**Validation in Springboot:**

@Valid is the annotation we use in the spring validation (use this in the restcontroller)on the pojo’s and use the specific annotations on the fields to validate them accoridingly..

[Ex.@NotNull](mailto:Ex.@NotNull)(message=””)

@NotBlank(message=””)

@NotEmpty(message=””)

Each annotation carries a message attribute to display the message there in.

We use the Spring validation dependency in the pom.xml to use the all the related annotations on the field.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

**Pagination in the SpringBoot:**

Pagination is used in the springboot to fetch the limited data and not all at the once.

For that ,

You use the pageable I/F of spring data JPA.

You can pass this inside the Jparepository method of find all.

You can pass the page number and size for the page.

Also you can decide the sortby and ascending() or descending() for the specific user case.

**Advantages of Pagination**

1. **Performance Optimization**: Loading a large dataset at once can be inefficient and slow, consuming significant memory and processing power. Pagination limits the amount of data processed and sent at one time, improving response times and reducing server load.
2. **Improved User Experience**: Instead of waiting for all data to load, users receive a manageable subset quickly, leading to a smoother experience. Pagination is essential for interfaces where users typically browse through pages, like product listings or search results.
3. **Network Bandwidth Efficiency**: By fetching data in chunks, pagination reduces the amount of data sent over the network, especially beneficial for mobile or low-bandwidth environments.
4. **Scalability**: For databases with millions of records, pagination allows the application to scale better, as only a portion of data is loaded into memory and processed at a time.
5. **Resource Management**: By handling data in segments, pagination reduces the risk of overloading system resources, making applications more stable under high loads.
6. **Easier Data Processing**: When paginated, data can be processed in smaller, manageable portions. This approach simplifies operations like sorting, filtering, or additional processing without needing to handle the full dataset.

Spring Data JPA of Spring Boot provides *Pageable* interface to developers to use page-based pagination to fetch data from their data source. Page-based pagination uses page number and page size to fetch data based on size and page number that is given in request.

The *Pageable* interface in Spring Data JPA provides a way to handle pagination and sorting of database queries by allowing you to request a specific subset, or "page," of a dataset. You create a *Pageable*instance using *PageRequest*, where you specify the page number (starting at 0), page size, and optional sorting parameters. This *pageable*object can be passed to repository methods, like ***findAll(Pageable pageable)***, to retrieve a *Page* object that contains a subset of data as well as metadata, such as total pages, total elements, and navigation information (e.g., whether there are more pages). This approach improves performance by fetching only the needed data for each page, and it makes it easy to implement pagination and sorting in applications, especially useful in REST APIs to deliver data in manageable chunks.

I/F used => Pageable pageable(instance passed to the repo)

Take page and size from the user => default values must be provided.

Pagerequest.of(page, size);=> Concrete implementation of pageable.

Pagerequest.of(page,size, Sort.by(“property”).ascending()/.descending());

**Springboot actuator:** <!-- Maven -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>  
  
  
Spring Boot Actuator is a sub-project within the Spring Boot framework that provides production-ready features to monitor and manage Spring Boot applications. It exposes operational data about the running application through various endpoints, primarily over HTTP.  
  
**Monitoring and Metrics:**

Actuator provides endpoints to gather and expose various metrics about the application's health, performance, and resource usage. This includes information on memory usage, CPU load, thread count, HTTP request latency, database connection pool status, and disk space.

* **Health Checks:**

The /health endpoint offers a quick way to check the overall health of the application and its dependencies, such as database connections, disk space, and other integrated services.

* **Application Information:**

Endpoints like /info and /env expose arbitrary application information and environment properties, respectively, which can be useful for debugging and understanding the application's configuration.

* **Debugging and Diagnostics:**

Actuator offers endpoints like /beans to list all Spring beans, /configprops to view @ConfigurationProperties, and /threaddump to perform a thread dump, aiding in debugging and performance analysis.

* **Extensibility:**

While Spring Boot Actuator provides many built-in endpoints, it is also highly extensible, allowing developers to add custom endpoints and metrics specific to their application's needs.

To expose all the endpoints use this in application.properties::::  
  
 management.endpoints.web.exposure.include=\*

<http://localhost:8080/actuator> => to access the actuator endpoints.