766', 'Mariusz', 'Piotrowski', '2024-06-09', 4, FALSE),
    ('WXY877', 'Sylwia', 'Sikorska', '2024-08-08', 5, TRUE),
    ('ZAB988', 'Wiesław', 'Duda', '2024-08-07', 6, FALSE),
    ('BCD109', 'Martyna', 'Lis', '2024-12-06', 1, TRUE),
    ('DEF210', 'Patryk', 'Kurowski', '2024-11-05', 2, FALSE),
```

```
    ('GHI321', 'Urszula', 'Kowalewska', '2024-10-04', 3, TRUE),
    ('JKL432', 'Marcin', 'Wilk', '2024-09-03', 4, FALSE),
    ('MNO543', 'Iwona', 'Chmielewska', '2024-08-02', 5, TRUE),
    ('PQR654', 'Sebastian', 'Urbaniak', '2024-07-01', 6, FALSE);
```

BookingGeneratorService - generuje rezerwacje przechowywane w bazie danych

```
@Service
public class BookingGeneratorService {

    @Autowired
    private BookingRepository bookingRepository;

    private static final String[] SUBSCRIBERS = {
            "ABC123", "DEF456", "GHI789", "JKL101", "MNO112", "PQR131",
            "STU415", "VWX161", "YZA718", "BCD192", "EFG213", "HIJ324",
            "KLM435", "NOP546", "QRS657", "TUV768", "WXY879", "ZAB980",
            "BCD101", "DEF212", "GHI323", "JKL434", "MNO545", "PQR656",
            "STU767", "VWX878", "YZA989", "BCD100", "EFG211", "HIJ322",
            "KLM433", "NOP544", "QRS655", "TUV766", "WXY877", "ZAB988",
            "BCD109", "DEF210", "GHI321", "JKL432", "MNO543", "PQR654"
    };
    private static final int[] PARKING_IDS = {1, 2, 3, 4, 5, 6};

    @PostConstruct
    public void generateBookings() {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Random random = new Random();

        for (int i = 0; i < 300; i++) {
            String carRegistration =
SUBSCRIBERS[random.nextInt(SUBSCRIBERS.length)];
            int parkingId =
PARKING_IDS[random.nextInt(PARKING_IDS.length)];
            Date bookingDate = generateRandomDate("2024-06-01", "2024-12-
31");

            Booking booking = new Booking();
            booking.setSubscriberCarRegistration(carRegistration);
            booking.setParkingId(parkingId);
            booking.setBookingDate(String.valueOf(bookingDate));

            bookingRepository.save(booking);
        }

        System.out.println("Bookings generated successfully.");
```

```
    }


    private Date generateRandomDate(String startDate, String endDate) {
        try {
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd");
            Date start = dateFormat.parse(startDate);
            Date end = dateFormat.parse(endDate);

            long randomDate = start.getTime() + (long) (Math.random() *
(end.getTime() - start.getTime()));
            return new java.sql.Date(randomDate); // Use java.sql.Date
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

Dostępne endpointy

Operacje get

> Uwaga: Daty podaje w formacie RRRR-MM-DD

1. http://localhost:8080/subscribers

    > wyświela wszystkich subksrybentów

```
_embedded:
  subscribers:
    0:
      firstName:      "Jan"
      lastName:       "Kowalski"
      endDate:        "2024-12-30T23:00:00.000+00:00"
      mainParking:    null
      allParkings:    false
      _links:
        self:
          href:       "http://localhost:8080/subscribers/ABC123"
        subscriber:
          href:       "http://localhost:8080/subscribers/ABC123"
    1:
      firstName:      "Piotr"
      lastName:       "Wiśniewski"
      endDate:        "2024-10-14T22:00:00.000+00:00"
      mainParking:    null
      allParkings:    false
      _links:
        self:
          href:       "http://localhost:8080/subscribers/DEF456"
        subscriber:
          href:       "http://localhost:8080/subscribers/DEF456"
    2:
      firstName:      "Anna"
      lastName:       "Nowak"
      endDate:        "2025-06-29T22:00:00.000+00:00"
      mainParking:    null
      allParkings:    true
      _links:
        self:
          href:       "http://localhost:8080/subscribers/XYZ789"
        subscriber:
          href:       "http://localhost:8080/subscribers/XYZ789"
```


2. http://localhost:8080/parkings

> wyświetla wszystkie dostępne parkingi

```
_embedded:
  parkings:
    0:
      coordinates:   "50.1234, 20.5678"
      address:       "ul. Parkingowa 1, Kraków"
      maxSlots:      100
    _links:
      self:
        href:      "http://localhost:8080/parkings/1"
      parking:
        href:      "http://localhost:8080/parkings/1"
    1:
      coordinates:   "51.9876, 17.4321"
      address:       "ul. Parkietowa 2, Warszawa"
      maxSlots:      80
    _links:
      self:
        href:      "http://localhost:8080/parkings/2"
      parking:
        href:      "http://localhost:8080/parkings/2"
    2:
      coordinates:   "52.3456, 16.7890"
      address:       "ul. Parkowa 3, Poznań"
      maxSlots:      120
    _links:
      self:
        href:      "http://localhost:8080/parkings/3"
      parking:
        href:      "http://localhost:8080/parkings/3"
    3:
      coordinates:   "34, 56"
      address:       "Mazowiecka 23, Krakow"
      maxSlots:      0
    _links:
      self:
```

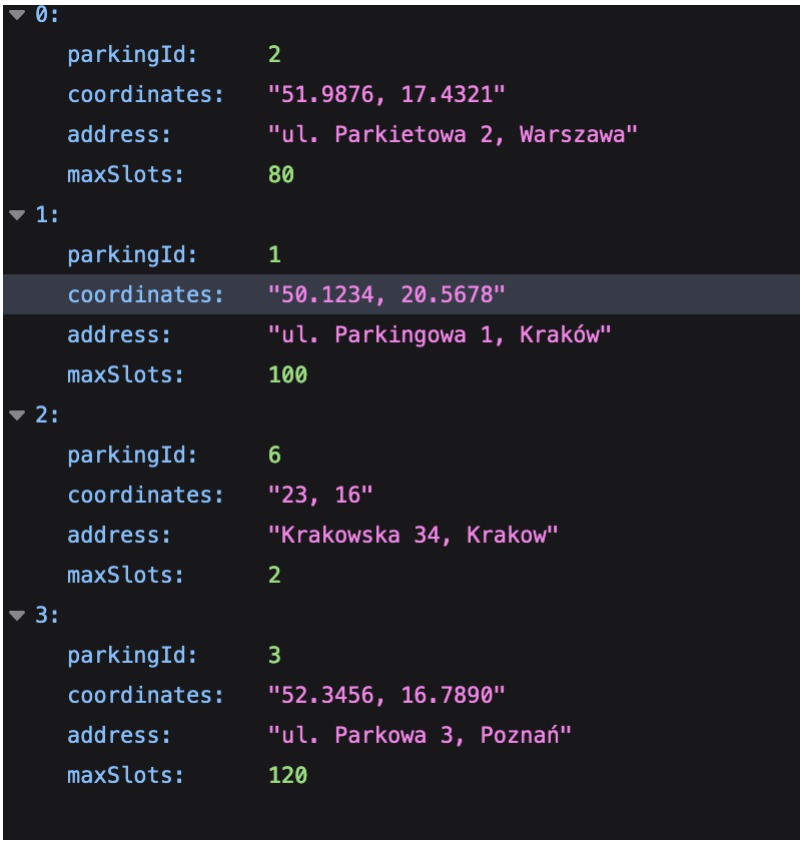

3. http://localhost:8080/bookings

> wyświetla wszystkie dostępne rezerwacje

```
[
    {
        "bookingId": 1,
        "parkingId": 4,
        "subscriberCarRegistration": "BCD100",
        "bookingDate": "2024-11-14"
    },
    {
        "bookingId": 2,
        "parkingId": 6,
        "subscriberCarRegistration": "WXY879",
        "bookingDate": "2024-10-27"
    },
    {
        "bookingId": 3,
        "parkingId": 1,
        "subscriberCarRegistration": "MNO545",
        "bookingDate": "2024-10-14"
    },
    {
        "bookingId": 4,
        "parkingId": 3,
        "subscriberCarRegistration": "BCD192",
        "bookingDate": "2024-06-22"
    },
    {
        "bookingId": 5,
        "parkingId": 5,
        "subscriberCarRegistration": "GHI323",
        "bookingDate": "2024-12-21"
    },
    {
        "bookingId": 6,
        "parkingId": 5,
        "subscriberCarRegistration": "VWX878",
        "bookingDate": "2024-09-26"
    },
    {
        "bookingId": 7,
        "parkingId": 3,
        "subscriberCarRegistration": "STU767",
        "bookingDate": "2024-07-03"
```


4. http://localhost:8080/getAllAvailableParkings/RRRR-MM-DD

> wyświetla parkingi na, które jeszcze jest miejsce na dany dzień



5. http://localhost:8080/getReport/startDate/endDate
   np. http://localhost:8080/getReport/2024-01-01/2024-12-31

