```java
//PetStoreApplication
package pet.store;

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

@SpringBootApplication
public class PetStoreApplication {

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

    }

}



//PetStoreController

package pet.store.controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import lombok.extern.slf4j.Slf4j;
import pet.store.controller.model.PetStoreData;
import pet.store.controller.model.PetStoreData.PetStoreCustomer;
import pet.store.controller.model.PetStoreData.PetStoreEmployee;
import pet.store.entity.PetStore;
import pet.store.service.PetStoreService;

//telling springboot that this is a rest controller
@RestController
@Slf4j
@RequestMapping("/pet_store")
public class PetStoreController {
    @Autowired
    private PetStoreService petStoreService;

    // PET STORE REQUEST
    @PostMapping
    @ResponseStatus(code = HttpStatus.CREATED)
    public PetStoreData insertPetStore(@RequestBody PetStoreData petStoreData) {
        log.info("Creating pet store {}", petStoreData);
        return petStoreService.savePetStore(petStoreData);
    }

    @PutMapping("/{petStoreId}")
    public PetStoreData updatePetStore(@PathVariable Long petStoreId, @RequestBody PetStoreData petStoreData) {
        petStoreData.setPetStoreId(petStoreId);
        log.info("Updating pet store {}", petStoreData);
        return petStoreService.savePetStore(petStoreData);
    }

    @GetMapping("/{petStoreId}")
    public PetStore retrievePetStoreById(@PathVariable Long petStoreId) {
        log.info("Retrieving pet store with ID={}", petStoreId);
        return petStoreService.findPetStoreById(petStoreId);
    }

    @GetMapping
    public List<PetStoreData> retrieveAllPetStores() {
        log.info("Retrieving all pet stores called.");
        return petStoreService.retrieveAllPetStores();
```

```
    }

    @DeleteMapping
    public void deleteAllPetStores(){
        log.info("Attempting to delete all Pet Stores");
        throw new UnsupportedOperationException("Deleting all Pet Stores is not allowed.");
    }

    @DeleteMapping("/{petStoreId}")
    public Map<String, String> deletePetStoreById(@PathVariable Long petStoreId) {
        log.info("Deleting pet store with ID= ", +petStoreId);
        petStoreService.deletePetStoreById(petStoreId);

        return Map.of("message", "Pet store with ID= " + petStoreId + " was deleted");
    }



    // EMPLOYEE REQUEST
    @PostMapping("/{petStoreId}/employee")
    @ResponseStatus(code = HttpStatus.CREATED)
    public PetStoreEmployee insertEmployee(@PathVariable Long petStoreId, @RequestBody PetStoreEmployee petStoreEmp
        log.info("Creating pet store employee {}", petStoreEmployee);

        return petStoreService.saveEmployee(petStoreId, petStoreEmployee);
    }

    @PutMapping("/{petStoreId}/employee/{employeeId}")
    public PetStoreEmployee updateEmployee(@PathVariable Long petStoreId, @PathVariable Long employeeId,
            @RequestBody PetStoreEmployee petStoreEmployee) {
        log.info("Updating employee with ID={} at pet store with ID={}", employeeId, employeeId);
        petStoreEmployee.setEmployeeId(employeeId);
        return petStoreService.saveEmployee(petStoreId, petStoreEmployee);
    }

    // CUSTOMER REQUEST
    @PostMapping("/{petStoreId}/customer")
    @ResponseStatus(code = HttpStatus.CREATED)
    public PetStoreCustomer insertCustomer(@PathVariable Long petStoreId,
            @RequestBody PetStoreCustomer petStoreCustomer) {
        log.info("Adding pet store customer {} to pet store with id {}", petStoreCustomer, petStoreId);

        return petStoreService.saveCustomer(petStoreId, petStoreCustomer);
    }



}


//GlobalErrorHandler

package pet.store.controller.error;

import java.util.HashMap;
import java.util.Map;
import java.util.NoSuchElementException;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import lombok.extern.slf4j.Slf4j;

@RestControllerAdvice
@Slf4j
public class GlobalErrorHandler {

    @ExceptionHandler(NoSuchElementException.class)
    @ResponseStatus(code = HttpStatus.NOT_FOUND)
    public Map<String, String> handleNoSuchErrorException(NoSuchElementException exception) {
        log.info("NoSuchElementException occured: {}", exception.getMessage());

        Map<String, String> errorResponse = new HashMap<String, String>();
        errorResponse.put("message", exception.toString());
```

```
        return errorResponse;
