Student Name: Nusaibah Mekkaoui

Student ID: B2205.010015

COM337 Advanced Programming

Dr. Prof. Selçuk Şener

Art Gallery Spring Boot Application

• Overview

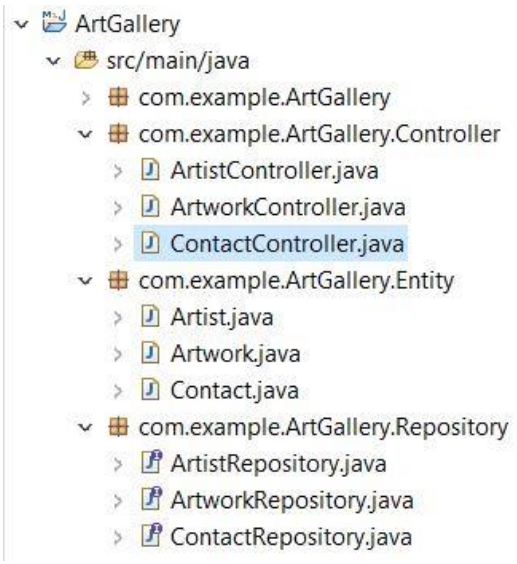
This document details the purpose and the functionalities of the program.

The program was designed as an Art Gallery Management System used by the gallery's employees to manage their featured artists, their artworks, and their contact information. The employees will be able to add, view, update and delete said information, as well as assign relevant information to one another. This Spring Boot Application is connected to a MySQL database where the information of the art gallery is stored.

1. About the Program

The program comprises 3 Entities, 3 Repositories, and 3 Controllers.

- Entities
  1. Artist Entity
  2. Artwork Entity
  3. Contact Entity
- Repositories
  1. Artist Repository
  2. Artwork Repository
  3. Contact Repository
- Controllers
  1. Artist Controller
  2. Artwork Controller
  3. Contact Controller



2. General Annotations

- @Entity: Defines a class an entity to create a table for it in the database
- @Table: Defines a name for each table, not always necessary but is useful when the name of the entity is the same as a reserved word
- @Id: indicates that the following field is the primary key of the database
- @GeneratedValue(strategy = GenerationType.IDENTITY): tells the database that the primary key should be generated automatically

3. Entities

1. Artist Entity

The data Fields included in the Artist Entity are:

1. Id
2. Name
3. Surname
4. Contact contact
5. List<Artwork> artworks

2. Navigation Properties

The navigation properties used in this entity are

a. One-To-One
- between the Artist and the Contact -- an artist may have only one email and phone number to be contacted by the gallery.

b. One-To-Many
- between the Artist and the Artwork—an artist may feature many artworks at this gallery.

3. Annotations

- @JsonManagedReference : Two of this annotation were used in this entity class to resolve an issue where the output would be displayed infinitely. The annotation works hand-in-hand with the @JsonBackReference annotation in both the Artwork Entity and the Contact Entity.

```
//imports

@Entity
@Table(name="Artist")
public class Artist {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

    @Column(name="FirstName")
    String name;

    @Column(name="LastName")
    String surname;

    @OneToOne(cascade = CascadeType.ALL)
    // @JsonIgnore
```

```
    @JsonManagedReference //used JsonManagedReference + JsonBackReference
to solve the infinite display issue
    Contact contact;

    @JsonManagedReference
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "artist")
    List<Artwork> artworks;

    public Artist(){

    }
//getters and setters
```

2. Artwork Entity

The data fields included in the Artwork Entity are:

1. Id
2. artwork
3. category
4. Artist artist

1. Navigation Properties

The navigation properties used in this entity are

a. Many-To-One

- between the Artwork and the Artist – many artworks may belong to one artist

2. Annotations

- @JsonBackReference : This annotation was used to resolve an issue where the output would be displayed infinitely. The annotation works hand-in-hand with the @JsonManagedReference annotation in the Artist Entity.

```
//imports

@Entity
@Table(name = "Artwork")
public class Artwork {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

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

```
    String artwork;

    @Column(name="Category")
    String category;

