Bazy Danych – Projekt

Zarządzanie Schroniskiem

Arkadiusz Mincberger, Oliwia Radoła, Anna Cichocka

Użyte technologie:

- React – Frontend

- Java Spring Boot – Backend

- MongoDB – Baza

Struktura Bazy:

Baza składa się z 4 kolekcji:

1. Kolekcja Shelter:

Struktura dokumentu:

```
_id: ObjectId('648236ef0b285e55a04a55d0')
name: "Schronisko A"
▾ address: Object
    street: "bukowa"
    postalCode: "12-321"
_class: "com.example.springapi.api.model.Shelter"
```

Reprezentacja w Javie:

```
public class Shelter {
    @Id
    private String id;
    1 usage
    private String name;
    1 usage
    private Address address;
```

Dokumenty kolekcji Shelter przechowują podstawowe informacje na temat schroniska, czyli jego ID, nazwę a także Obiekt address

2. Kolekcja Dog:

Struktura dokumentu:


```
_id: ObjectId('6483316ea0b08d069aafca1b')
name: "johnathan"
weight: 14
sex: "SAMIEC"
age: 12
description: "milutki"
imgSrc: "https://hips.hearstapps.com/hmg-prod/images/dog-puppy-on-garden-royalt…"
state: "ZAADOPTOWANY"
▾ shelter: Object
    _id: ObjectId('6482370f0b285e55a04a55d1')
    name: "Schronisko B"
  ▸ address: Object
_class: "com.example.springapi.api.model.Dog"
```


Reprezentacja w Javie:

```
public class Dog {

    @Id
    private String id;
    1 usage
    private String name;
    1 usage
    private int weight;
    1 usage
    private Sex sex;
    1 usage
    private int age;
    1 usage
    private String description;
    1 usage
    private String imgSrc;
    1 usage
    private State state;
    1 usage
    private Shelter shelter;
```

Dokumenty kolekcji Dog przechowują podstawowe informację na temat psa, gdzie Sex przyjmuje wartości „Samiec” lub „Samiczka”:

```java
public enum Sex {
    SAMIEC, SAMICZKA;

    ArkadiuszMin
    @JsonCreator
    public static Sex fromString(String key){
        for(Sex sex: Sex.values()){
            if(sex.name().equalsIgnoreCase(key)){
                return sex;
            }
        }
        return null;
    }
}
```

Funkcja z andotacją @JsonCreator odpowiada za mapowanie pobranego z bazy stringa na odpowiedni enum, lub wartość null w przypadku braku dopasowania.

A State przyjmuje wartości „Niezarezerwowany”, „Zarezerwowany” lub „Zaadoptowany”:

```java
public enum State {
    2 usages
    NIEZAREZERWOWANY, ZAREZERWOWANY, ZAADOPTOWANY;

    ArkadiuszMin
    @JsonCreator
    public static State fromString(String key){
        for(State state: State.values()){
            if(state.name().equalsIgnoreCase(key)){
                return state;
            }
        }
        return null;
    }
}
```

Do tego Dog posiada pole typu Shelter, modelujące relację One-To-Many Między kolekcjami Shelter oraz Dog.

3. Kolekcja Adopter:

Struktura dokumentu:

```
_id: ObjectId('64849c20b20c3376a3594a6f')
firstName: "Miłosz"
secondName: "Moralny"
phone: "692137420"
email: "aaa@gmail.com"
street: "Aleja Jakaśtam 43/86"
postalCode: "43-543"
city: "Nowy Jork"
_class: "com.example.springapi.api.model.Adopter"
```

Reprezentacja w Javie:

```
public class Adopter {
    @Id
    private String id;
    1 usage
    private String firstName;
    1 usage
    private String secondName;
    1 usage
    private String phone;
    1 usage
    private String email;
    1 usage
    private String street;
    1 usage
    private String postalCode;
    1 usage
    private String city;
```

Dokumenty kolekcji Adopter przechowują podstawowe informacje na temat adopterów.