> Tworzy raport dla każdego parkingu pomiędzy datą początkową a końcową

```
[
    {
        "parking": {
            "parkingId": 1,
            "coordinates": "52.22, 21.012",
            "address": "Mazowiecka 1, Warszawa",
            "maxSlots": 40
        },
        "totalCaluclatedDays": 366,
        "totalParkedSum": 52,
        "maxBookedSlots": 2,
        "maxDailyBookedPercentage": 5.0,
        "avarageDailyParkedSum": 0.1420765,
        "avarageDailyParkedPercentage": 0.0035519125
    },
    {
        "parking": {
            "parkingId": 2,
            "coordinates": "50.06, 19.94",
            "address": "ul. Grodzka 10, Kraków",
            "maxSlots": 10
        },
        "totalCaluclatedDays": 366,
        "totalParkedSum": 48,
        "maxBookedSlots": 2,
        "maxDailyBookedPercentage": 20.0,
        "avarageDailyParkedSum": 0.13114753,
        "avarageDailyParkedPercentage": 0.013114754
    },
    {
        "parking": {
            "parkingId": 3,
            "coordinates": "50.06, 18.94",
            "address": "ul. Kawiory 10, Kraków",
            "maxSlots": 8
        },
        "totalCaluclatedDays": 366,
        "totalParkedSum": 43,
        "maxBookedSlots": 2,
        "maxDailyBookedPercentage": 25.0,
        "avarageDailyParkedSum": 0.117486335,
        "avarageDailyParkedPercentage": 0.014685792
    },
    {
        "parking": {
            "parkingId": 4,
            "coordinates": "51.10, 17.03",
            "address": "Rynek 2, Wrocław",
            "maxSlots": 15
        },
```


6. http://localhost:8080/subscribers/carRegistration
   np. http://localhost:8080/subscribers/ABC123

> wyświetla szczegółowe informację o subskrybencie z podanym numerem rejestracyjnym samochodu

```
{
    "carRegistration": "ABC123",
    "firstName": "Jan",
    "lastName": "Kowalski",
    "endDate": "2024-12-30T23:00:00.000+00:00",
    "mainParking": 1,
    "allParkings": true
}
```


7. http://localhost:8080/getAllAvailableParkingsNow

> wyświetla wszystkie parkingi na których aktualnie są miejsca

```
[
    {
        "parkingId": 2,
        "coordinates": "50.06, 19.94",
        "address": "ul. Grodzka 10, Kraków",
        "maxSlots": 10,
        "freeSlots": 10,
        "percentageParkingLoad": "0%"
    },
    {
        "parkingId": 5,
        "coordinates": "53.13, 23.16",
        "address": "ul. Lipowa 15, Białystok",
        "maxSlots": 30,
        "freeSlots": 30,
        "percentageParkingLoad": "0%"
    },
    {
        "parkingId": 6,
        "coordinates": "54.35, 18.64",
        "address": "Dluga 5, Gdańsk",
        "maxSlots": 20,
        "freeSlots": 20,
        "percentageParkingLoad": "0%"
    },
    {
        "parkingId": 1,
        "coordinates": "52.22, 21.012",
        "address": "Mazowiecka 1, Warszawa",
        "maxSlots": 40,
        "freeSlots": 40,
        "percentageParkingLoad": "0%"
    },
    {
        "parkingId": 4,
        "coordinates": "51.10, 17.03",
        "address": "Rynek 2, Wrocław",
        "maxSlots": 15,
        "freeSlots": 15,
        "percentageParkingLoad": "0%"
    },
    {
        "parkingId": 3,
        "coordinates": "50.06, 18.94",
        "address": "ul. Kawiory 10, Kraków",
        "maxSlots": 8,
        "freeSlots": 8,
        "percentageParkingLoad": "0%"
    }
```


8. http://localhost:8080/listAllParkingSubscribers/{date}
   np. http://localhost:8080/listAllParkingSubscribers/2024-06-04

> wyświetla wszystkich subskrybentów którzy mają aktywną subskrypcje na danym parkingu danego dnia

```
[
    {
        "parkingId": 1,
        "coordinates": "52.22, 21.012",
        "address": "Mazowiecka 1, Warszawa",
        "subscribers": [
            {
                "carRegistration": "ABC123",
                "firstName": "Jan",
                "lastName": "Kowalski"
            },
            {
                "carRegistration": "BCD101",
                "firstName": "Marek",
                "lastName": "Lewandowski"
            },
            {
                "carRegistration": "BCD109",
                "firstName": "Martyna",
                "lastName": "Lis"
            },
            {
                "carRegistration": "EFG211",
                "firstName": "Roman",
                "lastName": "Król"
            },
            {
                "carRegistration": "EFG213",
                "firstName": "Michał",
                "lastName": "Dąbrowski"
            },
            {
                "carRegistration": "GHI321",
                "firstName": "Urszula",
                "lastName": "Kowalewska"
            },
            {
                "carRegistration": "GHI323",
                "firstName": "Karol",
                "lastName": "Kozłowski"
            },
            {
                "carRegistration": "GHI789",
                "firstName": "Piotr",
                "lastName": "Wiśniewski"
            },
            {
                "carRegistration": "KLM433",
                "firstName": "Jerzy",
```


9. http://localhost:8080/bookings/number
   np. http://localhost:8080/bookings/1

> wyświela szczegółowe informacje o konkretnej rezerwacji

```
{
    "parkingId": 4,
    "subscriberCarRegistration": "BCD100",
    "bookingDate": "2024-11-14",
    "_links": {
        "self": {
            "href": "http://localhost:8080/bookings/1"
        },
        "booking": {
            "href": "http://localhost:8080/bookings/1"
        }
    }
}
```

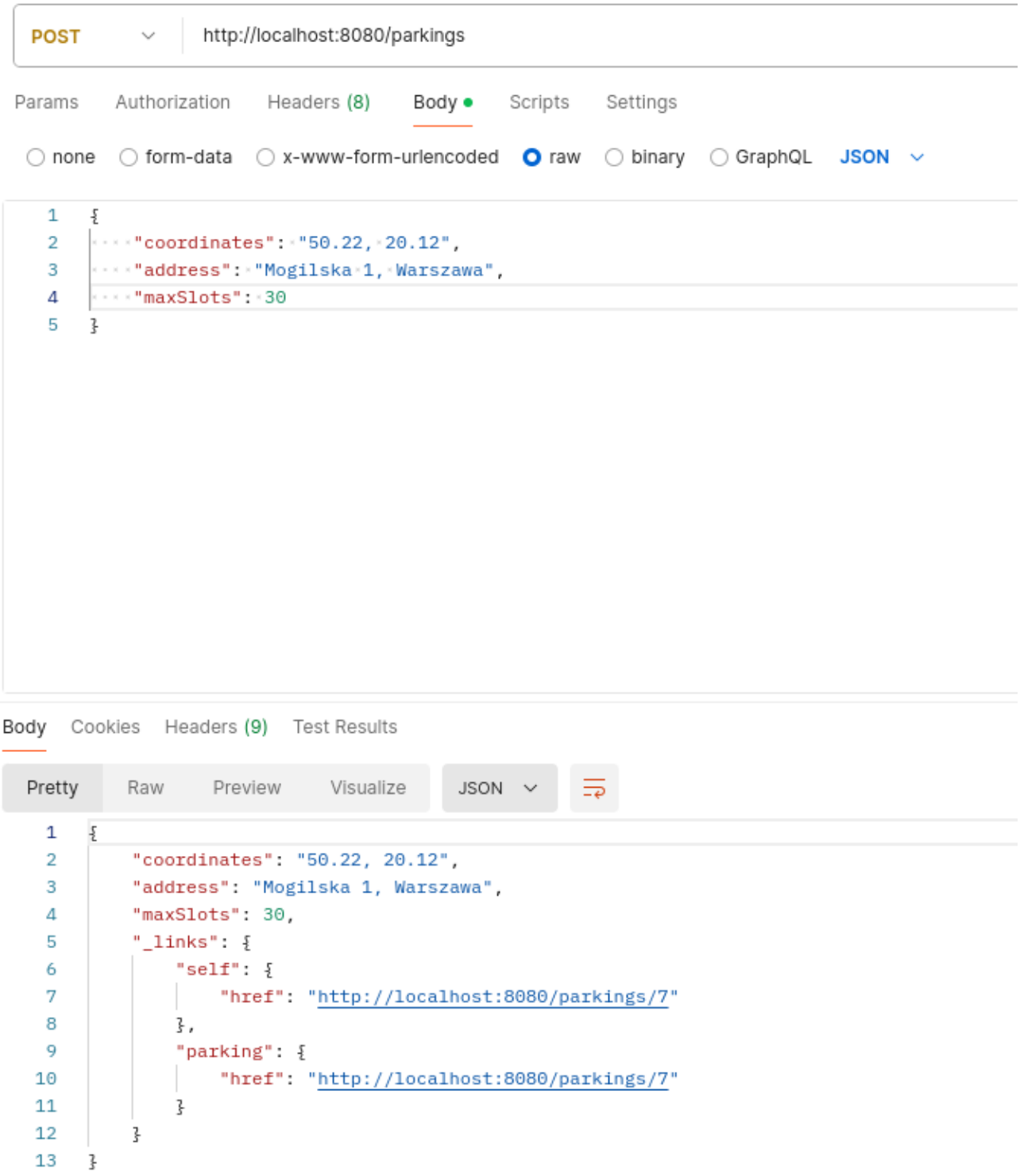

10. http://localhost:8080/parkings/number
    np. http://localhost:8080/parkings/1

