Module 4 - Spring Boot Fundamentals

Search, Filters, Pagination, Sorting

Warm-up quiz

Warm-up

- 8 questions, multiple choice.
- Reply format: 1 / 2 / 3 / 4.
- Time: 8 minutes.
- Topics: Spring MVC annotations, request input binding, query parameters, validation trigger, global exception handling.

Question 1

Which annotation is the best fit to map a controller method to GET /students?

1) @PutMapping("/students")
2) @RequestMapping(method = RequestMethod.POST, value = "/students")
3) @GetMapping("/students")
4) @PostMapping("/students")

Reply format: 1 / 2 / 3 / 4

Answer 1

Which annotation is the best fit to map a controller method to GET /students?

1) @PutMapping("/students")
2) @RequestMapping(method = RequestMethod.POST, value = "/students")
3) @GetMapping("/students")
4) @PostMapping("/students")

Correct answer: 3
Why: @GetMapping maps HTTP GET requests to a path, which is the standard method for reading data.

Question 2

A request comes in as GET /students/18. Which annotation reads 18 into a method parameter?

1) @RequestBody
2) @ResponseBody
3) @RequestParam
4) @PathVariable

Reply format: 1 / 2 / 3 / 4

Answer 2

A request comes in as GET /students/18. Which annotation reads 18 into a method parameter?

1) @RequestBody
2) @ResponseBody
3) @RequestParam
4) @PathVariable

Correct answer: 4
Why: @PathVariable extracts values from the URL path. In /students/18, the id is in the path.

Question 3

Which request uses query parameters?

1) POST /students
2) GET /students?page=1&size=5
3) DELETE /students/18
4) GET /students/18

Reply format: 1 / 2 / 3 / 4

Answer 3

Which request uses query parameters?

1) POST /students
2) GET /students?page=1&size=5
3) DELETE /students/18
4) GET /students/18

Correct answer: 2
Why: Query parameters appear after ? as key-value pairs like page=1 and size=5.

Question 4

Which annotation reads JSON from the request body and converts it into a Java object?

1) @Component
2) @RequestBody
3) @RequestParam
4) @PathVariable

Reply format: 1 / 2 / 3 / 4

Answer 4

Which annotation reads JSON from the request body and converts it into a Java object?

1) @Component
2) @RequestBody
3) @RequestParam
4) @PathVariable

Correct answer: 2
Why: @RequestBody tells Spring to read the request body and deserialize JSON into the declared Java type.

Question 5

In a controller method, what does @Valid do when used with a request DTO?

1) It converts path variables into integers
2) It validates the DTO before the controller method continues
3) It automatically saves the DTO into the database
4) It logs the request body to the console

Reply format: 1 / 2 / 3 / 4

Answer 5

In a controller method, what does @Valid do when used with a request DTO?

1) It converts path variables into integers
2) It validates the DTO before the controller method continues
3) It automatically saves the DTO into the database
4) It logs the request body to the console

Correct answer: 2
Why: @Valid triggers validation rules. If rules fail, Spring stops the request before calling service logic.

Question 6

What is the main purpose of @RestController?

1) It creates a global exception handler
2) It makes controller methods return response bodies, usually JSON
3) It enables validation annotations like @NotBlank
4) It marks a class as a database repository

Reply format: 1 / 2 / 3 / 4

Answer 6

What is the main purpose of @RestController?

1) It creates a global exception handler
2) It makes controller methods return response bodies, usually JSON
3) It enables validation annotations like @NotBlank
4) It marks a class as a database repository

Correct answer: 2
Why: @RestController is used for APIs. It writes return values into the HTTP response body.

Question 7

What is @RestControllerAdvice used for?

1) It converts JSON into Java objects
2) It seeds test data when the application starts
3) It handles exceptions across all controllers in one place
4) It maps GET requests to a controller method

Reply format: 1 / 2 / 3 / 4

Answer 7

What is @RestControllerAdvice used for?

1) It converts JSON into Java objects
2) It seeds test data when the application starts
3) It handles exceptions across all controllers in one place
4) It maps GET requests to a controller method

Correct answer: 3
Why: @RestControllerAdvice applies globally and lets you convert exceptions into consistent HTTP responses.

Question 8

A client calls GET /students?name=riya&page=0&size=5. Which inputs come from query parameters?

1) None of them
2) Only page and size
3) name, page, and size
4) Only name

Reply format: 1 / 2 / 3 / 4

Answer 8

A client calls GET /students?name=riya&page=0&size=5. Which inputs come from query parameters?

1) None of them
2) Only page and size
3) name, page, and size
4) Only name

Correct answer: 3
Why: Everything after ? is the query string. Each key-value pair is a query parameter.

Search, filters, pagination

What does search mean in a list API?