4. Kolekcja Adoption:

Struktura dokumentu:

```
  _id: ObjectId('648350bcc64aee40494c4faf')
▸ adopter: Object
▸ dog: Object
  dateReservation: 2023-06-08T22:00:00.000+00:00
  state: "ZAADOPTOWANY"
  _class: "com.example.springapi.api.model.Adoption"
```

Reprezentacja w Javie:

```java
public class Adoption {

    @Id
    private String adoptionId;
    1 usage
    private Adopter adopter;
    1 usage
    private Dog dog;
    private LocalDate dateConfirmation;
    1 usage
    private LocalDate dateReservation;
    1 usage
    private State state;
```

Dokumenty kolekcji Adoption przechowują informację na temat zaadoptowanego psa i adoptera który go zaadoptował w ramach konkretnej adopcji, a także stan rezerwacji, datę rezerwacji psa i datę potwierdzenia rezerwacji.

Funkcjonalności Backendu:

Za realizacje funkcjonalności po stronie Backendu odpowiadają 2 klasy:

- <Collection name>Controller – przyjmuje zapytania od Frontendu I wysyła odpowiedzi

- <Collection name>Service – Przyjmuje dane od Controllera I robi operacje typu create/read/update w bazie

1. Kolekcja Pies:

1.1 Funkcja getAllDogs:

DogController:

```java
@GetMapping("/all")
public ResponseEntity<List<Dog>> getAllDogs() { return dogService.getAllDogs(); }
```

DogService:

```java
public ResponseEntity<List<Dog>> getAllDogs(){
    try{
        List<Dog> dogs = dogRepository.findAll();
        return new ResponseEntity<>(dogs, HttpStatus.OK);
    }
    catch(Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Zwraca wszystkie Pieski z bazy wraz ze statusem.

1.2 Funkcja getDogByName:

DogController:

```java
@GetMapping("/byName")
public ResponseEntity<List<Dog>> getDogByName(@RequestParam String name){
    return dogService.getDogByName(name);
}
```

DogService:

```java
public ResponseEntity<List<Dog>> getDogByName(String name){
    try{
        List<Dog> dogs = dogRepository.findDogsByName(name);
        return new ResponseEntity<>(dogs, HttpStatus.OK);
    }
    catch(Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Zwraca listę psów o zadanym imieniu wraz ze statusem.

1.3 Funkcja GetDogByID:

DogController:

```
@GetMapping("/byId")
public ResponseEntity<Dog> getDogById(@RequestParam String id) { return dogService.getDogById(id); }
```

DogService:

```
public ResponseEntity<Dog> getDogById(String id){
    try{
        Optional<Dog> dog =  dogRepository.findDogById(id);
        return dog.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(null, HttpStatus.NOT_FOUND));
    }catch(Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Zwraca psa o zadanym ID. Jeżeli pies o zadanym ID nie znajduje się w bazie, zwraca odpowiedni status i wartość null.

1.4 Funkcja AddDog:

DogController:

```
@PostMapping("/add")
public ResponseEntity<String> addDog(@RequestBody AddDogFormat dogFormat) { return dogService.addDog(dogFormat); }
```

DogService:

```
public ResponseEntity<String> addDog(AddDogFormat dogFormat){
    try{
        if(dogFormat.getName() == null || dogFormat.getSex() == null ||
                dogFormat.getImgSrc() == null || dogFormat.getShelterId() == null){
            return new ResponseEntity<>( body: "You havent provided name, sex, image source or shelterId",
                    HttpStatus.BAD_REQUEST);
        }
        if(dogFormat.getAge() == 0 || dogFormat.getWeight() == 0){
            return new ResponseEntity<>( body: "You havent provided age or weight, or chose it to be 0",
                    HttpStatus.BAD_REQUEST);
        }
        if(dogFormat.getDescription() == null){
            dogFormat.setDescription("Brak");
        }
        Optional<Shelter> shelter = shelterRepository.findShelterById(dogFormat.getShelterId());
        if(shelter.isPresent()){
            Dog dog = new Dog(dogFormat.getName(),
                    dogFormat.getWeight(),
                    dogFormat.getSex(),
                    dogFormat.getAge(),
                    dogFormat.getDescription(),
                    dogFormat.getImgSrc(),
                    State.NIEZAREZERWOWANY,
                    shelter.get());
```