> wyświela szczegółowe informacje o konkretnym parkingu

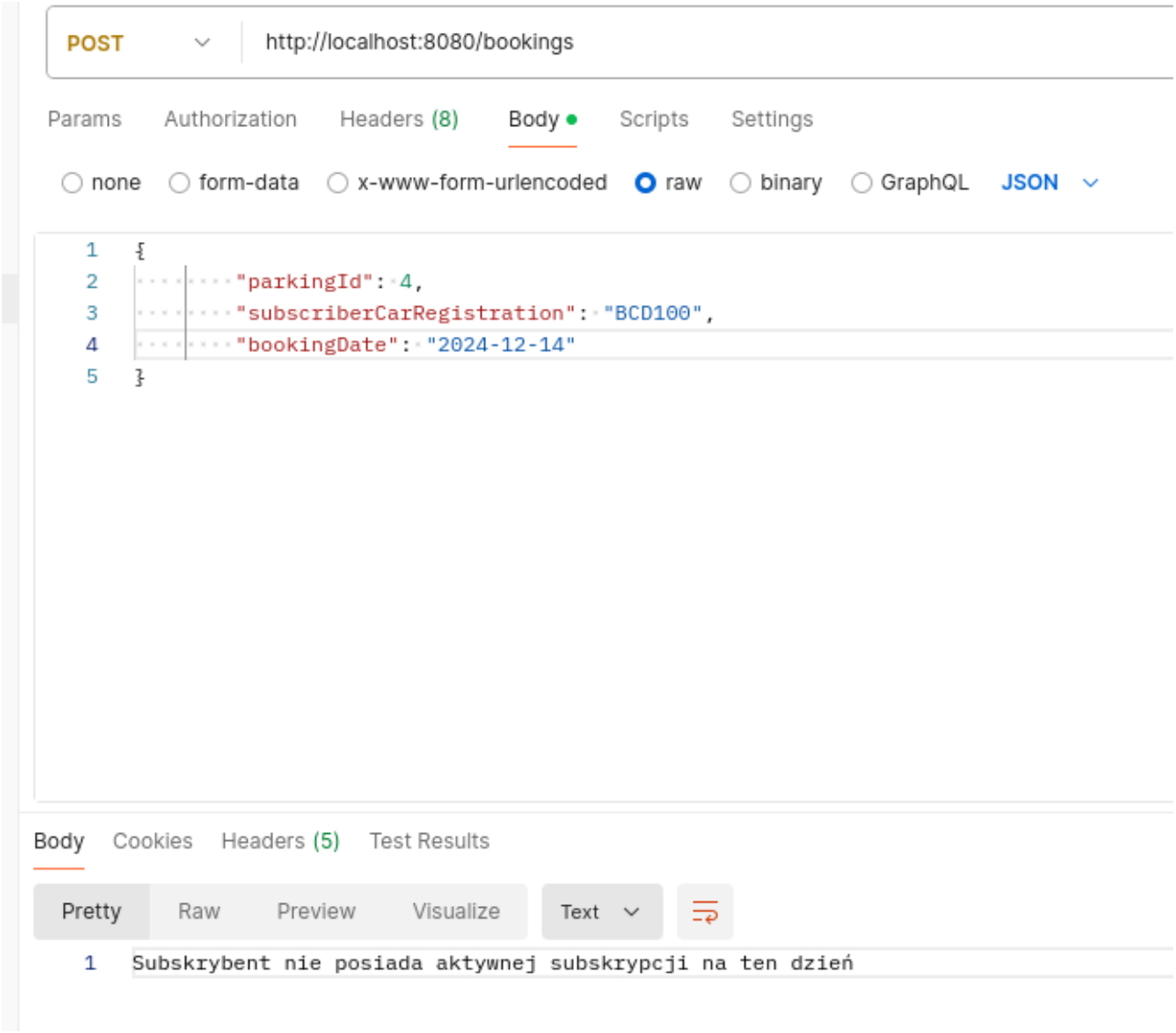

```
{
    "coordinates": "52.22, 21.012",
    "address": "Mazowiecka 1, Warszawa",
    "maxSlots": 40,
    "_links": {
        "self": {
            "href": "http://localhost:8080/parkings/1"
        },
        "parking": {
            "href": "http://localhost:8080/parkings/1"
        }
    }
}
```


Operacje Post

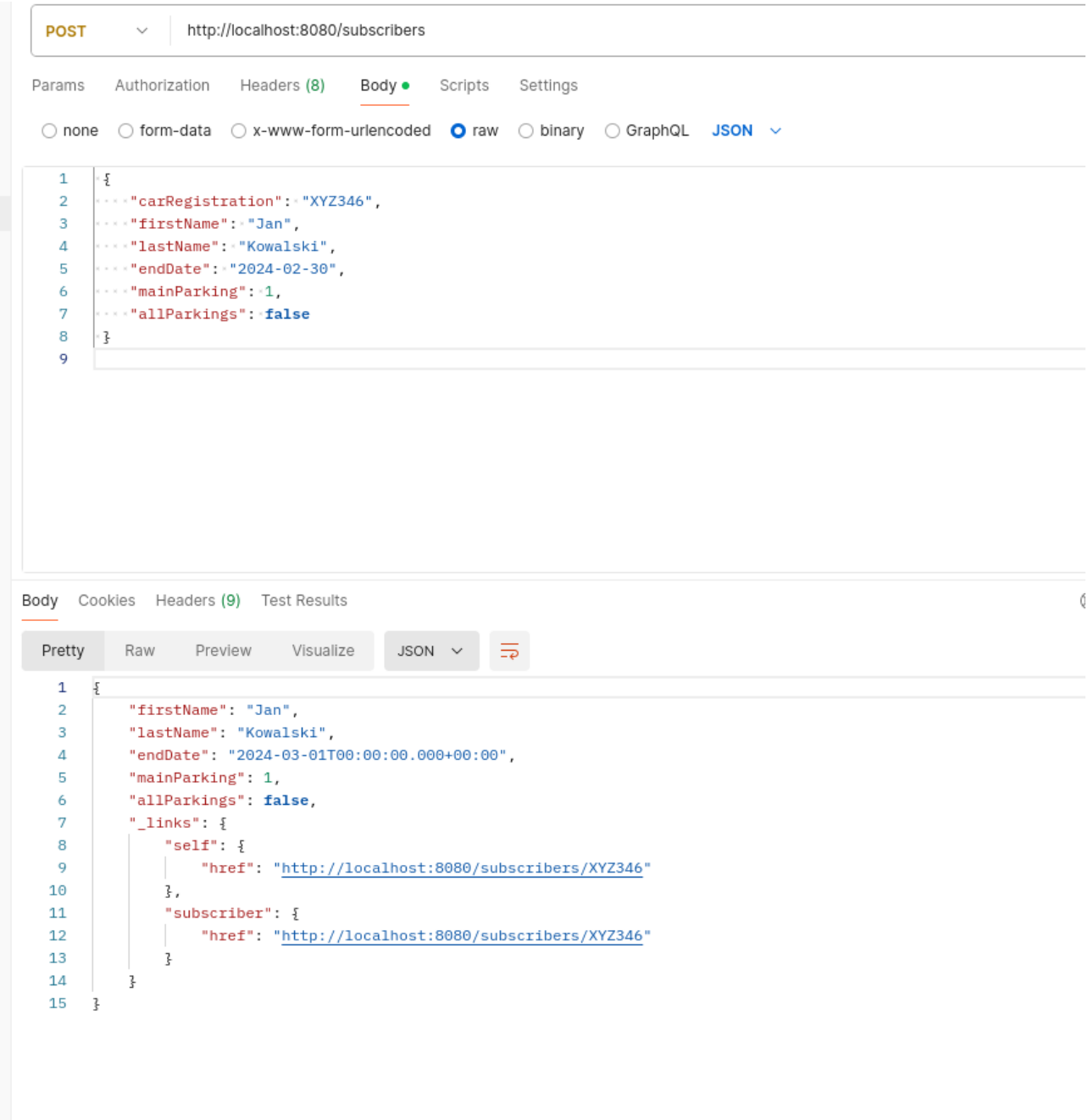
1. Parkingi


POST
http://localhost:8080/parkings
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"coordinates": "50.22, 20.12",
"address": "Mogilska 1, Warszawa",
"maxSlots": 30
}
Body
Cookies
Headers (9)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"coordinates": "50.22, 20.12",
"address": "Mogilska 1, Warszawa",
"maxSlots": 30,
"_links": {
"self": {
"href": "http://localhost:8080/parkings/7"
},
"parking": {
"href": "http://localhost:8080/parkings/7"
}
}
}

