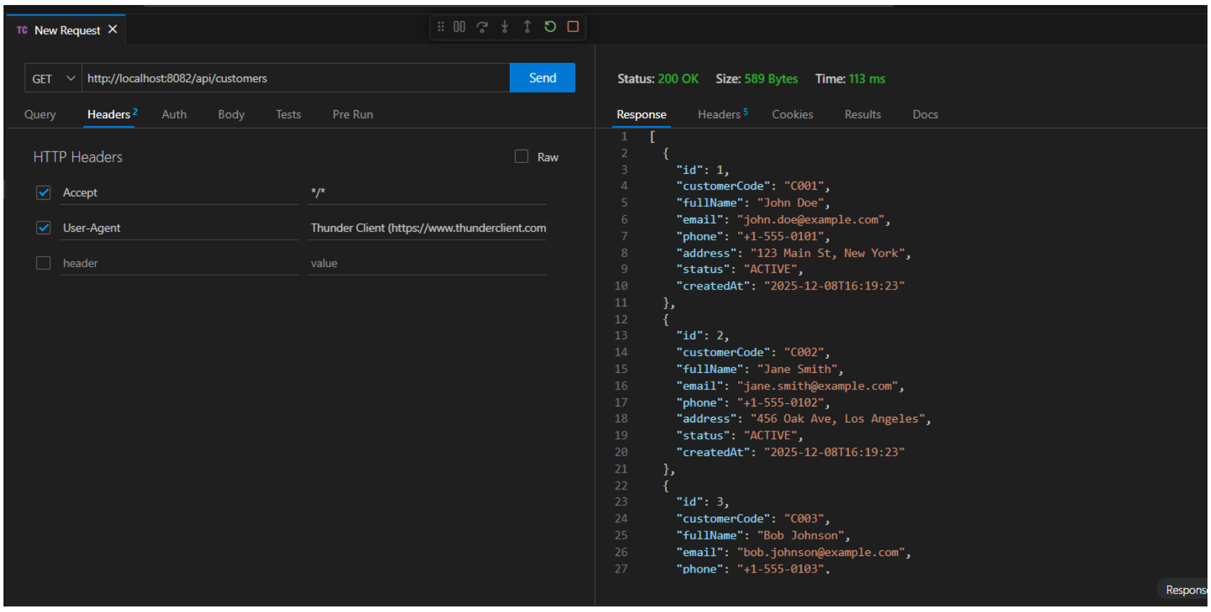
Lab 8 Practice

Name: Nguyễn Hữu Hoàng Nam

Student ID: ITCSIU23028

Get All customers api



GET http://localhost:8082/api/customers Send

✓ Accept	*/*
✓ User-Agent	Thunder Client (https://www.thunderclient.com
header	value

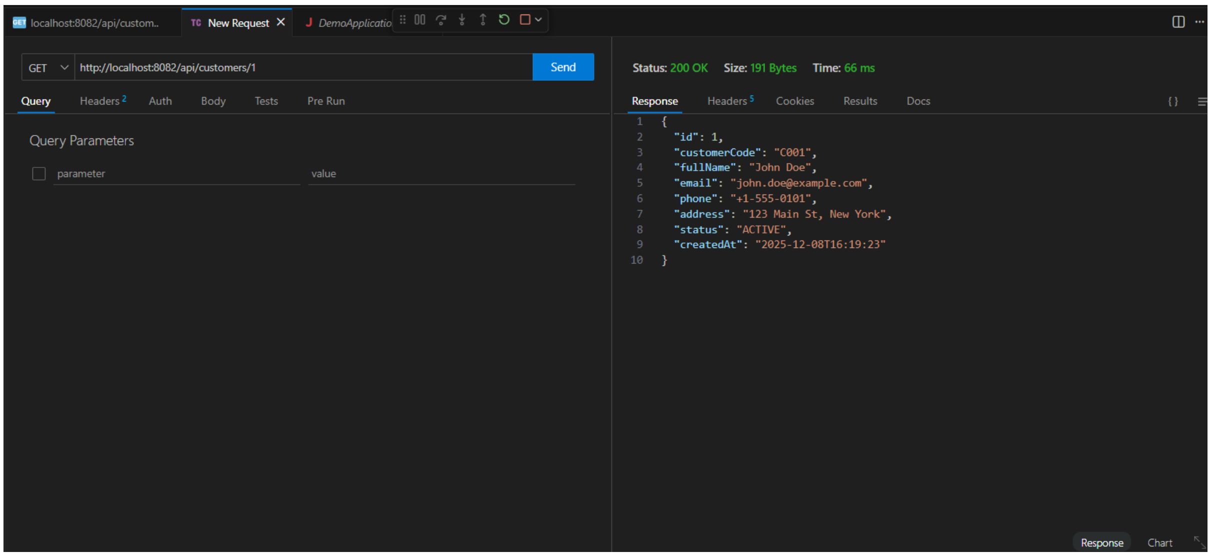
Status: 200 OK Size: 589 Bytes Time: 113 ms

```
[
  {
    "id": 1,
    "customerCode": "C001",
    "fullName": "John Doe",
    "email": "john.doe@example.com",
    "phone": "+1-555-0101",
    "address": "123 Main St, New York",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
  },
  {
    "id": 2,
    "customerCode": "C002",
    "fullName": "Jane Smith",
    "email": "jane.smith@example.com",
    "phone": "+1-555-0102",
    "address": "456 Oak Ave, Los Angeles",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
  },
  {
    "id": 3,
    "customerCode": "C003",
    "fullName": "Bob Johnson",
    "email": "bob.johnson@example.com",
    "phone": "+1-555-0103",
```

