9/6/25

Input Validation :-

Annotations for Validation

@NotNull → Ensures that the annotated field is not null.

@NotEmpty → Ensures that the annotated Field is not null and its size/length is greater than zero. (For Collections, arrays, and strings).

@NotBlank → Ensures that the annotated string is not null and its Trimmed length is greater than zero.

@Size → Validates that the annotated element's size falls within the specified range.

@Max → Ensures that the annotated element is a number, with a value no greater than the specified maximum.

@Email → Validates that the Annotated string is a Valid email address.

@Pattern → Validates that the Annotated string matches the Specified regular Expression.

@Positive → Ensures that the Annotated element is a Positive number (greater than zero).

@PositiveOrZero → Ensures that the Annotated element is a positive number ~~(greater than zero)~~ Or zero.

@Negative → Ensures that the Annotated element is a negative number;

@NegativeOrZero → Ensures that the Annotated element is a negative (or) zero number.

@Past → Ensures that the annotated date (or) Calender value is in the Past.

@PastOrPresent → Ensures that the Annotated date (or) Calendar Value is in the Past or Present.

@Future → Ensures that the Annotated date (or) Calendar value is in the future.

@FutureOrPresent → Ensures that the Annotated date or Calendar value is in the future (or) Present.

@Digits → Ensures that the Annotated number has up to a Specified number of integer and fraction digits.

@DecimalMin → Ensures that the Annotated element is a number with a value no less than the Specified minimum, allowing for decimal points.

@DecimalMax → Ensures that the Annotated element is a number with a value no greater than the Specified maximum, allowing for decimal points.

@AssertTrue → Ensures that the Annotated boolean field is "true".

@AssertFalse → Ensures that the Annotated boolean field is false.

@Valid → validates the Associated object recursively (applies bean Validation to nested objects.

Handling Validation Exceptions:-

1. MethodArgumentNotValidException is thrown by the @Valid Validation.
2. You can get a list of all the errors from the bindingResult of this exception.
3. You can use this list to return some useful error messages as the API response.

Exception Handling in Spring Boot MVC

Benefits of Exceptions Handling:-

1. Prevent application Crashes.
2. Provide user-friendly error response.
3. Facilitate debugging and Maintenance.
4. Ensure consistent error handling Across the Application.

Handling Exceptions:-

1. Use @ExceptionHandler to handle specific exceptions in Controller.

```
@ExceptionHandler (NoSuchElementException.class)
Public ResponseEntity<String> handleEmployeeNotFound (NoSuchElementException exception)
{
  return new ResponseEntity<> ("Employee not found", HttpStatus.NOT_FOUND);
}
```

→ it will only handle the Exception in the Employee Controller.

2. Use @RestControllerAdvice for global exception handling.
3. Return Appropriate HTTP status codes and Error messages.
4. Use Custom error response class to provide structed error details.

```
@RestController Adivce
Public class GlobalExceptionHandler {
@ExceptionHandler (ResourceNotFoundException.class)
Public ResponseEntity<ApiError> handleResourceNotFound (ResourceNotFoundException exception)
{
  ApiError. apiError = ApiError.builder().
                        .status (HttpStatus.NOT_FOUND)
                        .message (exception.getMessage())
                        .build();
  return new ResponseEntity<> (apiError, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

→ This Exceptions handles by the OverAll Application.

10/6/25

Transforming API Response

1. Extend your Class with @ResponseBodyAdivce<Object> to define the Custom return type.
2. Use @RestControllerAdivce for global API Response transformation.
3. Return appropriate HTTP status codes and error messages.
4. You Can also return the timestamp of the API response.

11/6/25

Java 8 Lambda And Stream API:-

Lambda expression is a short way to write a Method for an interface that has only one abstract method called a functional interface.

Functional Interface:-

1. it is in Java is an interface that has only one abstract Method.
2. it can have any number of default (or) static methods but only one abstract method.
3. Main Purpose of functional interfaces is to support lambda expressions.
4. Ex:- Runnable interface, which has a single method run(), is a functional interface.
5. @FunctionalInterface Can be used to explicity declare an interface as a functional interface.
   → it is not Mandatory, it helps the Compiler Catch errors if the interface does not Conform to the rules of a functional interface.