```
            dogRepository.insert(dog);
            return new ResponseEntity<>( body: "Succesfully added a dog to database", HttpStatus.CREATED);
        }else{
            return new ResponseEntity<>( body: "Shelter with provided ID doesnt exists", HttpStatus.NOT_FOUND);
        }
    }
    catch(Exception e){
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Funkcja przyjmuje Informację na temat psa w postaci JSONa a następnie mapuje go na następujący obiekt Javowy z pominięciem pola dogId:

```
public class AddDogFormat {
    private String name;
    private int weight;
    private Sex sex;
    private int age;
    private String description;
    private String imgSrc;
    private String shelterId;
    private String dogId;
}
```

Następnie następuje walidacja wymaganych pól (name, weight, sex, age, description, imgsrc, shelterId), oraz pobranie schroniska o odpowiednim ID z bazy. W przypadku braku znalezienia schroniska, zostaje zwrócona informacja o braku schroniska w bazie. Jeżeli schronisko zostaje znalezione, pies zostaje dodany do bazy.

1.5 Funkcja UpdateDog

DogController:

```
@PutMapping("/update")
public ResponseEntity<String> updateDog(@RequestBody AddDogFormat dogFormat){ return dogService.updateDog(dogFormat);}
```

DogService:

```
public ResponseEntity<String> updateDog(AddDogFormat dogFormat){
    try{
        if(dogFormat.getName() == null || dogFormat.getSex() == null
                || dogFormat.getImgSrc() == null || dogFormat.getShelterId() == null
                || dogFormat.getDogId() == null){
            return new ResponseEntity<>( body: "You havent provided name, sex, image source or shelterId",
                    HttpStatus.BAD_REQUEST);
        }
        if(dogFormat.getAge() == 0 || dogFormat.getWeight() == 0){
            return new ResponseEntity<>( body: "You havent provided age or weight, or chose it to be 0",
                    HttpStatus.BAD_REQUEST);
        }
        Optional<Dog> dog = dogRepository.findDogById(dogFormat.getDogId());
        if(dog.isEmpty()){
            return new ResponseEntity<>( body: "No dog with provided ID", HttpStatus.NOT_FOUND);
        }
        Optional<Shelter> shelter = shelterRepository.findShelterById(dogFormat.getShelterId());
        if(shelter.isEmpty()){
            return new ResponseEntity<>( body: "No shelter with provided ID", HttpStatus.NOT_FOUND);
        }
        Dog foundDog = dog.get();
        foundDog.setName(dogFormat.getName());
        foundDog.setSex(dogFormat.getSex());
        foundDog.setImgSrc(dogFormat.getImgSrc());
        foundDog.setShelter(shelter.get());
        foundDog.setAge(dogFormat.getAge());
        foundDog.setWeight(dogFormat.getWeight());
        foundDog.setDescription(dogFormat.getDescription());

        dogRepository.save(foundDog);

        return new ResponseEntity<>( body: "Dog updated succesfully", HttpStatus.OK);
    }
    catch(Exception e){
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Funkcja przyjmuje te same informację na temat psa co funkcja addDog, z uwzględnieniem pola dogId.

Na starcie następuje walidacja wymaganych pól, a następnie z bazy pobierany jest odpowiedni pies i odpowiednie schronisko. W przypadku braku któregoś z nich zwracany jest odpowiedni status i informacja. Następnie zaktualizowane zostają informację odnośnie psa i zostaje on zapisany do bazy.

1.6 Funkcja deleteDog:

DogController:

```
@DeleteMapping("/delete")
public ResponseEntity<String> deleteDog(@RequestParam String id) { return dogService.deleteDog(id); }
```

DogService:

```
public ResponseEntity<String> deleteDog(String Id){
    try{
        Optional<Dog> dog = dogRepository.findDogById(Id);
        if (dog.isEmpty()){
            return new ResponseEntity<>( body: "No dog with provided ID in database", HttpStatus.NOT_FOUND);
        }
        dogRepository.deleteById(Id);
        return new ResponseEntity<>( body: "Succesfully deleted a dog", HttpStatus.OK);
    }catch (Exception e){
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja przyjmuje Id psa do usunięcia, po czym usuwa go z bazy. W przypadku powodzenia, jak i błędu, zwraca odpowiedni status i informacje.