    @ManyToOne
    @JsonBackReference
    Artist artist;

    public Artwork(){

    }

//getters and setters

    public void removeArtist(Artist a) {
        this.removeArtist(a);
        a.artworks.remove(this);
    }
}
```

3. Contact Entity

The data fields included in the Contact Entity are:

1. Id
2. Email
3. Phone Number
4. Artist artist

1. Navigation Properties

The navigation properties used in this entity are

a. One-To-One

- Between the Contact and the Artist – One phone number and email (contact information) may be assigned to only one artist.

2. Annotations

1. @JsonBackReference : This annotation was used to resolve an issue where the output would be displayed infinitely. The annotation works hand-in-hand with the @JsonManagedReference annotation in the Artist Entity

```
//imports

@Entity
@Table(name="Contact Information")
public class Contact {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    Long id;

    @Column(name="Email")
    String email;

    @Column(name="Phone Number")
    Long phoneNum;

    @OneToOne(mappedBy="contact")
    @JsonBackReference //used JsonBackReference to solve the infinite
output issue
    Artist artist;

    public Contact() {

    }

//getters and setters
```

3. Repositories

This repositories inherit the methods from the JpaRepository which has predefined methods and has the parameters of the Entity and the data type of the primary key.

1. Artist Repository

- #### Custom Functions

  The custom function findByArtworksId(Long artId) will help find an artist's name, surname, email, phone number, and all of their artwork simply by entering one of their artwork's id.

```
public interface ArtistRepository extends JpaRepository<Artist, Long>{

    public List<Artist> findByArtworksId(Long artId);

}
```

2. Artwork Repository

- #### Custom Functions

  The custom function findByArtistId(Long artiId) will help find all of the artist's artwork simply by entering the artist's id.

```
public interface ArtworkRepository extends JpaRepository<Artwork, Long>{

    public List<Artwork> findByArtistId(Long artiId);

}
```

3. Contact Repository

- #### Custom Functions

  The custom function findByArtistId(Long artistId) will find an artist's email and phone number simply by entering the artist's id.

```
public interface ContactRepository extends JpaRepository<Contact, Long>{

    public Contact findByArtistId(Long artistId);

}
```

4. Controllers

1. Artist Controller

1. getArtistsByArtworksId()

This function returns the artwork ID and its corresponding artist ID, artist's name, surname based on that artwork's ID, all as a string. A custom function created in the artist repository and a for loop are used to get the information which is then displayed it on the browser.

2. assignArtistToArtwork()

As both the artist and the artwork are added to separate tables, we must assign one to the other to relate their ID's (foreign keys) to each other. This function assigns an artist to a specific artwork when the artwork ID and the artist ID are provided to the URL. The artist's ID will then be shown in the Artworks Table in the database. The function will then return a string indicating that the artist has been assigned successfully.

```
//imports

@RestController
```

```
@RequestMapping("artists")
public class ArtistController {

    @Autowired
    ArtistRepository artistRep;

    @Autowired
    ArtworkRepository artworkRep;

    @Autowired
    ContactRepository contRep;

    //the same function which returns the result in json format is down
below
    @GetMapping
    public String getArtists() {
        String result = "";

        for (Artist a : artistRep.findAll()) {
            result += "Artist ID: " + a.getId() + "<br>" + a.getName() +
" " + a.getSurname() + "<br>"
                    + "Contact Information: " + a.getContact().getEmail()
+ " - " + a.getContact().getPhoneNum()
                    + "<br>" + "Artworks:<br>";

            for (Artwork ar : a.getArtworks()) {
                result += ar.getArtwork() + " - " + ar.getCategory() +
"<br>";
            }
            result += "<br><br>";
        }
        return result += "<br>";
    }

    //the same function which returns the result in json format is down
below
    @GetMapping("{id}")
    public StringBuilder getArtist(@PathVariable Long id) {
        Artist a = artistRep.findById(id).get();

        StringBuilder result = new StringBuilder("Artist ID:
"+a.getId()+"<br>"+ a.getName() + " " + a.getSurname() + "<br>" +
                "<br>Contact Information:<br>" +
a.getContact().getEmail() + " - "
                + a.getContact().getPhoneNum() +"<br><br>Artworks:<br>");

        artistRep.findById(id).get().getArtworks()
                .forEach(ar -> result
```