Flow:

1. Client sends GET request to /api/customers
2. CustomerRestController receives the request at getAllCustomers() method
3. Controller calls customerService.getAllCustomers()
4. CustomerServiceImpl calls customerRepository.findAll()
5. Repository queries database and returns List<Customer> entities
6. Service converts each Customer entity to CustomerResponseDTO using convertToResponseDTO()
7. Service returns List<CustomerResponseDTO> to controller
8. Controller wraps the list in ResponseEntity.ok() with HTTP 200 status
9. Spring converts DTOs to JSON format
10. Client receives JSON array of customer objects

Get customer by id api



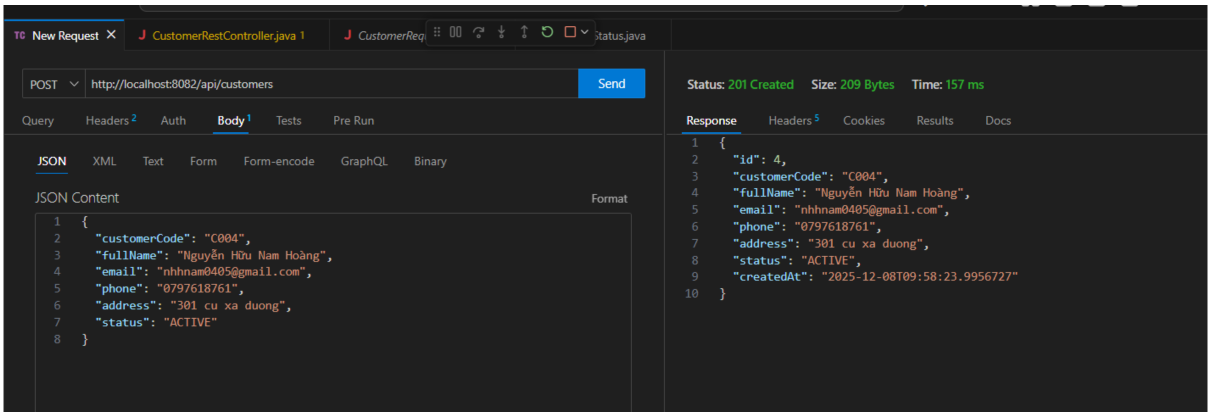
```
GET  http://localhost:8082/api/customers/1   Send
```

Status: 200 OK   Size: 191 Bytes   Time: 66 ms

```
{
  "id": 1,
  "customerCode": "C001",
  "fullName": "John Doe",
  "email": "john.doe@example.com",
  "phone": "+1-555-0101",
  "address": "123 Main St, New York",
  "status": "ACTIVE",
  "createdAt": "2025-12-08T16:19:23"
}
```

Flow:

1. Client sends GET request to /api/customers/{id}
2. Controller calls customerService.getCustomerById(id)
3. CustomerServiceImpl calls customerRepository.findById(id)
4. Repository queries database for customer with matching ID
5. If found, returns Optional<Customer> containing the entity
6. If not found, service throws ResourceNotFoundException
7. Service converts Customer entity to CustomerResponseDTO using convertToResponseDTO()
8. Service returns CustomerResponseDTO to controller
9. Controller wraps the DTO in ResponseEntity.ok() with HTTP 200 status
10. Spring converts DTO to JSON format
11. Client receives JSON object of the customer



```
POST  http://localhost:8082/api/customers   Send
```

```
{
  "customerCode": "C004",
  "fullName": "Nguyễn Hữu Nam Hoàng",
  "email": "nhhnam0405@gmail.com",
  "phone": "0797618761",
  "address": "301 cu xa duong",
  "status": "ACTIVE"
}
```