2. Rezerwacje

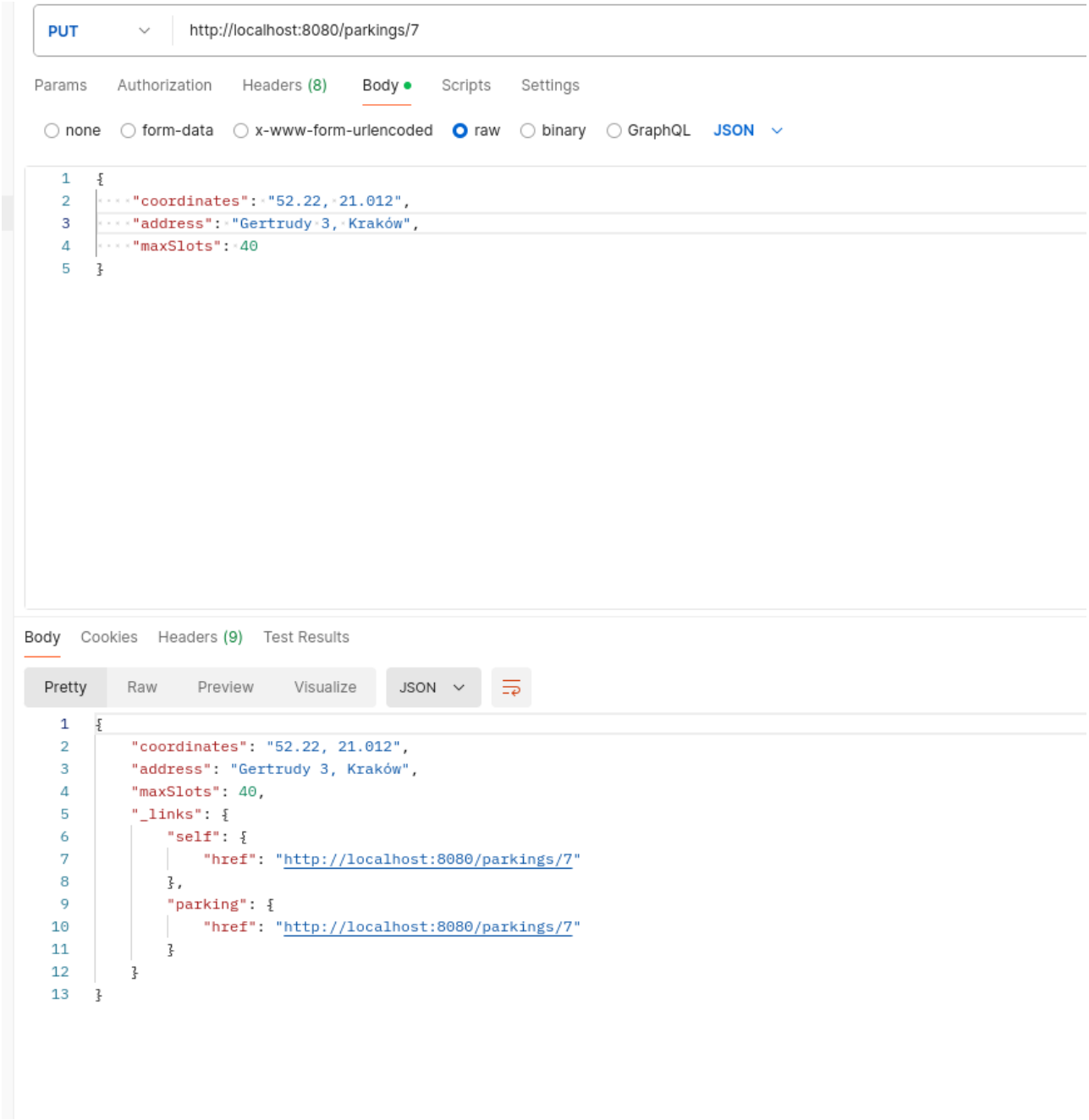


3. Subskrybenci

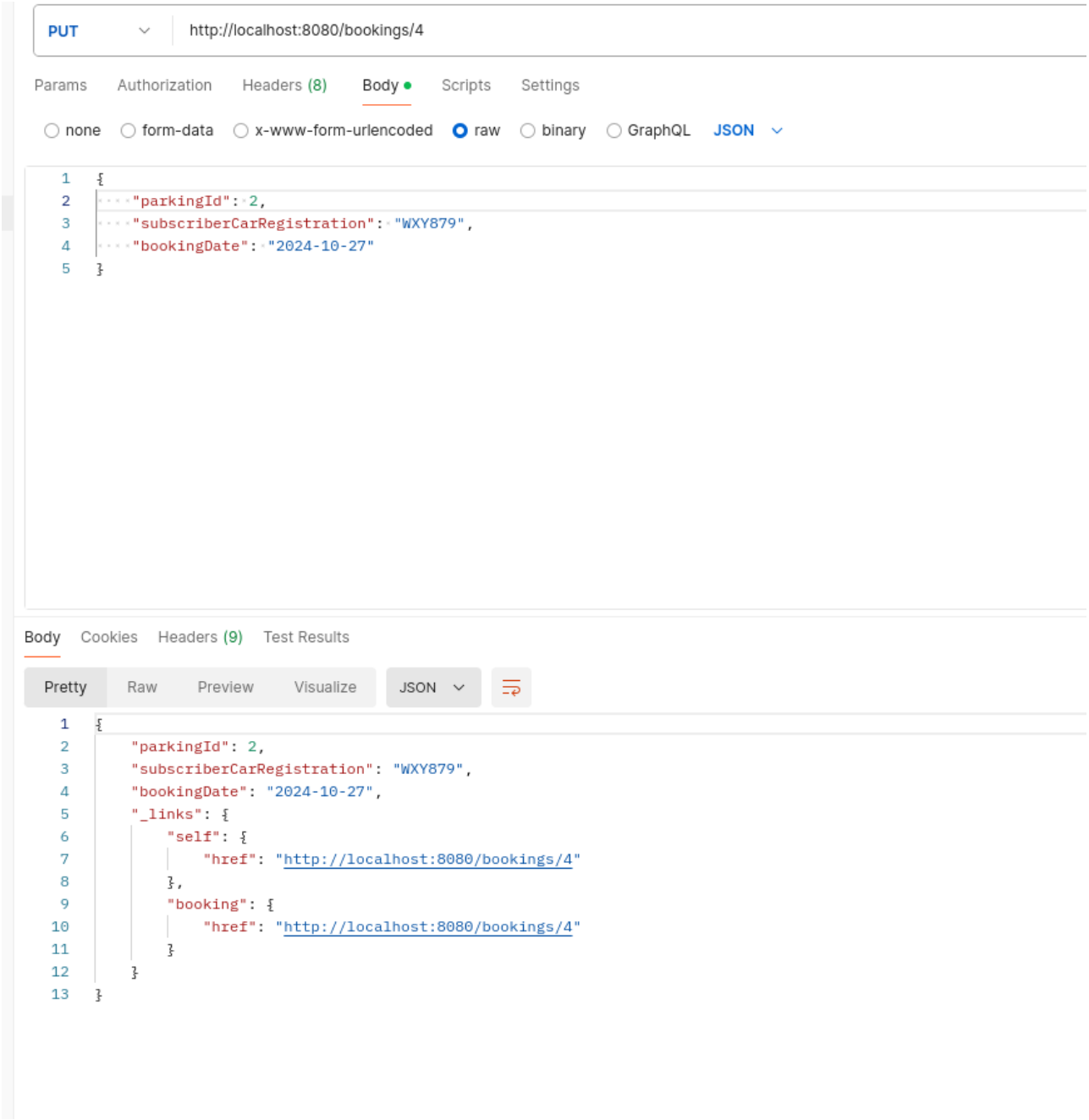


Operacje Put

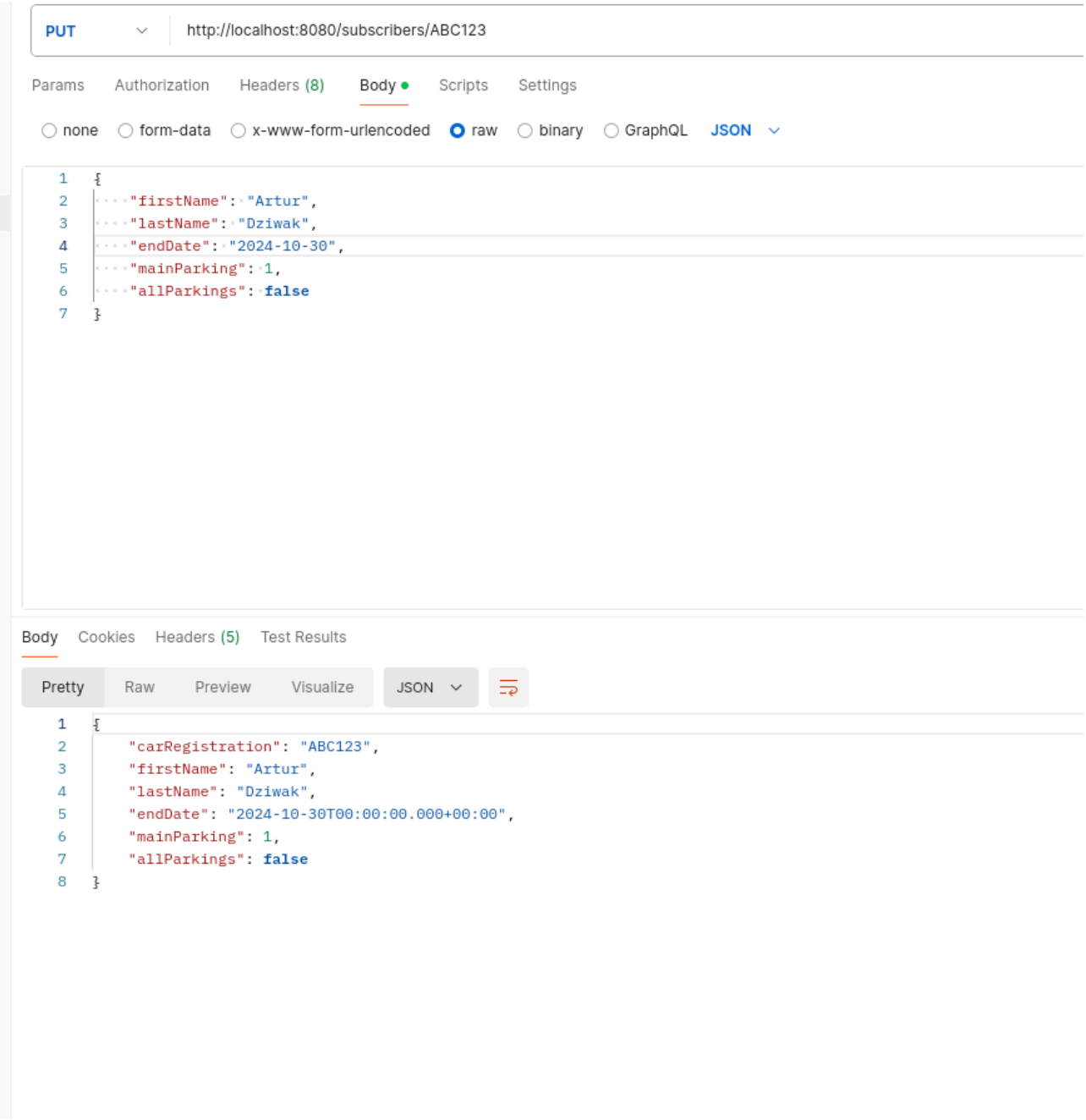
1. Parkingi


PUT
http://localhost:8080/parkings/7
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"coordinates": "52.22, 21.012",
"address": "Gertrudy 3, Kraków",
"maxSlots": 40
}
Body
Cookies
Headers (9)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"coordinates": "52.22, 21.012",
"address": "Gertrudy 3, Kraków",
"maxSlots": 40,
"_links": {
"self": {
"href": "http://localhost:8080/parkings/7"
},
"parking": {
"href": "http://localhost:8080/parkings/7"
}
}
}