```
                        .append(ar.getArtwork() + " " + ar.getCategory()
+ "<br>"));

        return result;
    }

    //the same function which returns the result in json format is down
below
    @GetMapping("artworks/{artId}")
    public String getArtistByArtworkId(@PathVariable Long artId) {
        String result = "";
        Artwork artwork = artworkRep.findById(artId).get();
        for(Artist a : artistRep.findByArtworksId(artId)) {
            result += "<br>Artist ID: "+ a.getId()+"<br>"+a.getName()+"
"+a.getSurname()+"<br>";
        }
        return "Artwork ID: "+artwork.getId()+"<br>"+result;
    }

    @PostMapping("save")
    public String saveArtist(@RequestBody Artist arti) {
        artistRep.save(arti);
        return "Artist Saved Successfully!";
    }

    @GetMapping("assignArtist/{artworkId}/{artiId}")
    public String assignArtistToArtwork(@PathVariable Long artworkId,
@PathVariable Long artiId) {
        Artwork artwork = artworkRep.findById(artworkId).get();
        artwork.setArtist(artistRep.findById(artiId).get());
        artworkRep.save(artwork);
        return "Artist Assigned To Artwork Successfully!";
    }

    @PutMapping("update/{artistId}")
    public String updateArtistInfo(@RequestBody Artist artist,
@PathVariable Long artistId) {
        Artist arti = artistRep.findById(artistId).get();
        arti.setName(artist.getName());
        arti.setSurname(artist.getSurname());
        artistRep.save(arti);
        return "Artist Information Updated Successfully";
    }

    @DeleteMapping("remove/{id}")
    public String deleteArtist(@PathVariable Long id) {
        artistRep.deleteById(id);
        return "Artist Deleted Successfully!";
```

```
    }

}
```

2. Artwork Controller

1. getArtworkByArtistId()

This function returns the artist ID and the artist's corresponding artwork ID, artwork and category based on that artist's ID, all as a string. A custom function created in the artist repository and a for loop are used to get the information and display it on the browser.

```
//imports

@RestController
@RequestMapping("artworks")
public class ArtworkController {

    @Autowired
    ArtworkRepository artworkRep;

    @Autowired
    ArtistRepository artistRep;

    @GetMapping
    public String getArtworks() {
        String result ="";

        for(Artwork ar : artworkRep.findAll()) {
            result += ar.getArtwork()+" - "+ar.getCategory()+ ". Artist
ID:" + ar.getArtist().getId() +"<br>";
        }
        return "Artworks:<br>"+ result;
    }

    @GetMapping("{id}")
    public String getArtwork(@PathVariable Long id) {
        Artwork ar = artworkRep.findById(id).get();
        return "Artwork ID:
"+ar.getId()+"<br>"+ar.getArtwork()+"  -  "+ar.getCategory()+"<br>";
    }

    @GetMapping("artist/{artistId}")
    public String getArtworkByArtistId(@PathVariable Long artistId) {
        String result = "";
        Artist artist = artistRep.findById(artistId).get();
```

```
        for(Artwork ar : artworkRep.findByArtistId(artistId)) {
            result += ar.getArtwork()+"  -  "+ar.getCategory()+"<br>";
        }
        return "Artist ID:
"+artist.getId()+"<br>"+"<br>Artworks:<br>"+result;
    }

    @PostMapping("save")
    public String saveArtwork(@RequestBody Artwork artworkId){
        artworkRep.save(artworkId);
        return "Artwork Saved Successfully!";
    }

    @PutMapping("update/{artworkId}")
    public String updateArtworkInfo(@RequestBody Artwork artwork,
@PathVariable Long artworkId) {
        Artwork art = artworkRep.findById(artworkId).get();
        art.setArtwork(artwork.getArtwork());
        art.setCategory(artwork.getCategory());
        artworkRep.save(art);
        return "Artwork Information Updated Successfully";
    }

     @DeleteMapping("remove/{artId}")
      public String removeArtwork(@PathVariable Long artId) {
         artworkRep.deleteById(artId);
         return "Artwork Deleted Succesfully";
      }