```
@FunctionalInterface
interface Walkable{

  int walk(int steps, boolean isEnabled);
}
```

```
walkable obj= (steps, isEnabled)->{
 System.out.println("walking fast"+ steps + "steps");
  return 2*steps;
};

walkable obj2= (steps, isEnabled)->2*steps;
```

12/6/25

Stream :-

1. it is a sequence of elements that supports various Operations to Process data.
2. it's not a data structure but a way to proces data from source like Collections, arrays, or I/O channels.
3. it work by Creating a pipeline of Operations.
4. You start with a source of data, then apply a series of ~~im~~ intermediate Operations (like filtering, mapping, or Sorting) To transform the data, and finally, a terminal Operation (like Collecting, Counting, or Printing) to Produce a result.
5. The Main Purpose is to enable functional-style Operations on Collections of the data.
6. They allow you to write Code that is more Concise, readable and easier to Parallelize.
7. They also Promote immutability, as they don't modify the Original data Source.

```
List<String> fruits = List.of("kiwi", "Banana", "Apple");
Stream<String> streams = fruits.stream();
// streams.forEach((fruit) -> {
        S.o.pln(fruit);
   }

Stream .sorted() . map(fruit -> fruit.length())
       . forEach (fruit -> S.o.Pln(fruit));
```

Homework:-

1. Make a list of all the annotations we have seen so far in the Spring Web Framework.

@Entity → Maks a class as a database table.
→ Used in JPA to tell Spring: "This class represents a table in DB".

@Service → Automatically makes it a Spring-managed bean.
→ Used to handle main logic (like Calculations, DB calls).

@Repository → Used to talk to the database
→ Spring Automatically handles exceptions here.

@Controller → Used in MVC applications to handle HTTP requests and return views (HTML Pages).
→ Used with Thymeleaf / JSP.

@RestController → Shortcut for @Controller + @ResponseBody
→ Used in REST APIs to return JSON data instead of views.

@Autowired → Automatically injects dependencies.
→ No need to manually create objects using new.

@RequestMapping → Maps URLs to methods.
→ Can be used with GET, POST, PUT, DELETE.

@PathVariable → Used to get data from URL Path.

@RequestParam → Used to get data from URL query Parameters.

@RequestBody → it return data as JSON, not HTML view
→ no needed in @RestController

@ResponseStatus → Sets a Custom HTTP status Code for the response.

@ExceptionHandler → Handles specific exceptions in your app.
→ Used inside a Controller or @RestControllerAdvice.

@RestControllerAdvice → Used to handle exceptions globally.
→ Like a global error handler.

2. Create the following REST endpoints for the Following Entity

Department	REST APIs:-
- id	GET : /departments
- title	POST : / departments
- isActive	PUT : /departments
- CreatedAt	DELETE : /departments
	GET : /departments/{id}

3. Write Exception Handling logic for the Department Entity.

4. Add the Appropriate Annotations For fields in the Department and Employee Entities.

5. Create Custom Annotation to handle a Prime number input.

6. Create custom Annotation to Check if the Password string field.
   a. contains One UpperCase, lower case, special character.
   b. minimum length is 10 characters.

Mistakes Made by My And Rectified by the AI:-

1. Always Type the URL manually in Postman To Avoids hidden characters.
2. Never Press enter in the URL bar to Prevents newline %0A from being added.
