

Project_Day_2

29 January 2026 00:50

On Second day of project we will see how two features can be built.

1st feature is Pagination : Here the scenario is list of users which is created is getting large and there is limit to show these users over UI, as you can't populate all the users over a same page.

The screenshot shows a web browser window with a purple header bar. On the left, there is a modal dialog titled "Add New User" with fields for Name, Email, and Password, and a "Add User" button. On the right, there is a table titled "Users List" with columns ID, Name, Email, and Action. The table contains 12 rows of data. A red box highlights the "Users List" table. A handwritten note on the right side of the screenshot reads: "As you can see loading 100k user can cause issues here. To overcome it Pagination will be used." An arrow points from the text to the "Users List" table.

ID	Name	Email	Action
1	Test	test@test.com	<button>View</button>
2	Test	Test@test.com	<button>View</button>
3	Test	3test@test.com	<button>View</button>
4	Test	3test@test.com	<button>View</button>
5	Test	4test@test.com	<button>View</button>
6	test	5test@test.com	<button>View</button>
7	Test	6test@test.com	<button>View</button>
11	Test	1234@abc.com	<button>View</button>
12	Test Test	111@abc.com	<button>View</button>

Implementation brief

1. Pageable interface will be used to get the size you require and you can set up page, size, sorting and sort by as per your requirement.

```
@GetMapping  
public ResponseEntity<Page<User>>  
getAllUsers(@RequestParam(defaultValue = "0") int page,  
            @RequestParam(defaultValue = "3") int size,  
            @RequestParam(defaultValue = "id") String  
sortBy,  
            @RequestParam(defaultValue = "asc") String  
sortDir) {  
  
    Sort sort = sortDir.equalsIgnoreCase("desc")  
        ? Sort.by(sortBy).descending()  
        : Sort.by(sortBy).ascending();  
  
    Pageable pageable = PageRequest.of(page, size, sort);  
    return ResponseEntity.ok(userService.getAllUsers(pageable));  
}
```



2nd feature

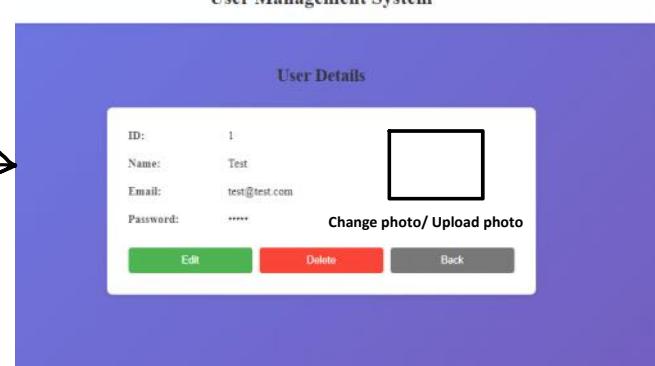
Here in this feature it is asked to upload the photo of user and save it. There are multiple ways to do it and you can save that in database also.

Solution designed here will be using AWS S3 bucket to get the photos loaded there. As of now the page looks like below.

Current



Expected



Implementation brief

1. Create S3 bucket in AWS and give it appropriate permission policy.
2. Add dependency in pom.xml and application.properties
3. Change frontend and backend code.

Bucket policy

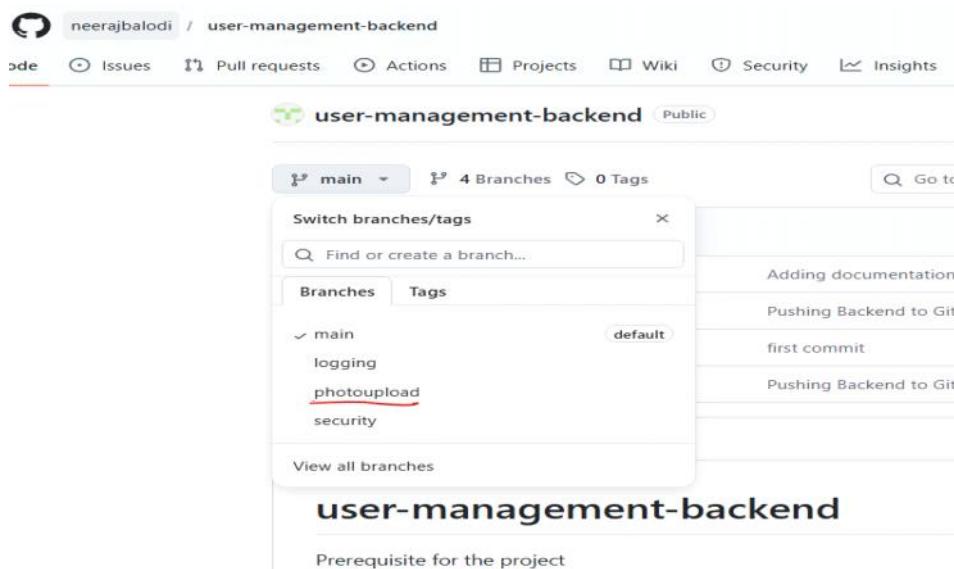
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket p

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicReadGetObject",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::your-bucket-name/*"  
        }  
    ]  
}
```

Both these features are implemented for backend and frontend and in git you can visit both below branches in git in **photoupload** branch.

Backend : <https://github.com/neerajbalodi/user-management-backend.git>

Frontend : <https://github.com/neerajbalodi/user-management-frontend.git>



The screenshot shows the GitHub repository 'user-management-backend'. The main page displays 4 branches and 0 tags. A modal window titled 'Switch branches/tags' is open, showing a list of branches: 'main' (selected), 'logging', 'photoupload' (underlined in red), and 'security'. To the right of the modal, a sidebar lists recent commits: 'Adding documentation', 'Pushing Backend to Git', 'first commit', and 'Pushing Backend to Git'.