}
```

3. Contact Controller
    1. getContactsByArtistId()

       returns the artist ID, contact ID, email and address of a specific artist as a string.
    2. assignContact()

       As both the artist and the contact are added to separate tables, we must assign one to the other to relate their ID's (foreign keys) to each other. This function assigns a contact to a specific artist when the artist ID and the contact ID are provided to the URL. The contact ID will then be shown in the Artist Table in the database. The function will then return a string indicating that the contact has been assigned successfully.

3. deleteContact()

Deletes a certain email and phone number from the table and sort of nullifies the one-to-one relation between the artist and the contact so as to not create an error. Once the process is complete, the function will return a string that indicates that the contact information has been deleted successfully.

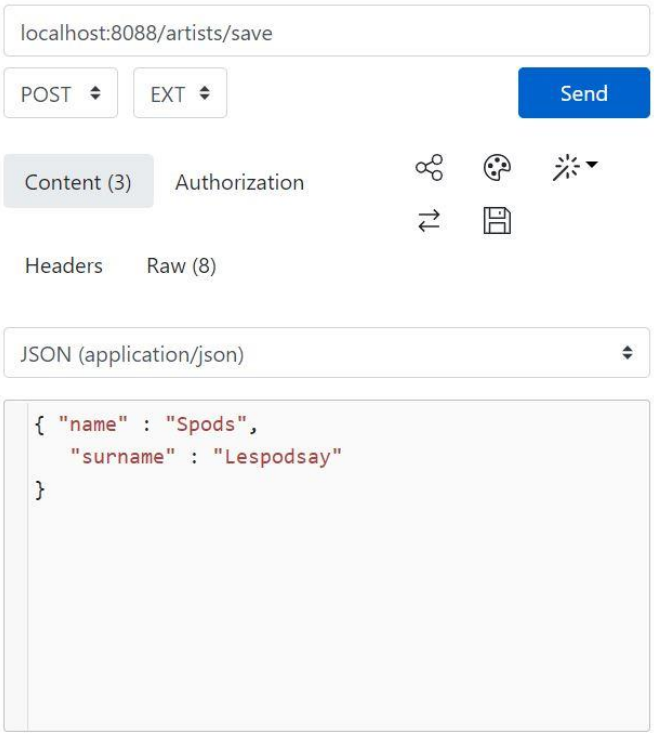
```
//imports

@RestController
@RequestMapping("contacts")
public class ContactController {

    @Autowired
    ContactRepository contRep;

    @Autowired
    ArtistRepository artistRep;

    @GetMapping
    public String getContacts() {
        String result = "";

        for(Contact c : contRep.findAll()) {
            result += "Contact ID: "+ c.getId()+"<br>"+c.getEmail()+" ,
"+c.getPhoneNum()+" , Artist ID: "+c.getArtist().getId()+"<br><br>";
        }
        return result;
    }

    @GetMapping("{id}")
    public String getContact(@PathVariable Long id) {
        Contact c = contRep.findById(id).get();
        return "Contact ID:
"+c.getId()+"<br>"+c.getEmail()+"  -  "+c.getPhoneNum()+"<br>";

    }

    @GetMapping("artist/{artistId}")
    public String getContactsByArtistId(@PathVariable Long artistId) {
        Contact c = contRep.findByArtistId(artistId);
        Artist a = artistRep.findById(artistId).get();
        return "Artist ID: "+a.getId()+"<br><br>"+"Contact ID:
"+c.getId()+"<br>"+c.getEmail()+"  -  "+c.getPhoneNum()+"<br>";
    }

    @PostMapping("save")
    public String saveContact(@RequestBody Contact cont){
```

```
        contRep.save(cont);
        return "Contact Information Saved Successfully!";
    }

    @GetMapping("assignContact/{artiId}/{contactId}")
    public String assignContact(@PathVariable Long artiId, @PathVariable
Long contactId) {
        Artist artist = artistRep.findById(artiId).get();
        artist.setContact(contRep.findById(contactId).get());
        artistRep.save(artist);
        return "Contact Assigned To Artist Successfully";
    }

    @PutMapping("update/{contactId}")
    public String updateContactInfo(@RequestBody Contact contact,
@PathVariable Long contactId) {
        Contact cont = contRep.findById(contactId).get();
        cont.setEmail(contact.getEmail());
        cont.setPhoneNum(contact.getPhoneNum());
        contRep.save(cont);
        return "Contact Information Saved Successfully!";
    }

    @DeleteMapping("remove/{id}")
    public String deleteContact(@PathVariable Long id){
        Contact contact = contRep.findById(id).orElse(null);
        if(!contRep.findById(id).equals(null)){
            contact.getArtist().setContact(null);
            contRep.save(contact);
            contRep.delete(contact);
        }
        return "Deleted Successfully!";