2. Rezerwacje

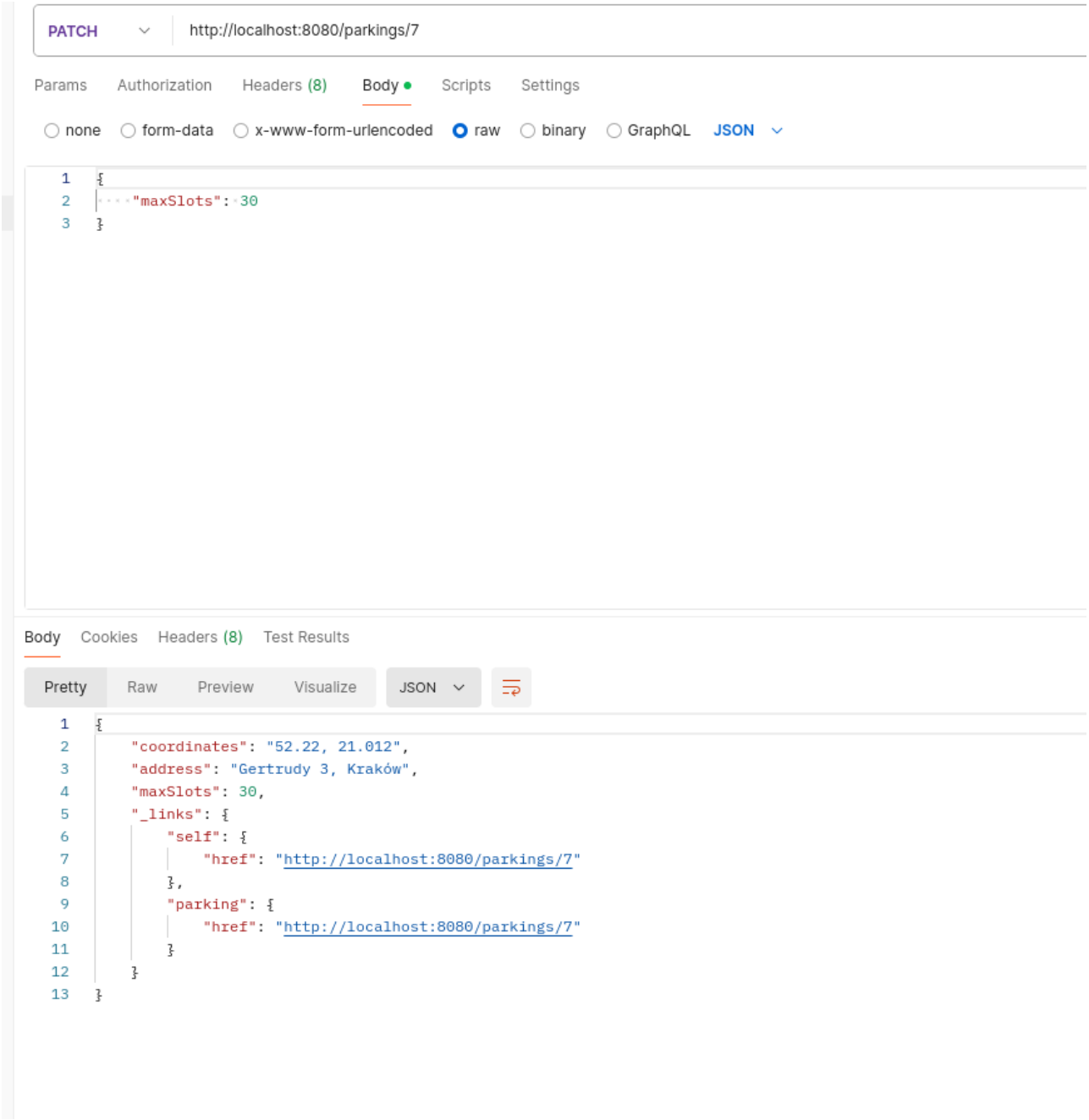


3. Subskrybenci

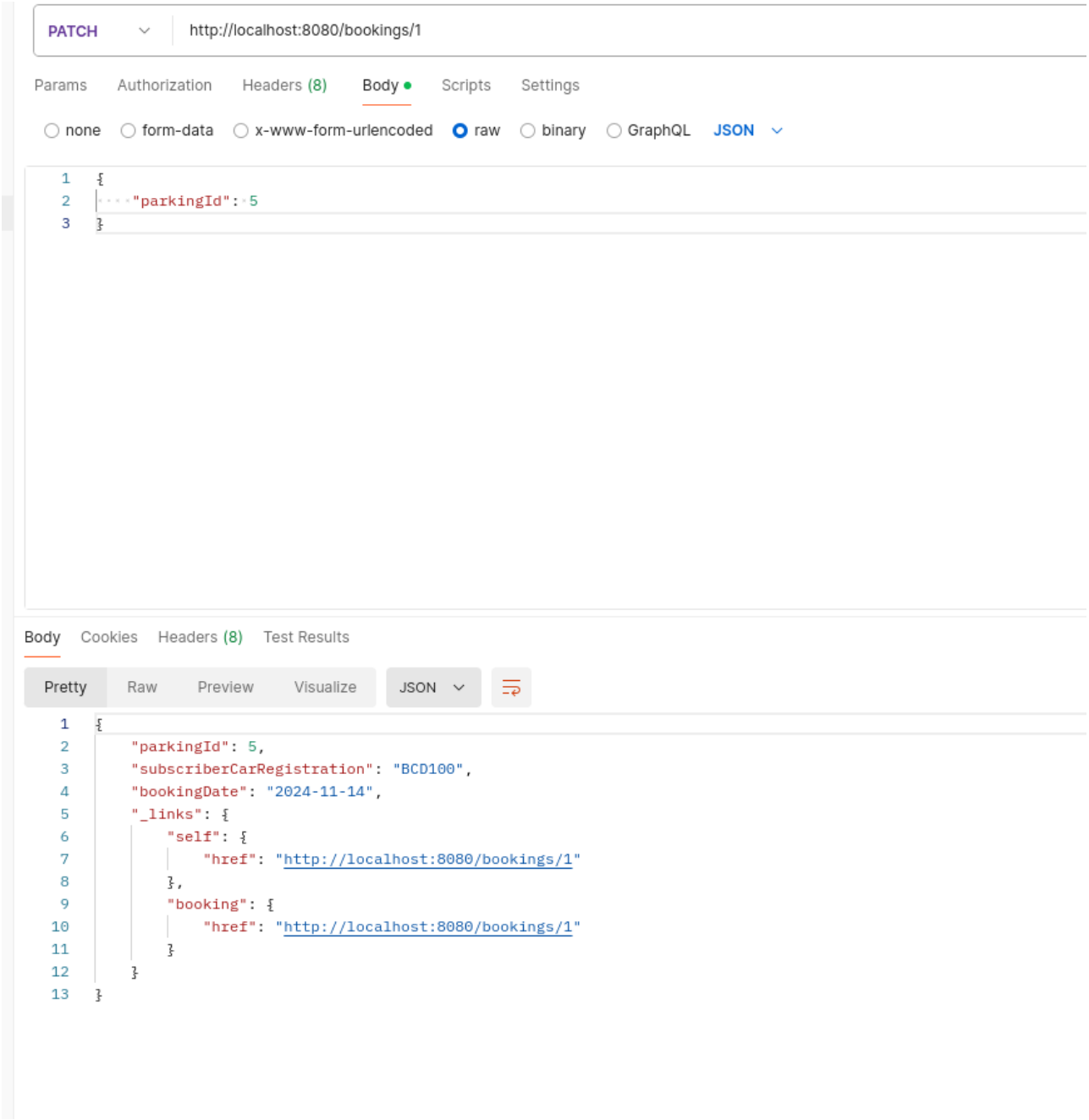


Operacje Patch

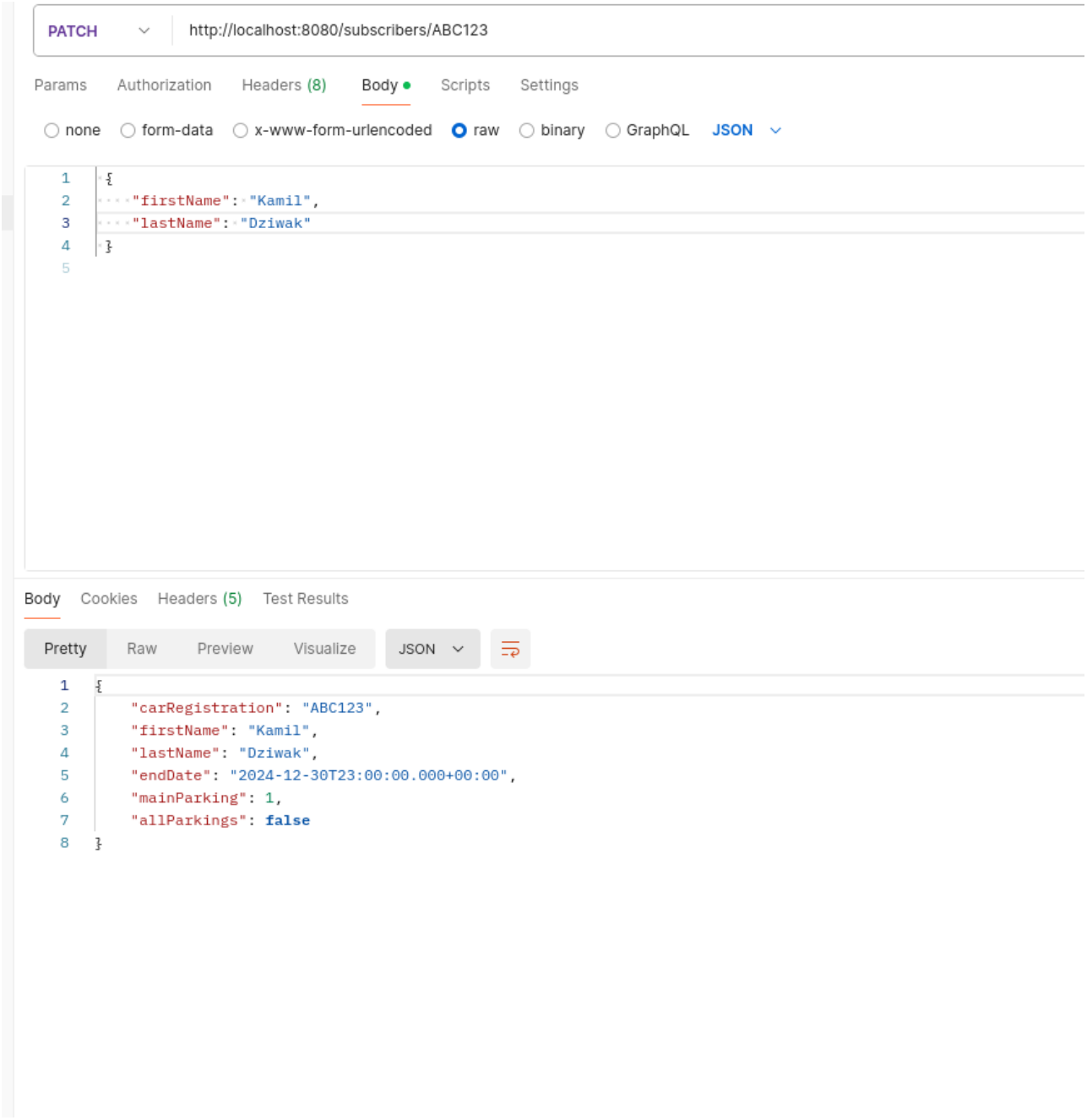
1. Parkingi


PATCH
http://localhost:8080/parkings/7
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"maxSlots": 30
}
Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"coordinates": "52.22, 21.012",
"address": "Gertrudy 3, Kraków",
"maxSlots": 30,
"_links": {
"self": {
"href": "http://localhost:8080/parkings/7"
},
"parking": {
"href": "http://localhost:8080/parkings/7"
}
}
}