3. Use "Send" button only because it's safer.
4. Don't paste URLs from PDFs (or) chats because They often include hidden newlines.
5. Must Watch the IntelliJ Console Because it shows what Spring Boot is doing.

13/6/25

Git and GitHub:-

Git:-

→ Version Control System is a Tools that helps to track changes in Code.

→ Git is a Version Control System
↓
Tracks changes to Files, like Code, Over time

→ it is Popular, free, Open source, fast, Scalable.

1. it helps to Track the history
2. it helps to colloborate the Team.
3.

Github:-

→ it is a Website that allows developers to store and Manage their Code using Git.

1. Folder(Repo)

Version Control:-

→ it is a System that Tracks changes to files overtime, allowing developers to revert to previous versions, Collaborate efficiently, and manages different Code versions.

→ it Prevents data loss and ensures smooth teamwork in software development.

without git:-

→ For every Time we ~~wa~~ Use old and update. So, that we alsway want old Code For updation.

→ For Any Projects we want to follow the same flow.

with git:-

→ we don't need To use old code To Make Any updates init because it stored in a local.

→ So, that we can update Easy.

Repo:- Repo is a folder where git is initialized.

Git ?

→ it is a distributed Version Control system that Tracks changes in code, allowing multiple developers to Collaborate efficiently.

Use Cases of Git :-

* Software Development
* Project Management
* Tracking Versions.

GitHub ?

→ it is a cloud-based plateform that hosts Git repositories, enabling developers to Collaborate, manage Code and track changes efficiently.

Git V/s GitHub

Git works locally to track the changes in folders and Files. helps in Organizing the code with the help of Branches.

GitHub is A Cloud Plateform that hosts the local repo Online. So that Anyone with the repo link can Access the Project and Contribute changes.

Git Workflow

1. git add --all / git add .
   → This ~~Commamd~~ Command will add the files to the staging area.
2. git Comm~~it~~it -m 'message'
   → This Command will Commit the Changes locally.
3. git push remoteName BranchName
   → This Command will Push the changes from local repo to GitHub.
4. git Push remoteName BranchName.

14/6/25

4. git Pull remoteName BranchName
   ↓
   Latest changes in the branch.

Git Branch Commands:-

git branch → Lists all local branches in the Repository.

git branch <branch-name> → Creates a new branch with the Specified Name

git checkout <branch-name>/ git switch <branch-name>
→ Switches to the Specified branch.

git checkout -b <branch-name>/ git switch -c <branch-name>
→ Creates and switches to a new branch.

git merge <branch-name> → Merges the Specified branch into the Current branch.

git branch -d <branch-name> → Deletes a local branch (only if folly merged).

git branch -D <branch-name> → Force deletes a local branch.

git push origin --delete <branch-name> → Deletes a remote branch.

git branch -m <old-name> <new-name> → Renames a branch.

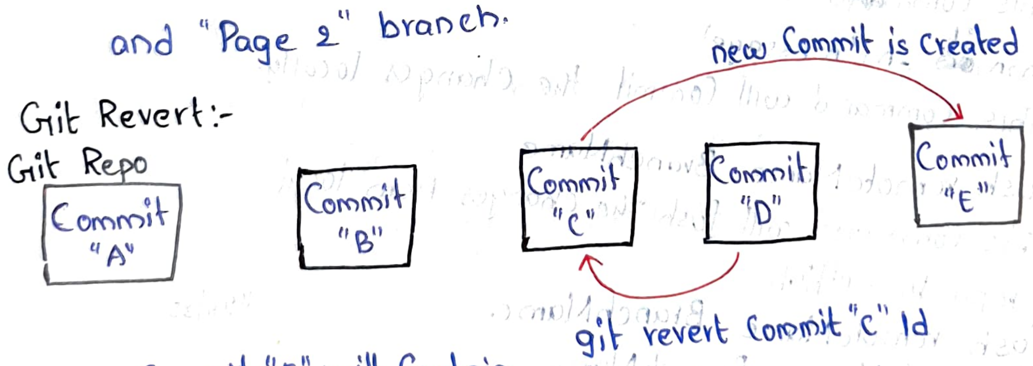
git branch -r → Lists remote branche.

Git Merge:-

Merging is nothing but just adding the code Present in One branch to another branch.

→ "Master" branch will now have the code from "Page 1" branch and "Page 2" branch.

Git Revert:-



now, Commit "E" will Contain the Code of Commit "C"