    }

    @ExceptionHandler(UnsupportedOperationException.class)
    @ResponseStatus(code = HttpStatus.METHOD_NOT_ALLOWED)
    public Map<String, String> hadleUnsupportedOPerationException(UnsupportedOperationException exception) {
        log.info("UnsupportedOperationException occured: {}", exception.getMessage());
        Map<String, String> errorResponse = new HashMap<String, String>();
        errorResponse.put("message", exception.toString());

        return errorResponse;
    }

    @ExceptionHandler(Exception.class)
    @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
    public Map<String, String> handleException(Exception exception) {
        log.info("Exception occured: {}", exception.getMessage());
        Map<String, String> errorResponse = new HashMap<String, String>();
        errorResponse.put("message", exception.toString());
        return errorResponse;
    }

}


//PetStoreData

package pet.store.controller.model;

import java.util.HashSet;
import java.util.Set;

import lombok.Data;
import lombok.NoArgsConstructor;
import pet.store.entity.Customer;
import pet.store.entity.Employee;
import pet.store.entity.PetStore;

@Data
@NoArgsConstructor
public class PetStoreData {
    //creating the variables for this class
    private Long petStoreId;
    private String petStoreName;
    private String petStoreAddress;
    private String petStoreCity;
    private String petStoreState;
    private Long petStoreZip;
    private String petStorePhone;
    private Set<PetStoreCustomer> customers = new HashSet<PetStoreCustomer>();
    private Set<PetStoreEmployee> employees = new HashSet<PetStoreEmployee>();

    public PetStoreData(PetStore petStore) {
        //filling the variables with data
        petStoreId = petStore.getPetStoreId();
        petStoreName = petStore.getPetStoreName();
        petStoreAddress = petStore.getPetStoreAddress();
        petStoreCity = petStore.getPetStoreCity();
        petStoreState = petStore.getPetStoreState();
        petStoreZip = petStore.getPetStoreZip();
        petStorePhone = petStore.getPetStorePhone();

        for (Customer customer : petStore.getCustomers()) {
            customers.add(new PetStoreCustomer(customer));

            for (Employee employee : petStore.getEmployees()) {
                employees.add(new PetStoreEmployee(employee));

            }
        }
    }

    @Data
    @NoArgsConstructor
    //Associating the petstorecustomers with petstore
    public static class PetStoreCustomer {
        private Long customerId;
```

```
        private String customerFirstName;
        private String customerLastName;
        private String customerEmail;

        public PetStoreCustomer(Customer customer) {
            customerId = customer.getCustomerId();
            customerFirstName = customer.getCustomerFirstName();
            customerLastName = customer.getCustomerLastName();
            customerEmail = customer.getCustomerEmail();
        }
    }

    @Data
    @NoArgsConstructor
    //Associating the petstoreemployees with petstore
    public static class PetStoreEmployee {
        private Long employeeId;
        private String employeeFirstName;
        private String employeeLastName;
        private String employeePhone;
        private String employeeJobTitle;

        public PetStoreEmployee(Employee employee) {
            employeeId = employee.getEmployeeId();
            employeeFirstName = employee.getEmployeeFirstName();
            employeeLastName = employee.getEmployeeLastName();
            employeePhone = employee.getEmployeePhone();
            employeeJobTitle = employee.getEmployeeJobTitle();
        }
    }
}


//CustomerDao
package pet.store.dao;

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

import pet.store.entity.Customer;

public interface CustomerDao extends JpaRepository<Customer, Long> {

}


//EmployeeDao

package pet.store.dao;

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

import pet.store.entity.Employee;

public interface EmployeeDao extends JpaRepository<Employee, Long> {

}


//PetStoreDao

package pet.store.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import pet.store.entity.PetStore;

public interface PetStoreDao extends JpaRepository<PetStore, Long> {

}


//Customer

package pet.store.entity;

import java.util.HashSet;
import java.util.Set;
```

```java
import com.fasterxml.jackson.annotation.JsonBackReference;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Entity