- A **list API** returns a collection
    - Example: GET /students returns a list of Students.
- **Search** means narrowing that collection using optional query parameters.
- The REST pattern is to keep the same list endpoint and add search inputs as query parameters.
  - GET /students?name=riya
  - GET /students?email=gmail.com
- A search API must clearly define matching rules so results are predictable.

Matching rules for our API

- Name filter rule
  - match if student name contains the query value
  - case-insensitive, and leading or trailing spaces are ignored
- Email filter rule
  - match if student email contains the query value
  - case-insensitive, and leading or trailing spaces are ignored
- Active filter rule
  - match if student active equals the query value
  - active=true returns only active students, active=false returns only inactive students
- Missing filters are not applied.
  - no name means do not filter by name
  - no email means do not filter by email
  - no active means do not filter by active

Combining filters

- If the client sends multiple filters, the server applies all of them.
- This API uses AND logic.
  - the student must match name, email, and active when those filters are provided
- AND is a safe default because it prevents unexpectedly large results.
- Example
  - GET /students?name=riya&email=gmail.com&active=true
  - returns only students that match all provided filters

Quick check

Using the below dataset, what does *GET /students?name=riya&email=gmail.com* return?

1) Riya Menon, riya.menon@gmail.com, active=true
2) Priya Nair, priya.nair@yahoo.com, active=false
3) Rahul Das, rahul.das@gmail.com, active=true
4) Ananya Pillai, ananya.pillai@outlook.com, active=true

1) Student 1 only
2) Students 1 and 2
3) Students 1 and 3
4) All students

Answer: 1
Why: name=riya matches 1 and 2, email=gmail.com matches 1 and 3, AND logic keeps only student 1.

Filters: Optional Controls

- A **filter** is optional input that changes which items appear in a list response.
- Filters belong in query parameters because they modify the list result, not the endpoint identity.
- We'll update our API to support three filters:
  - *name* filters by student name
  - *email* filters by student email
  - *active* filters by student active status
- Examples
  - GET /students?name=an
  - GET /students?email=outlook
  - GET /students?active=true

Server pipeline for list requests

- The server must use one consistent order for list responses. The order we'll implement:
  - 1. fetch all students from the repository
  - 2. apply filters (name, email, active)
  - 3. apply sorting
  - 4. apply pagination
- Pagination must run after filtering and sorting, otherwise results change across pages.
- Preventing surprises with filters
  - Empty filter values behave like filter not provided.
    - name= behaves like no name filter
  - Case-insensitive matching prevents confusing results.
    - email=GMAIL.COM matches rahul.das@gmail.com
  - Trimming spaces prevents accidental mismatches.
    - name= riya behaves like name=riya

Quick check

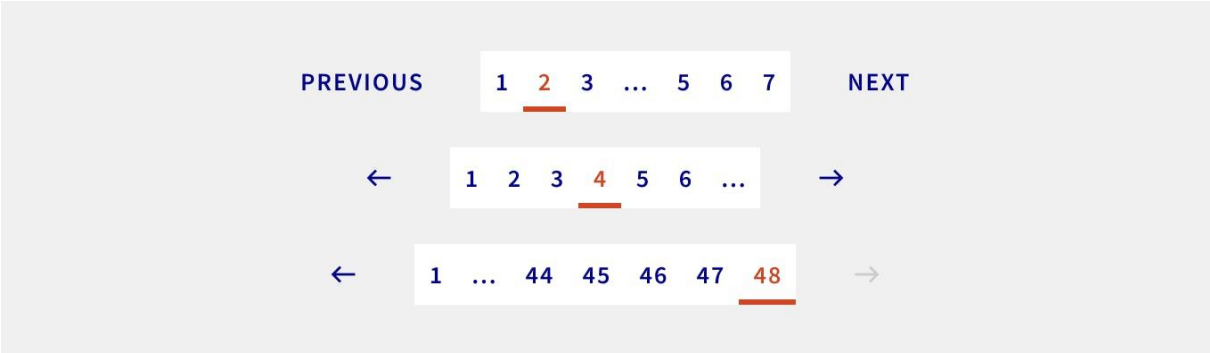
Which request applies two filters at the same time?

1) GET /students?name=riya&email=gmail.com
2) GET /students?name=riya&page=1
3) GET /students?page=1&size=5
4) GET /students?sortBy=name&sortDir=asc

Answer: 1
Why: name and email are filter parameters. The others are pagination or sorting inputs.

Why pagination exists in real APIs

- Without pagination, GET /students can return too many items.
- That causes slow responses and heavy network usage.
- Client screens can become slow or hard to render.
- Pagination returns results in pages so clients can fetch only what they need.