Status: 201 Created   Size: 209 Bytes   Time: 157 ms

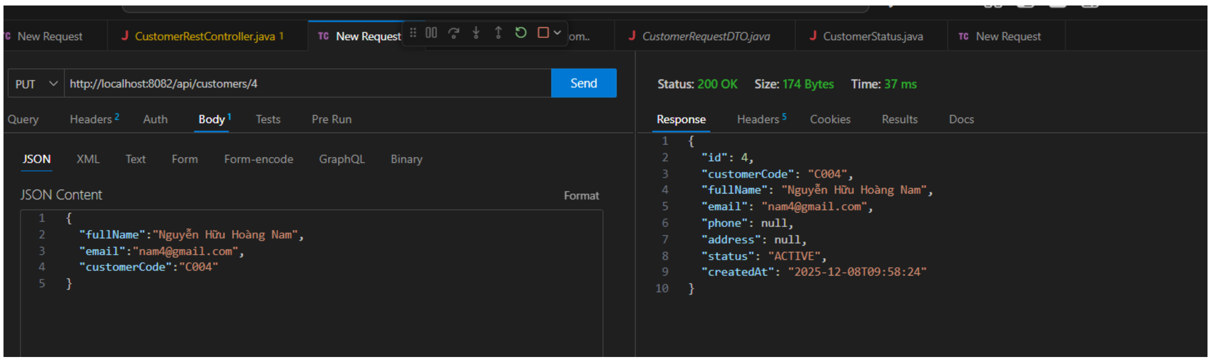
```
{
  "id": 4,
  "customerCode": "C004",
  "fullName": "Nguyễn Hữu Nam Hoàng",
  "email": "nhhnam0405@gmail.com",
  "phone": "0797618761",
  "address": "301 cu xa duong",
  "status": "ACTIVE",
  "createdAt": "2025-12-08T09:58:23.9956727"
}
```

Flow:

1. Client sends POST request to /api/customers body containing customer data
2. CustomerRestController receives the request
3. Spring converts JSON to CustomerRequestDTO
4. @Valid annotation triggers validation on the DTO fields
5. If validation fails, Spring throws exception
6. If validation passes, controller calls customerService.createCustomer(requestDTO)

7. CustomerServiceImpl checks if customerCode already exists
8. If exists, throws DuplicateResourceException
9. Service checks if email already exists using customerRepository.existsByEmail()
10. If exists, throws DuplicateResourceException
11. Service calls customerRepository to insert into database
12. Repository saves entity and returns saved Customer with generated ID and timestamps
13. Service returns CustomerResponseDTO to controller
14. Controller wraps the DTO in ResponseEntity with HTTP 201 CREATED status
15. Spring converts DTO to JSON format
16. Client receives JSON object of the newly created customer

Update and delete customer api



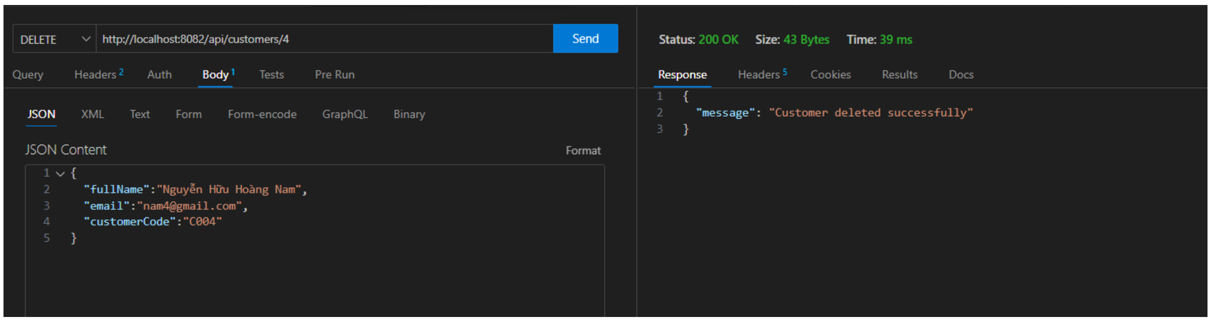
PUT http://localhost:8082/api/customers/4