@Data
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long customerId;

    private String customerFirstName;
    private String customerLastName;
    private String customerEmail;

    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @ManyToMany(mappedBy = "customers", cascade = CascadeType.PERSIST)
    private Set<PetStore> petStores = new HashSet<PetStore>();

    @JsonBackReference
    public Set<PetStore> getPetStores() {
        return petStores;
    }

}


//Employee

package pet.store.entity;

import com.fasterxml.jackson.annotation.JsonBackReference;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Entity
@Data
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;

    private String employeeFirstName;
    private String employeeLastName;
    private String employeePhone;
    private String employeeJobTitle;

    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "pet_store_id", nullable = false)
    private PetStore petStore;

    @JsonBackReference
    public PetStore getPetStore() {
        return petStore;
    }
}
```

```
//PetStore

package pet.store.entity;

import java.util.HashSet;
import java.util.Set;

import com.fasterxml.jackson.annotation.JsonBackReference;
import com.fasterxml.jackson.annotation.JsonManagedReference;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToMany;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Entity
@Data
public class PetStore {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long petStoreId;
    private String petStoreName;
    private String petStoreAddress;
    private String petStoreCity;
    private String petStoreState;
    private Long petStoreZip;
    private String petStorePhone;

    @EqualsAndHashCode.Exclude
    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name = "pet_store_customer", joinColumns = @JoinColumn(name = "pet_store_id"), inverseJoinColumns =
    private Set<Customer> customers = new HashSet<Customer>();


    @EqualsAndHashCode.Exclude
    //pet_store or petStore
    @OneToMany(mappedBy = "petStore", cascade = CascadeType.ALL, orphanRemoval = true)
    private Set<Employee> employees = new HashSet<Employee>();

    @JsonManagedReference
    public Set<Employee> getEmployees() {
        return this.employees;
    }

    @JsonManagedReference
    public Set<Customer> getCustomers() {
        return customers;
    }

    }


//PetStoreService

    package pet.store.service;

import java.util.LinkedList;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Objects;

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

import pet.store.controller.model.PetStoreData;
import pet.store.controller.model.PetStoreData.PetStoreCustomer;
import pet.store.controller.model.PetStoreData.PetStoreEmployee;
```

```
import pet.store.dao.CustomerDao;
import pet.store.dao.EmployeeDao;
import pet.store.dao.PetStoreDao;
import pet.store.entity.Customer;
import pet.store.entity.Employee;
import pet.store.entity.PetStore;

@Service
public class PetStoreService {
    // adding autowired so that spring can inject the DAO object
    @Autowired
    private PetStoreDao petStoreDao;
    @Autowired
    private EmployeeDao employeeDao;
    @Autowired
    private CustomerDao customerDao;

    @Transactional(readOnly = false)
    public PetStoreData savePetStore(PetStoreData petStoreData) {
        PetStore petStore = findOrCreatePetStore(petStoreData.getPetStoreId());

        copyPetStoreFields(petStore, petStoreData);
        return new PetStoreData(petStoreDao.save(petStore));
    }

    private void copyPetStoreFields(PetStore petStore, PetStoreData petStoreData) {
        petStore.setPetStoreId(petStoreData.getPetStoreId());
        petStore.setPetStoreName(petStoreData.getPetStoreName());
        petStore.setPetStoreAddress(petStoreData.getPetStoreAddress());
        petStore.setPetStoreCity(petStoreData.getPetStoreCity());
        petStore.setPetStoreState(petStoreData.getPetStoreState());
        petStore.setPetStoreZip(petStoreData.getPetStoreZip());
        petStore.setPetStorePhone(petStoreData.getPetStorePhone());

    }

    private PetStore findOrCreatePetStore(Long petStoreId) {
        PetStore petStore;
        if (Objects.isNull(petStoreId)) {
            petStore = new PetStore();
        } else {
            petStore = findPetStoreById(petStoreId);
        }
        return petStore;
    }

