

phoneBook SpringBoot project.



entities -

1) User.java.

@Entity → This class will be mapped to a database table (ORM Hibernate/JPA).

@Table (name = USER) → Table name will be USER in the database.

@Id → marks this primary key column.

@GeneratedValue (strategy = GenerationType.AUTO)
→ Auto-generate IDs.

id → unique identifier for each user.

@NotBlank → validation if field can't be empty or null.

@Size (min = 2, max = 15) → Name must be bet'n 3 & 15 character.

name → user's display name.

@Column (unique = true) — ensure each email unique in DB

@NotBlank (message = "email must be required")

private String email;



login entity

@Column (length = 255) ← set column size DB

private String password;

private String role;
private boolean enable;

role → defines user's role e.g - ROLE_USER or
ROLE_ADMIN

enable → Account status (active / inactive)

private String name;

@Column (length=500)

private String about;

profileImage →

about → description

* Relationship with Contact

@OneToMany (cascade = CascadeType.ALL, fetch = FetchType.
LAZY, mappedBy = "user")

private List <Contact> contacts = new ArrayList <>();

@OneToOne → one user have multiple contact
cascade = CascadeType.ALL → Any operation
(save, delete) on user will also apply
to its contacts.

Fetch = FetchType.LAZY → Contacts are Loaded
only when Accessed (improve performance)
mappedBy = "user" → Refers to the User
entity.
(Bidirectional mapping)

- * toString() method

- Fetch Type LAZY \rightarrow To Avoid Loading All contacts every times use fetch (performance optimization)

password length 255?

- To support encrypted passwords like bcrypt, which can be longer.

Contact.java

```
@Table(name="contact") → tablename contact
private int eid → primary key
contact detail →
    name, second name
    work
    email
    phone
    image
    description
```

Relationship with User

@ManyToOne

@JoinColumn(name="user_id")
private User user;

@ManyToOne → many contacts belong to one user

@JoinColumn(name="user_id") →
Foreign key column user_id in
contact table links each contact to its user

This is inverse side of relationship defines
in User.java @OneToMany(mappedBy="user")

one user → many contact
many contacts → one user

getter, setter & toString()

provide methods to access / update field.

toString() → returns String representation of contacts (useful for debugging).

questions -

what happens if user is deleted?

→ Because of cascade in User.java, all associated contacts will be deleted.

why description has length = 700?

→ To allow longer text field on storing contact notes or description.

why did you use @ManyToOne in contact and @OneToMany in user?

→ Each contact belongs to single user, but a user can have many contacts.

- That's why I used @ManyToOne on contact & @OneToMany (mappedBy = "user") on user.

- This created a proper - child parent - child relationship in the database, using Foreign Key (user-id) in contacts table.

what happen if you delete user? Do contact get also delete?

→ I set Cascade = Cascadetype, ALL and orphanRemoval = true. deleting a user will also delete contacts.

- I not set contact DB orphan, which may issue foreign key issue. So I used orphanRemoval to keep the database clean.

Why did you use FetchType.LAZY instead of EAGER?

→ if I fetch a user who has 10,000 contacts, with EAGER, Hibernate will load all contact immediately.

that Big Performance hit :-

with LAZY, contacts are fetched only when I call User.getContacts(). It give me more control & Avoid unnecessary queries.

What problems occur with bidirectional mapping (user ↔ contact)?

→ The biggest issue is infinite recursion when converting to JSON or using toString()

for example →

~~user ↔ contact~~

user → contact → user → contact

I fix this by

@JsonIgnore or @JsonBackReference
on child side to avoid user field in
to string().

what strategy did you use for ID generation?
why?

→ I used @GeneratedValue (strategy = GenerationType.AUTO)
for mysql, it usually behave like IDENTITY
(Auto-increment).

- I choose AUTO so Hibernate decide based on
DB dialect, making portable.

How do you ensure email or phone is unique
in user or contact?

→ I added @Column (unique = true) and also
validated at the service layer.