    }

}
```

5. Outputs

Should the user like to eneter new data directly through MySQL, they have to first enter the email and phone number (contact information) before they enter the artist information and, finally, the artwork information. However, using an API tester, this poses no problems whatsoever.

1. Save



id	first_name	last_name	contact_id
1	Lumi	Lumiere	1
2	Koko	Knockout	2
3	Spods	Lespodsay	3
NULL	NULL	NULL	NULL

Before Saving Artist

id	first_name	last_name	contact_id
1	Lumi	Lumiere	NULL
2	Koko	Knockout	NULL
3	Spods	Lespodsay	NULL
NULL	NULL	NULL	NULL

After Saving Artist

```
Artist Saved Successfully!
```

2. Assigning Contact to Artist

id	first_name	last_name	contact_id
1	Lumi	Lumiere	NULL
2	Koko	Knockout	NULL
3	Spods	Lespodsay	NULL
NULL	NULL	NULL	NULL

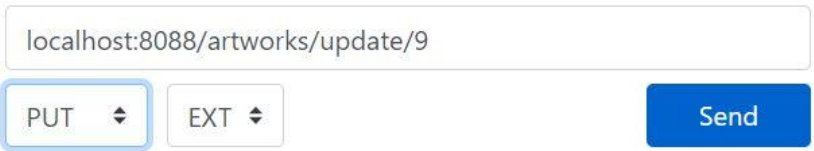
Before Assigning Contact to Artist

id	first_name	last_name	contact_id
1	Lumi	Lumiere	1
2	Koko	Knockout	2
3	Spods	Lespodsay	3
NULL	NULL	NULL	NULL

After Assigning Contact to Artist

```
Contact Assigned To Artist Successfully
```

3. Update



id	artwork	category	artist_id
1	Farmer Centaur	Digital Art	1
2	Kingdom Hearts Sora	Digital Art	2
3	Six Rikus	Digital Art	2
4	Namine	Digital Art	3
5	Venty Wenty	Digital Art	3
6	Sora Birthday Art	Digital Art	3
7	Riku My Beloved	Digital Art	3
8	Kairi My Beloved	Digital Art	3
9	Dreameater Rwiku	Digital Art	3

Before updating artwork category

id	artwork	category	artist_id
1	Farmer Centaur	Digital Art	1
2	Kingdom Hearts Sora	Digital Art	2
3	Six Rikus	Digital Art	2
4	Namine	Digital Art	3
5	Venty Wenty	Digital Art	3
6	Sora Birthday Art	Digital Art	3
7	Riku My Beloved	Merch	3
8	Kairi My Beloved	Merch	3
9	Dreameater Rwiku	Merch	3

After updating artwork category

```
Artwork Information Updated Successfully
```

4. Delete

Deleting an artist will delete all of their contact information and artworks

Before deleting artist

id	first_name	last_name	contact_id
1	Lumi	Lumiere	1
2	Koko	Knockout	2
3	Spods	Lespodsay	3
4			

id	email	phone number
1	lumorie@gmail.com	5522415984
2	knockout@hotm...	5553698101
3	lespodsay@gmai...	5527553274

id	artwork	category	artist_id
1	Farmer Centaur	Digital Art	1
2	Kingdom Hearts Sora	Digital Art	2
3	Six Rikus	Digital Art	2
4	Namine	Digital Art	3
5	Venty Wenty	Digital Art	3
6	Sora Birthday Art	Digital Art	3
7	Riku My Beloved	Merch	3
8	Kairi My Beloved	Merch	3
9	Dreameater Rwiku	Merch	3
NULL	NULL	NULL	NULL

After deleting artist

id	first_name	last_name	contact_id
1	Lumi	Lumiere	1
2	Koko	Knockout	2
NULL	NULL	NULL	NULL

id	email	phone number
1	lumorie@gmail.com	5522415984
2	knockout@hotm...	5553698101
NULL	NULL	NULL

id	artwork	category	artist_id
1	Farmer Centaur	Digital Art	1
2	Kingdom Hearts Sora	Digital Art	2
3	Six Rikus	Digital Art	2
NULL	NULL	NULL	NULL

```
Artist Deleted Successfully!
```