2. Kolekcja Shelter:

2.1 Funkcja getAllShelters:

ShelterController:

```
@GetMapping("/all")
public ResponseEntity<List<Shelter>> getAllShelters(){
    return shelterService.getAllShelters();
}
```

ShelterService:

```
public ResponseEntity<List<Shelter>> getAllShelters(){
    try{
        List<Shelter> shelters = shelterRepository.findAll();
        return new ResponseEntity<>(shelters, HttpStatus.OK);
    }
    catch(Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Zwraca wszystkie schroniska z bazy wraz ze statusem

2.2 Funkcja getShelterById:

ShelterController:

```
@GetMapping("/byId")
public ResponseEntity<Shelter> getShelterById(@RequestParam String id){
    try{
        Optional<Shelter> shelter = shelterService.getShelterById(id);
        return new ResponseEntity<>((Shelter) shelter.orElse( other: null), HttpStatus.OK);
    }
    catch(Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

ShelterService:

```
public Optional<Shelter> getShelterById(String id) throws Exception{
    Optional<Shelter> shelter = shelterRepository.findShelterById(id);
    return shelter;
}
```

Zwraca Schronisko o podanym ID. W przypadku braku schroniska o podanym ID zwraca wartość null oraz odpowiedni status.

.

2.3 Funkcja AddShelter:

ShelterController:

```
@PostMapping("/add")
public ResponseEntity<String> addShelter(@RequestBody AddShelterFormat addShelterFormat){
    return shelterService.addShelter(addShelterFormat);
}
```

ShelterService:

```
public ResponseEntity<String> addShelter(AddShelterFormat addShelterFormat){
    try{
        if(addShelterFormat.getName() == null || addShelterFormat.getStreet() == null || addShelterFormat.getPostal() == null){
            return new ResponseEntity<>( body: "You need to provide name, street and postal code of shelter", HttpStatus.BAD_REQUEST);
        }
        Optional<Shelter> foundShelter = shelterRepository.findShelterByName(addShelterFormat.getName());
        if (foundShelter.isPresent()){
            return new ResponseEntity<>( body: "Shelter with the name " + foundShelter.get().getName() + " already exists",
                    HttpStatus.BAD_REQUEST);
        }else{
            Address address = new Address(addShelterFormat.getStreet(), addShelterFormat.getPostal());
            Shelter shelter = new Shelter(addShelterFormat.getName(), address);
            shelterRepository.insert(shelter);
            return new ResponseEntity<>( body: "Shelter added successfully!", HttpStatus.CREATED);
        }
    }catch(Exception e){
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Funkcja przyjmuje informację na temat schroniska w postaci JSONa, a następnie mapuje go na następujący obiekt Javowy:

```java
@Data
public class AddShelterFormat {
    private String name;
    private String street;
    private String postal;
}
```

Początkowo następuje walidacja wymaganych danych (wszystkich).
Następnie jest sprawdzane czy schronisko o danej nazwie już istnieje, jeżeli tak zwracana jest odpowiednia informacja.
W przeciwnym razie do bazy dodawane jest nowe schronisko.

3. Kolekcja Adopter

3.1 Funkcja getAllAdopters

```java
@GetMapping("/all")
public ResponseEntity<List<Adopter>> getAllAdopters(){
    return adopterService.getAllAdopters();
}
```

```java
public ResponseEntity<List<Adopter>> getAllAdopters(){
    try {
        return new ResponseEntity<>(adopterRepository.findAll(), HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca wszystkich adoptujących.