2. Rezerwacje

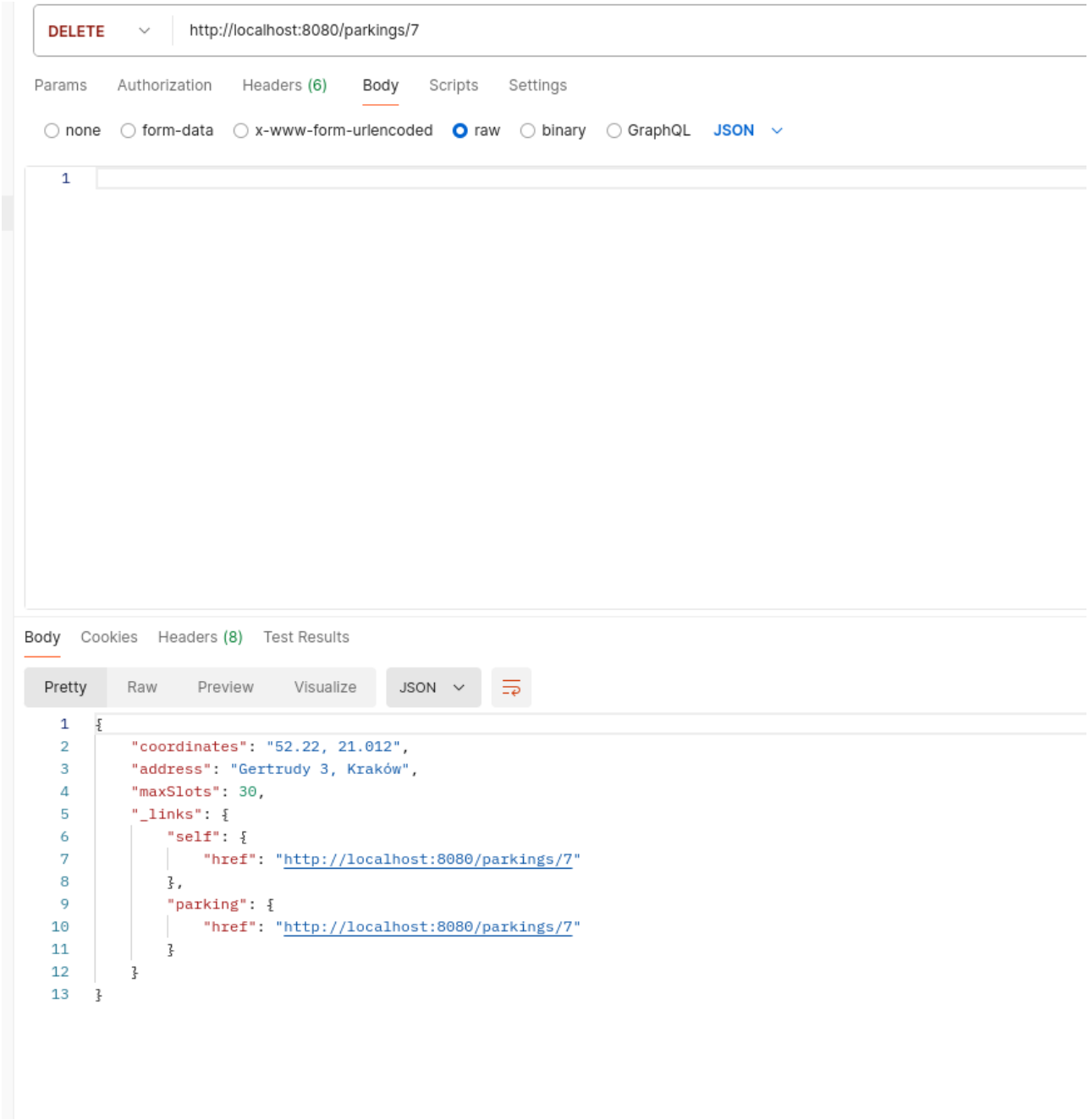


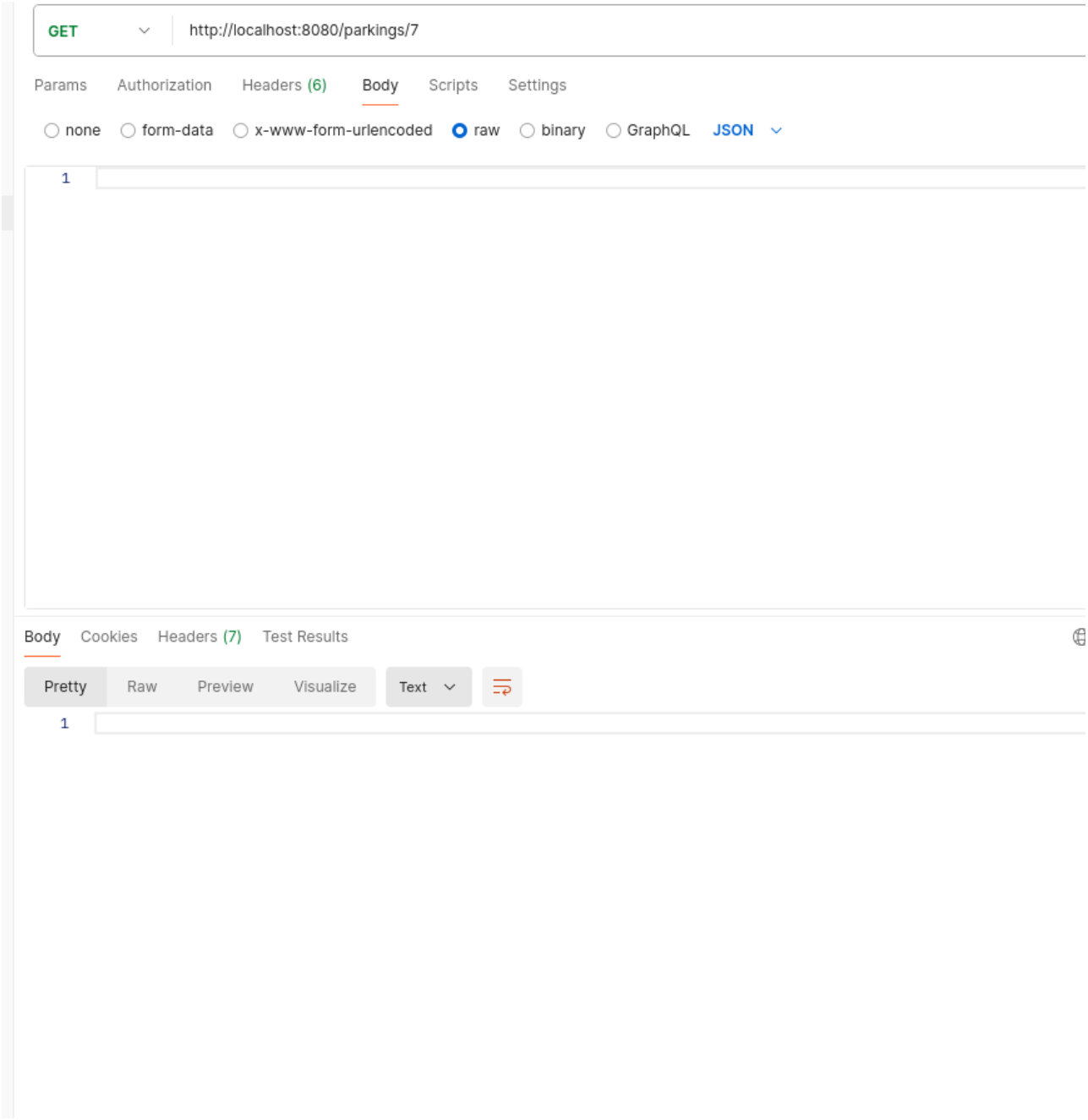
3. Subskrybenci



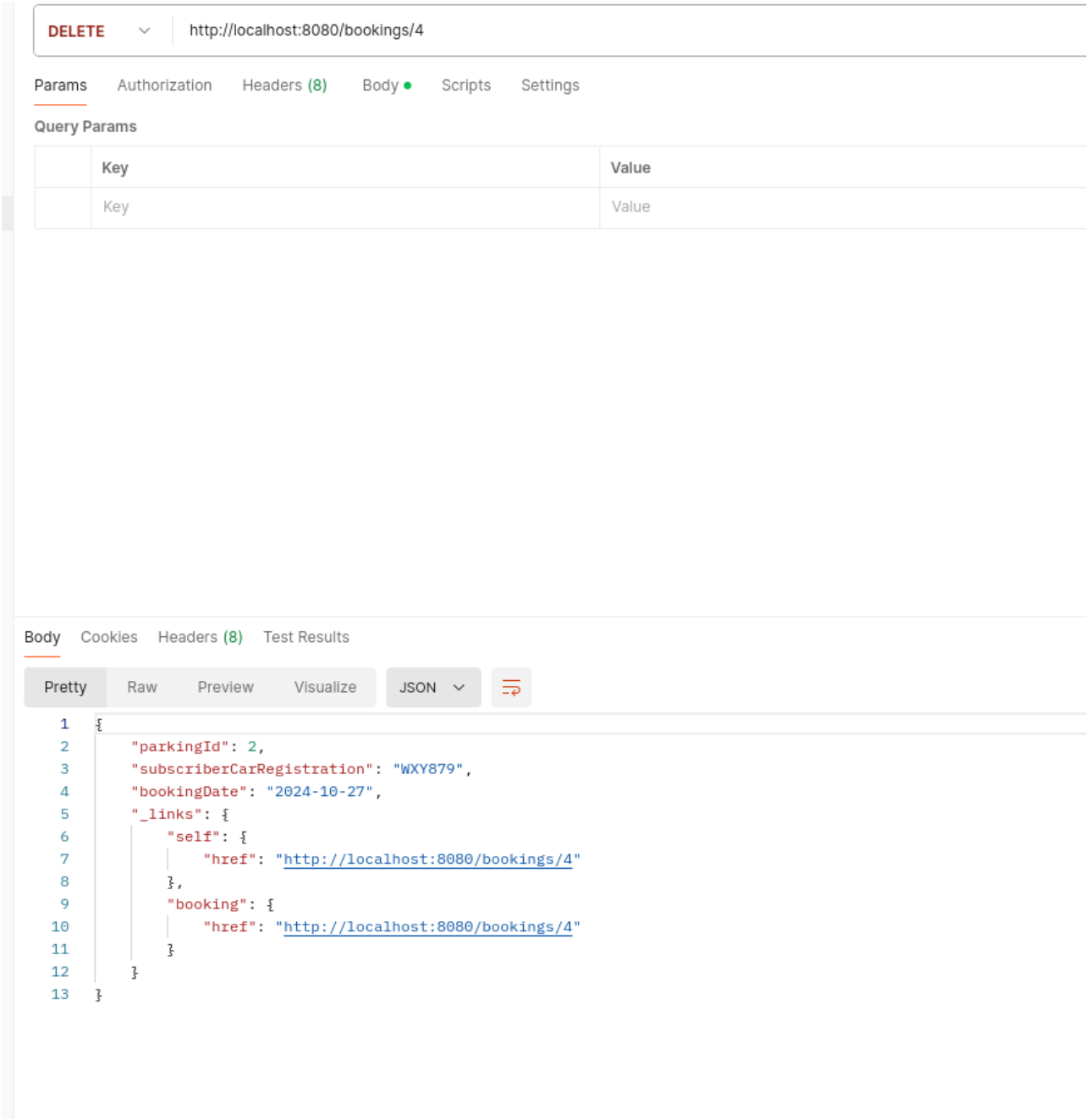
Operacje Delete

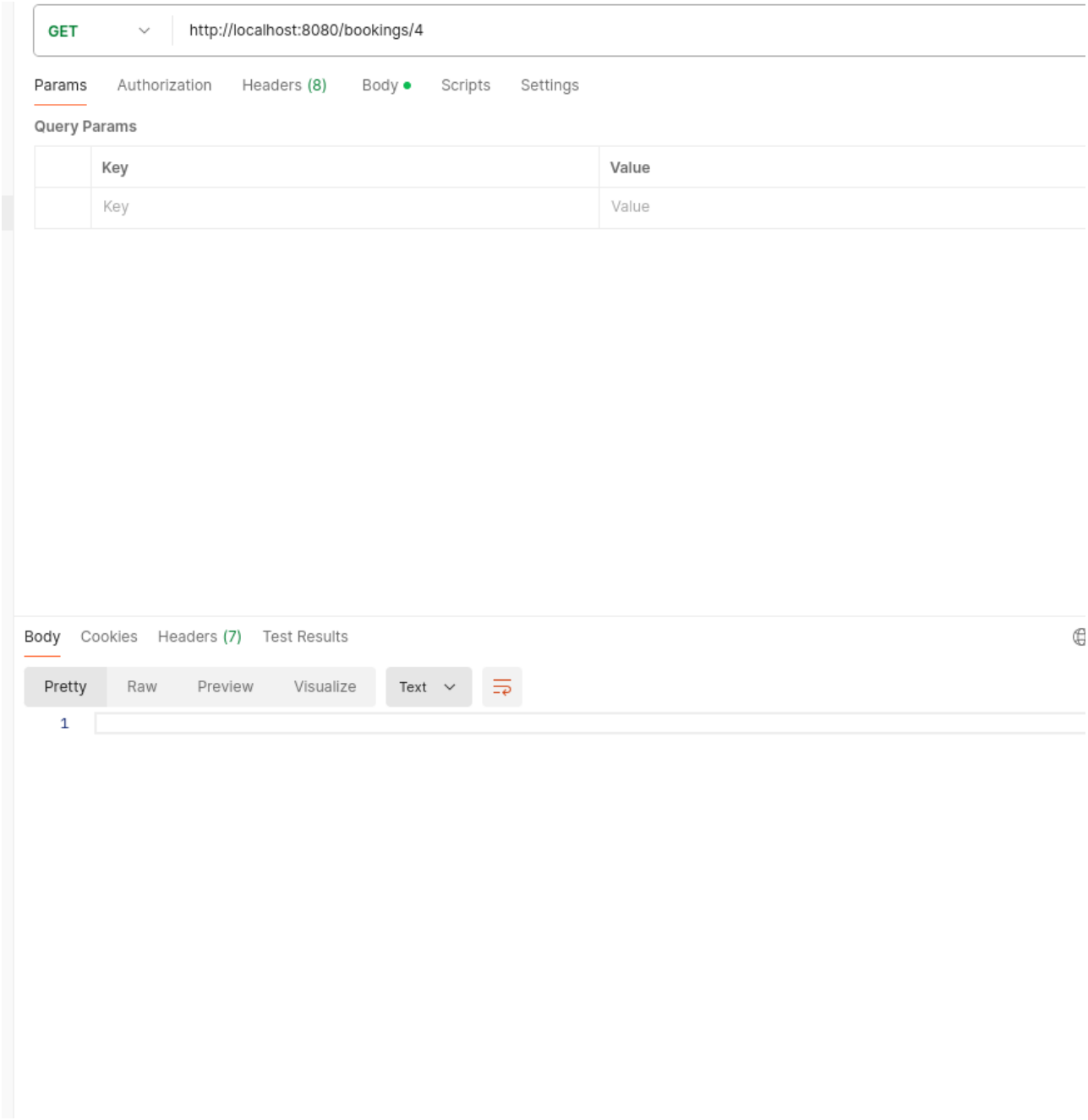
1. Parkingi


DELETE
http://localhost:8080/parkings/7
Params
Authorization
Headers (6)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"coordinates": "52.22, 21.012",
"address": "Gertrudy 3, Kraków",
"maxSlots": 30,
"_links": {
"self": {
"href": "http://localhost:8080/parkings/7"
},
"parking": {
"href": "http://localhost:8080/parkings/7"
}
}
}

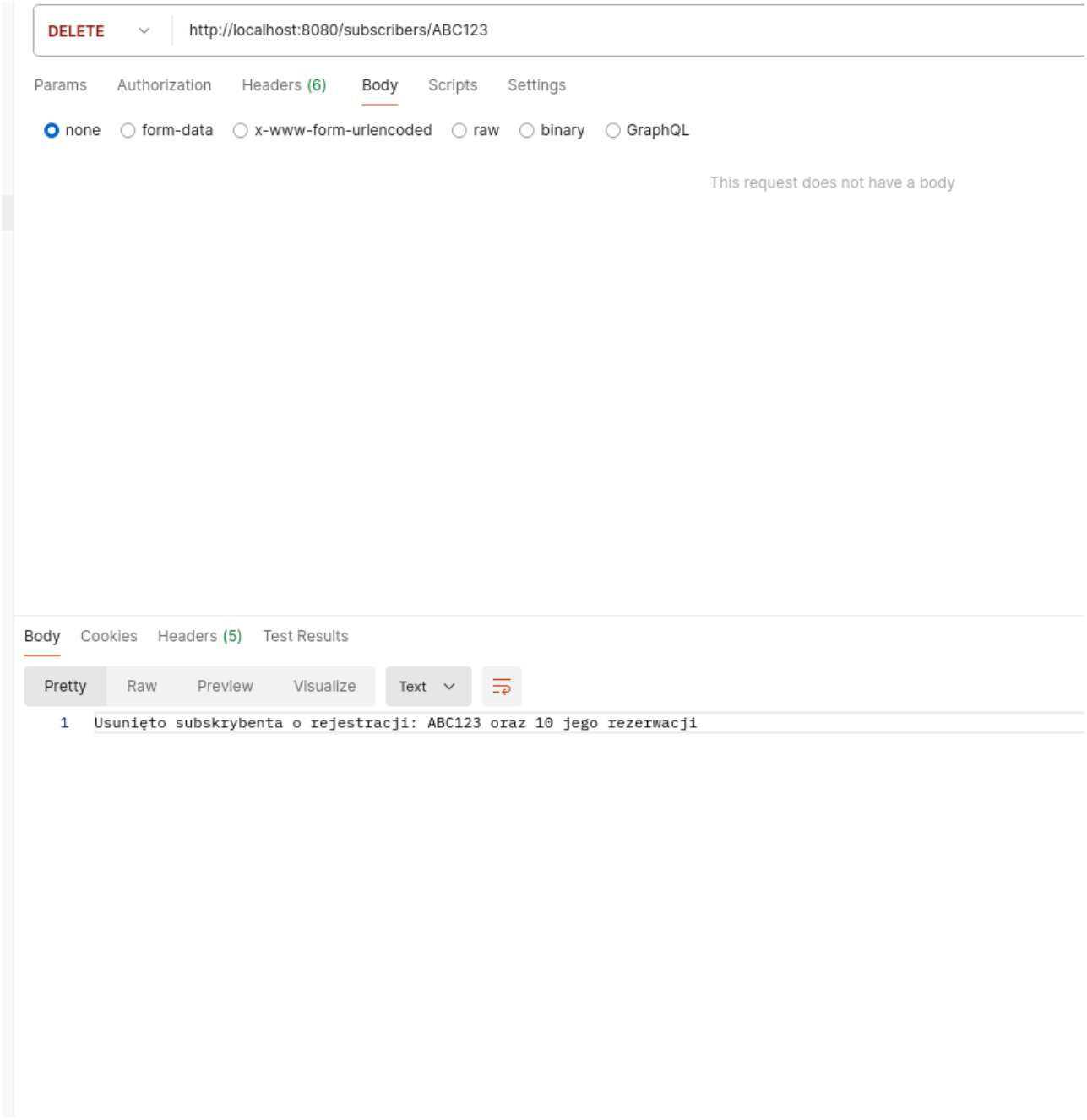

GET
http://localhost:8080/parkings/7
Params
Authorization
Headers (6)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
1
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
Text
1

2. Rezerwacje



3. 
GET
http://localhost:8080/bookings/4
Params
Authorization
Headers (8)
Body
Scripts
Settings
Query Params
Key
Value
Key
Value
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
Text
1

4. Subskrybenci



Dane do logowania

```
spring.application.name=parkingSystem
spring.datasource.url=jdbc:mysql://localhost:3306/parking-system
spring.datasource.username=springstudent
spring.datasource.password=Spr!ngStudent1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.sql.init.mode=always
```