If two user try inserting same email at the
same time, the DB constraint prevents it and
Hibernate throws a ConstraintViolation
Exception.

How do you prevent LazyInitializationException

→ LazyInitializationException happens when I try to access lazy collection after the session closed.

Fixes →

- 1) use @Transactional so session stay open.
- 2) or use JOIN FETCH in JPQL query.
- 3) or fetch into DTO instead entity.

Why should we override equals() & hashCode in JPA entities?

→ Because entities are stored in sets/maps in hibernate. If I don't override them properly, duplicate records or wrong lookups can occurs.

Use business key (like email) instead of DB id (since it is null).

How do you handle password storage in

→ I never store plain-text passwords. I use BCrypt password encoder in spring security.

during login spring security automatically matches raw password with hash password.



If you send a user as JSON, what happens to contacts?

→ without handling, it can infinite loop because bidirectional mapping.

I solved by

- using `@JsonManagedReference` on user contact

And `@JsonBackReference` on contact user.

so JSON serialization works without recursion.

what DB schema is generated for these two entity?

Users → id, name, email, password

Contacts → id, name, email, phone, user id.

* ContactRepository.java

This is use repository layer for Contact entity.

- 1) JPARepository <Contact, Integer>
periode & CRUD + pagination + sorting
- 2) Custom query with @Query → retrieve all contacts belonging to specific user (using User.id foreign key).
- 3) You can use JPQL Not SQL → It queries Contact entity, not table directly
- 4) It return List<Contact> → meaning multiple contacts per user

@Query

why do we use c.user.id instead of Joining manually with user table?

- Because JPQL work with entities and relationship, since Contact is mapped with user (@ManyToOne), we directly Access c.userid. Spring/JPA will automatically translate it into SQL with proper JOIN.

what is the difference between query and spring data JPA derived query method?

- - @Query :- lets us write custom JPQL / SQL queries.
- derived queries → spring data builds query based on method name (e.g. findByEmail (string email))

ex - User findByEmail (string email);

what happens if no user found for given email?

- The method returns null. To avoid NullPointerException, you can wrap the result in optional <User> instead.

optional <User> findByEmail (string email);

Can we use native SQL queries here?

Yes

Query @value = "select * from user where email = :email"
User user = userRepository.getUserByName (params ("email")
String email);

What is difference between JPQL & SQL

JPQL work with entities

SQL work with table & columns

username → email.



DAO.

UserRepository - Java

1) JpaRepository <User, Integer> →

Gives you all CRUD method (save, findById, findAll, delete, etc).

2) @Query → define custom JPQL query (using entity name user, not the table name)

3) getUserByUsername → custom finder method to retrieve a User by their email.

4) @Param("email") → binds method parameter with query parameter

Questions

why do we extend JpaRepository <User, Integer> instead of writing custom DAO classes?



JPA Repository provide built-in-CRUD, pagination & sorting methods reducing Boilerplate code DAO.

— Spring Data JPA generate implementation automatically

what happen if the user has no contacts?

→ It will return empty list, not null. This prevent `NullPointerException`. You can check safely `isEmpty()`.

How would you make the query paginated?

→ Instead returning `List<Contact>`, we can return a `Page<Contact>`.

This allow Pagination & Sorting automatically.

Can we use specific Spring Data derived query instead of `@Query`.

→ Yes. relationship exist betⁿ Contact & user.

`List<Contact> findByUserId (int userId)`

How to delete contact?

- I implemented delete functionality using Spring Boot JPA and Thymleaf. When user click the delete button request goes to Controller through URL like /user/delete/{id}. In the controller, I use repository's deleteById(id) method to remove the contact from database. Before deletion check login or not for Authentication. After delete redirect contact list page.

step by step

In this URL /user/delete/{id} I create ContactRepository interface object.

```
public String delete(@PathVariable("id") Integer id)
```

```
Optional<Contact> contact = ContactRepository.findById(id);
```

```
Contact contact =
```

```
contact.setUser(null); ← user unlink  
                        (user-id) foreign
```

```
this.ContactRepository.delete(contact);
```

method deleteId

}



@PathVariable -

to capture dynamic value from URL
 & pass them into controller method.