Pagination parameters

- **page** chooses which page of results to return.
  - 0-based, so page=0 is the first page
- **size** chooses how many items appear in one page.
- Example
  - GET /students?page=1&size=5
  - returns the second page with 5 items
- Note: the keys, page & size, are simply standard query parameters. These keys may not be used in all applications.

How page and size map to indexes

- Pagination is a slice of the final list after filtering and sorting.
- Start index is page * size.
- End index is start + size.
- Returned items are indexes start to end - 1.
- Example
  - page = 2, size = 5
  - start = 10, end = 15
  - returns indexes 10 to 14

Out-of-range pages and response shape

- If the client asks for a page beyond the total matched items, return an empty list and keep metadata consistent.
- A page response includes items, page, size, and total so client UI logic stays simple.

```
{
  "items": [],
  "page": 3,
  "size": 5,
  "total": 12
}
```

Quick check

If the filtered list has 20 items and the request is page=1&size=5, which indexes are returned?

1) 0 to 4
2) 5 to 9
3) 10 to 14
4) 15 to 19

Answer: 2
Why: start is 1 * 5 = 5, so the response returns indexes 5 to 9.

Sorting and full pipeline

Sorting in list APIs

- **Sorting** controls the order of items in a list response.
- Sorting does not change which students are included. It only changes the sequence.
- Sorting is applied after filtering and before pagination.
  - filtering decides what stays
  - sorting decides the order
  - pagination slices the ordered list

Sorting inputs for this API

- This API uses two query parameters for sorting.
  - sortBy chooses the field to sort on
  - sortDir chooses the direction
- Allowed values in this lecture
  - sortBy can be name or email
  - sortDir can be asc or desc
- Defaults if missing
  - sortBy=name
  - sortDir=asc
- Unsupported values are rejected as a client error.
  - return 400 Bad Request with allowed values in the message
- Examples
  - GET /students?sortBy=name&sortDir=asc
  - GET /students?sortBy=email&sortDir=desc

Quick check

Which request sorts students by email in descending order?

1) GET /students?sortBy=email&sortDir=desc
2) GET /students?sortBy=desc&sortDir=email
3) GET /students?sortBy=email&sortDir=asc
4) GET /students?email=desc

Answer: 1
Why: sortBy chooses the field and sortDir chooses the direction.

Contract for GET /students

- Endpoint
  - GET /students
- Optional filters
  - name
  - email
  - active
- Optional sorting
  - sortBy
  - sortDir
- Optional pagination
  - page
  - size
- All of these are query parameters because they modify how the list is returned.

Processing order with one full example

- Pipeline
  - start with all students
  - apply filters
  - apply sorting
  - apply pagination
- Meaning
  - filter to students whose name contains riya
  - keep only students with active=true
  - order by email ascending
  - return the first page with 3 students

```
GET /students?name=riya&active=true&sortBy=email&sortDir=asc&page=0&size=3 HTTP/1.1
Host: localhost:8080
```

Quick check

Which part of the query controls the page size?

1) page=0
2) size=3
3) sortDir=asc
4) name=riya

Answer: 2
Why: size controls how many items appear in a page.

Live Coding

Live coding objective

- Objective
  - extend GET /students to support filtering, sorting, and pagination using query parameters
- Why do we keep search on GET /students endpoint?
  - It is still the same resource collection
  - query parameters modify how the list is returned, not what the resource is
- By the end, the client can call
  - GET /students?name=riya&email=gmail.com&active=true
  - GET /students?sortBy=name&sortDir=asc
  - GET /students?page=1&size=5

Changes to be made during live coding

- Model
  - add active field to Student
- Startup seed data
  - seed enough students so pagination is meaningful
- Controller
  - read optional query parameters using @RequestParam
- Service
  - implement the pipeline and matching rules
  - apply defaults for sorting and pagination
  - reject unsupported sortBy and sortDir with 400 Bad Request

Response shape for the list endpoint

- Return both items and metadata so clients can build a UI.
- This API returns items, page, size, and total.

```
{
  "items": [
    { "id": 1, "name": "Riya Menon", "email": "riya.menon@gmail.com", "active": true }
  ],
  "page": 0,
  "size": 3,
  "total": 12
}
```

Coding Activity

Activity

- Follow the same contract used in the demo.
- Do not create a new endpoint like /students/search.
- Keep behavior predictable.
  - filters use AND logic
  - sorting runs before pagination
  - pagination runs after filtering and sorting

That’s a wrap!

Key takeaways

- Search is a filtered list endpoint with clear matching rules.
- Filters belong in query parameters because they modify list results.
- Pagination uses page and size to return predictable slices.
- Sorting uses sortBy and sortDir to control order, not membership.
- A predictable pipeline makes APIs easier to use and easier to debug.
  - filter
  - sort
  - paginate