3.2 Funkcja getAdopterById

```java
@GetMapping("/byId")
public ResponseEntity<Adopter> getAdopterById(@RequestParam String id){
    return adopterService.getAdopterById(id);
}
```

```
public ResponseEntity<Adopter> getAdopterById(String Id){
    try{
        Optional<Adopter> adopter =  adopterRepository.findAdopterById(Id);
        if(adopter.isEmpty()){
            return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(adopter.get(),HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null,HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Funkcja zwraca adoptującego o id podanym w argumencie.

4. Kolekcja Adoption

4.1 Funkcja getAdoptionByAdoptionId

```
@GetMapping("/byAdoptionId")
public ResponseEntity<Adoption> getAdoptionByAdoptionId(@RequestParam String id){
    return adoptionService.getAdoptionByAdoptionId(id);
}
```

```
public ResponseEntity<Adoption> getAdoptionByAdoptionId(String id){
    try{
        Optional<Adoption> adoption = adoptionRepository.findAdoptionByAdoptionId(id);
        if(adoption.isEmpty()){
            return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(adoption.get(),HttpStatus.OK);
    }catch (Exception e){
        return new ResponseEntity<>(null,HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca adopcję o adoptionId podanym w argumencie.

4.2 Funckja getAdoptionByDogId

```java
@GetMapping("/byDogId")
public ResponseEntity<Adoption> getAdoptionByDogId(@RequestParam String id){
    return adoptionService.getAdoptionByDogId(id);
}
```

```java
public ResponseEntity<Adoption> getAdoptionByDogId(String id){
    try{
        Optional<Adoption> adoption = adoptionRepository.findAdoptionByDogId(id);
        if(adoption.isEmpty()){
            return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(adoption.get(),HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null,HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca adopcję o dogId podanym w argumencie.

4.3 Funkcja getAdoptionsByAdopterId

```java
@GetMapping("/byAdopterId")
public ResponseEntity<List<Adoption>> getAdoptionByAdopterId(@RequestParam String id){
    return adoptionService.getAdoptionsByAdopterId(id);
}
```

```java
public ResponseEntity<List<Adoption>> getAdoptionsByAdopterId(String id){
    try {
        return new ResponseEntity<>(adoptionRepository.findAdoptionsByAdopterId(id),HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca adopcję o adopterId podanym w argumencie.

4.4 Funkcja getAllAdoptions

```java
@GetMapping("/all")
public ResponseEntity<List<Adoption>> getAllAdoptions(){
    return adoptionService.getAllAdoptions();
}
```

```java
public ResponseEntity<List<Adoption>> getAllAdoptions(){
    try {
        return new ResponseEntity<>(adoptionRepository.findAll(), HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca wszystkie adopcje.

4.5 Funkcja addAdoption

```
@PostMapping("/add")
public ResponseEntity<String> addAdoption(@RequestBody AddAdoptionFormat addAdoptionFormat){
    return adoptionService.addAdoption(addAdoptionFormat);
}
```

```
public ResponseEntity<String> addAdoption(AddAdoptionFormat adoption){
    try {
        Optional<Dog> _dog = dogRepository.findDogById(adoption.getDogId());
        if (_dog.isEmpty()) {
            return new ResponseEntity<>( body: "No dog with given id exists", HttpStatus.BAD_REQUEST);
        }
        if (_dog.get().getState() != State.NIEZAREZERWOWANY) {
            return new ResponseEntity<>( body: "Dog already reserved", HttpStatus.BAD_REQUEST);
        }
        if (adoption.getFirstName() == null || adoption.getSecondName() == null) {
            return new ResponseEntity<>( body: "No first name or second name provided", HttpStatus.BAD_REQUEST);
        }
        if (adoption.getPhone() == null || adoption.getEmail() == null) {
            return new ResponseEntity<>( body: "No phone number or email provided", HttpStatus.BAD_REQUEST);
        }
        if (adoption.getCity() == null || adoption.getPostalCode() == null || adoption.getStreet() == null) {
            return new ResponseEntity<>( body: "No valid address provided", HttpStatus.BAD_REQUEST);
        }

        Adopter newAdopter = new Adopter(
                adoption.getFirstName(),
                adoption.getSecondName(),
                adoption.getPhone(),
                adoption.getEmail(),
                adoption.getStreet(),
                adoption.getPostalCode(),
                adoption.getCity());

        //check if adopter already exists
        ExampleMatcher matcher = ExampleMatcher.matchingAll()
                .withIgnoreNullValues()
                .withIgnorePaths( ...ignoredPaths: "_id");