* how to update contact

inside show All contact user page

I Add Form tag

why use form tag

because method is post

After submit click button

and redirect the URL

th:ref="@{/User/UpdateContact/{id}}"

In this URL controller set handler
 method provide id using path variable

and use Model model In this
 spring MVC to send data from
 controller to the view (thymleaf.jsp)

for ex - Model is like holds key-value pair
 Controller Add object.

- model.addAttribute("key", object);

- view(Thymleaf.jsp)

key or th:field="key"

And return to update page.

update page pre-filled its detail in an update. After edit & submit.

in update page (Thymleaf page)

```
{ for m action="#" , th:method = "post" th:action =
"@{/user/process-update}" th:object = {contact}
enctype = "multipart/form-data" }
```

It is used for uploading files, because it allows both text and binary data to be sent to the server.

```
<input th:field = "${id}" type = "hidden" />
```

```
th:field = "${name}" Access value in object
```

// After goes inside controller process update

```
@PostMapping ("/process-update")
public String update (@ModelAttribute contact contact,
                      @RequestParam ("id") Integer id) {
```

```
String name = p.getName ();
User user = us.getUserByName (name);
```

```
contact contact.setcontact (user);
this.contactRepository.save (contact);
```

```
return "normal/show-contact";
```


principled is represent currently login user
we can in the controller fetch
username (ID/email) & Authenticate
user can load data security.

How to show contact to user?

```
public String showContact(Model m, Principal p) {  
    m.addAttribute("title", "Smart Contact");  
    String unam = p.getName();
```

```
User user = userRepository.getUserBy(unam);
```

```
List<Contact> contacts = this.contactRepository.  
    findContactByUser(user.getId());
```

```
m.addAttribute("contacts", contacts);
```

```
return "normalContact";
```


How to validate a form or login form?

→ implement validation using JSR-380 annotation like `@NotBlank`, `@Email`, `@Size` model class.

→ In Controller I used `@Valid` with `BindingResult` to check errors.

[If validation fail return error on message same page

How to upload image user?

→ 1. Add input form user.

```
<form method="post" th:action="@{/userupload}"  
    enctype="multipart/form-data">
```

```
</form>
```

2. request sent from multipart/form-data

3. controller method - get file via multipartfile , save it to /static/img/

4. Database store file ^{name} only not ~~data~~ actual file.

5) Thymleaf display image using `` tag.

Config



CustomUserDetail.java
→ In spring security, the UserDetails interface used to represent Authentication user.

I created CustomUserDetails to adapt UserDetails. It provides spring security with username (email), password, authorities (roles).

The method like isEnabled & isAccountNonLocked let control account status.

This way spring security can integrate directly with database user model.

```
method - getPassword()
          getUsername() // email set
          get boolean isAccountNonLocked() // true
          boolean isCredentialsExpired() // true - password not expire
          boolean isEnabled() // true - account active
```


UserDetailsService Impl Java

- spring security is the core interface that tells spring security How load a user from database using username/email.
- when someone tries to login spring security automatically calls your loadUserByUsername() method.

for example →

~~user ↔ contact~~

user → contact → user → contact

I fix this by

@JsonIgnore or @JsonBackReference
on child side, to avoid user field in
toString().

what strategy did you use for ID generation?
why?

→ I used @GeneratedValue (strategy = GenerationType.AUTO)
for MySQL, it usually behave like IDENTITY
(Auto-increment).

- I choose AUTO so Hibernate decide based on
DB dialect, making portable.

How do you ensure email or phone is unique
in user or contact?

→ I added @Column (unique = true) and also
validated at the service layer.

If two user try inserting same email At the
same time, the DB constraint prevents it and
Hibernate throws a ConstraintViolation
Exception.