    public PetStore findPetStoreById(Long petStoreId) {
        return petStoreDao.findById(petStoreId).orElseThrow(() -> new NoSuchElementException("Pet Store with ID= "
    }

    @Transactional(readOnly = true)
    public List<PetStoreData> retrieveAllPetStores() {
        List<PetStore> petStores = petStoreDao.findAll();
        List<PetStoreData> returnedPetStoreList = new LinkedList<PetStoreData>();

        for (PetStore petStore : petStores) {
            PetStoreData petStoreData = new PetStoreData(petStore);
            petStoreData.getCustomers();
            petStoreData.getEmployees();

            returnedPetStoreList.add(petStoreData);
        }

        return returnedPetStoreList;
    }

    @Transactional(readOnly = false)
    public PetStoreEmployee saveEmployee(Long petStoreId, PetStoreEmployee petStoreEmployee) {

        Employee employee = findOrCreateEmployee(petStoreId, petStoreEmployee.getEmployeeId());
        PetStore petStore = findPetStoreById(petStoreId);

        copyEmployeeFields(employee, petStoreEmployee);
        employee.setPetStore(petStore);
        petStore.getEmployees().add(employee);
        return new PetStoreEmployee(employeeDao.save(employee));
```

```
    }

    private void copyEmployeeFields(Employee employee, PetStoreEmployee petStoreEmployee) {
        employee.setEmployeeId(petStoreEmployee.getEmployeeId());
        employee.setEmployeeFirstName(petStoreEmployee.getEmployeeFirstName());
        employee.setEmployeeLastName(petStoreEmployee.getEmployeeLastName());
        employee.setEmployeePhone(petStoreEmployee.getEmployeePhone());
        employee.setEmployeeJobTitle(petStoreEmployee.getEmployeeJobTitle());
    }

    private Employee findOrCreateEmployee(Long petStoreId, Long employeeId) {
        Employee employee;

        if (Objects.isNull(employeeId)) {
            employee = new Employee();
        } else {
            employee = findEmployeeById(petStoreId, employeeId);
        }
        return employee;
    }

    private Employee findEmployeeById(Long petStoreId, Long employeeId) {
        Employee employee = employeeDao.findById(employeeId)
                .orElseThrow(() -> new NoSuchElementException("No Employee found"));
        if (employee.getPetStore().getPetStoreId() != petStoreId) {
            throw new IllegalArgumentException("Employee with ID: " + employeeId + "does not work at this pet store
        }
        return employee;
    }

    @Transactional(readOnly= false)
    public PetStoreCustomer saveCustomer(Long petStoreId, PetStoreCustomer petStoreCustomer){
        PetStore petStore = findPetStoreById(petStoreId);
        Long customerId = petStoreCustomer.getCustomerId();
        Customer customer = findOrCreateCustomer(petStoreId, customerId);

        copyCustomerFields(customer, petStoreCustomer);
        customer.getPetStores().add(petStore);
        petStore.getCustomers().add(customer);

        return new PetStoreCustomer(customerDao.save(customer));
    }

    private void copyCustomerFields(Customer customer, PetStoreCustomer petStoreCustomer) {
        customer.setCustomerId(petStoreCustomer.getCustomerId());
        customer.setCustomerFirstName(petStoreCustomer.getCustomerFirstName());
        customer.setCustomerLastName(petStoreCustomer.getCustomerLastName());
        customer.setCustomerEmail(petStoreCustomer.getCustomerEmail());
        }

    private Customer findOrCreateCustomer (Long petStoreId, Long customerId) {
        if(Objects.isNull(customerId)) {
            return new Customer();
        }
        else {
            return findOrCreateCustomer(petStoreId, customerId);
        }
    }

    @Transactional(readOnly = false)
    public void deletePetStoreById(Long petStoreId) {
        PetStore petStore = findPetStoreById(petStoreId);
        petStoreDao.delete(petStore);
    }

}


//application.yaml


Spring:
  datasource:
    username: pet_store
    password: pet_store
    url: jdbc:mysql://localhost:3306/pet_store
```

```
  jpa:
    hibernate:
      #the "create" option will wipe out pre-existing data and recreate the tables
      #the "update" will keep existing data and update the tables with new information
      ddl-auto: update
    show-sql: true
    defer-datasource-initialization: true

  sql:
    init:
      mode: never
```

```
//REFERENCES
//YOUTUBE: https://youtu.be/bSbQG20ZGK0
//GITHUB: https://github.com/fmd5045/Week13-15-PetStore-Project
```