        Example<Adopter> example = Example.of(newAdopter, matcher);
        Optional<Adopter> foundAdopter = adopterRepository.findOne(example);
```

```
        Adopter adopter;
        if (foundAdopter.isPresent()) {
            adopter = foundAdopter.get();
        } else {
            adopter = newAdopter;
            adopterRepository.insert(adopter);
        }

        Dog dog = _dog.get();
        Adoption newAdoption = new Adoption(adopter, dog);
        adoptionRepository.insert(newAdoption);
        dog.setState(State.ZAREZERWOWANY);
        dogRepository.save(dog);

        return new ResponseEntity<>( body: "Successfully added new adoption", HttpStatus.CREATED);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja waliduje otrzymane dane o adopcji, sprawdza czy dany piesek istnieje oraz jest wolny, a następnie sprawdza czy adoptujący o podanych danych już istnieje. Jeśli tak, to jest on przypisany do nowej adopcji. Jeśli nie – tworzony jest nowy adoptujący. Nowo utworzony obiekt Adoption dodawany jest do bazy.

4.6 Funkcja confirmAdoption

```
@GetMapping("/confirm")
public ResponseEntity<String> confirmAdoption(@RequestParam String id){
    return adoptionService.confirmAdoption(id);
}
```

```
public ResponseEntity<String> confirmAdoption(String adoptionId){
    try{
    Optional<Adoption> adoption = adoptionRepository.findById(adoptionId);
    if(adoption.isEmpty()){
        return new ResponseEntity<>( body: "No adoption with given id exists", HttpStatus.BAD_REQUEST);
    }
    if(adoption.get().getState() == State.ZAADOPTOWANY){
        return new ResponseEntity<>( body: "Dog already adopted", HttpStatus.BAD_REQUEST);
    }

    Dog dog = adoption.get().getDog();
    adoption.get().setState(State.ZAADOPTOWANY);
    adoption.get().setDateConfirmation(LocalDate.now());
    dog.setState(State.ZAADOPTOWANY);
    adoptionRepository.save(adoption.get());
    dogRepository.save(dog);
    return new ResponseEntity<>( body: "Successfully updated adoption status", HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja przujmuje adoptionId i zmienia status adopcji na zaadoptowany.

4.7 Funkcja getAllToConfirm

```
@GetMapping("/toConfirm")
public ResponseEntity<List<Adoption>>  getAllToConfirm(){
    return  adoptionService.getAllToConfirm();
}
```

```
public ResponseEntity<List<Adoption>> getAllToConfirm(){
    try {
        return new ResponseEntity<>(adoptionRepository.findAdoptionByState(State.ZAREZERWOWANY), HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca wszystkie adopcje do zatwierdzenia – czyli takie o statusie ”reserved”

4.8 Funkcja getAllAdopted

```java
@GetMapping("/allAdopted")
public ResponseEntity<List<Adoption>>  getAllAdopted(){
    return  adoptionService.getAllAdopted();
}
```

```java
public ResponseEntity<List<Adoption>> getAllAdopted(){
    try {
        return new ResponseEntity<>(adoptionRepository.findAdoptionByState(State.ZAADOPTOWANY), HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca wszystkie adopcje o statusie o statusie ”adopted”

4.9 Funkcja getShelterData

```java
@GetMapping("/data")
public ResponseEntity<SheltersDataFormat> getShelterData(){
    return adoptionService.getSheltersData();
}
```

```java
public ResponseEntity<SheltersDataFormat> getSheltersData(){
    try {
        SheltersDataFormat data = new SheltersDataFormat();
        int adopted = dogRepository.findDogsByState(State.ZAADOPTOWANY).size();
        data.setDogCount(dogRepository.findAll().size() - adopted);
        data.setAdoptedCount(adopted);
        data.setAdoptersCount(adopterRepository.findAll().size());
        return new ResponseEntity<>(data,HttpStatus.OK);
    } catch (Exception e){
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Funkcja zwraca dane o:

- ilości piesków w schronisku
- Ilości adoptowanych piesków
- Ilości adoptujących

w postaci jednego obiektu