JSON Content

```
{
  "fullName":"Nguyễn Hữu Hoàng Nam",
  "email":"nam4@gmail.com",
  "customerCode":"C004"
}
```

Status: 200 OK Size: 174 Bytes Time: 37 ms

```
{
  "id": 4,
  "customerCode": "C004",
  "fullName": "Nguyễn Hữu Hoàng Nam",
  "email": "nam4@gmail.com",
  "phone": null,
  "address": null,
  "status": "ACTIVE",
  "createdAt": "2025-12-08T09:58:24"
}
```



DELETE http://localhost:8082/api/customers/4

JSON Content

```
{
  "fullName":"Nguyễn Hữu Hoàng Nam",
  "email":"nam4@gmail.com",
  "customerCode":"C004"
}
```

Status: 200 OK Size: 43 Bytes Time: 39 ms

```
{
  "message": "Customer deleted successfully"
}
```

Flow(update):

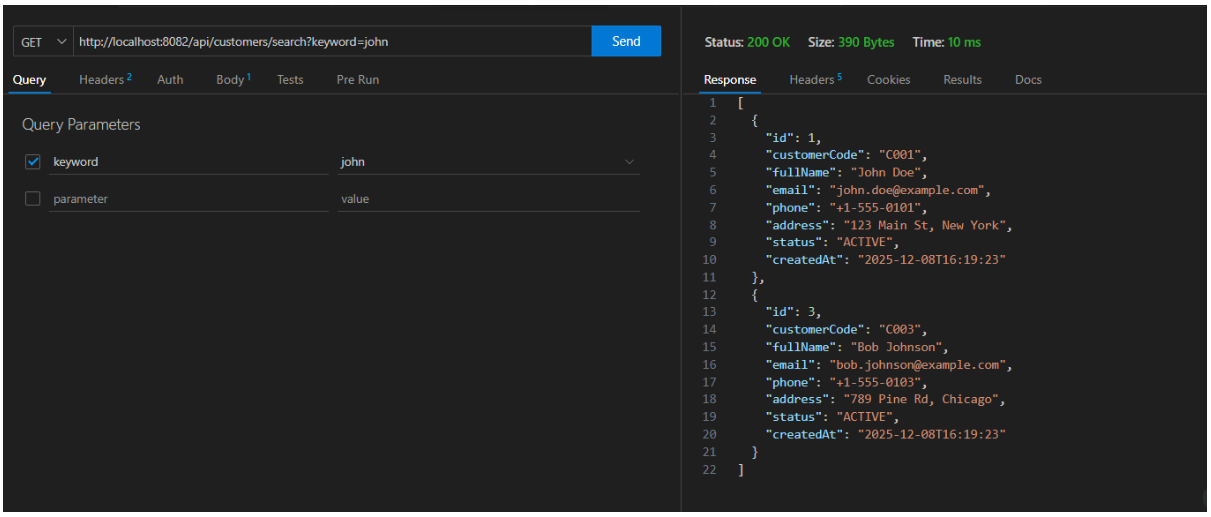
1. Client sends PUT request to /api/customers/{id}
2. Spring converts JSON to CustomerRequestDTO object
3. @Valid annotation triggers validation on the DTO fields
4. If validation fails, Spring throws exception and returns HTTP 400 Bad Request
5. If validation passes, controller calls customerService.updateCustomer(id, requestDTO)
6. CustomerServiceImpl calls customerRepository.findById(id) to get existing customer
7. If not found, throws ResourceNotFoundException
8. Service checks if email is being changed to a different one
9. If email changed, checks if new email already exists using customerRepository.existsByEmail()
10. If new email exists, throws DuplicateResourceException
11. Service updates the existing customer fields (fullName, email, phone, address)
12. Service calls customerRepository.save(existingCustomer) to update in database
13. Repository returns updated Customer entity

14. Service returns CustomerResponseDTO to controller
15. Controller wraps the DTO in ResponseEntity.ok() with HTTP 200 status
16. Spring converts DTO to JSON format
17. Client receives JSON object of the updated customer

Flow(delete):

1. Client sends DELETE request to /api/customers/{id}
2. Controller calls customerService.deleteCustomer(id)
3. CustomerServiceImpl checks if customer exists using customerRepository.existsById(id)
4. If not found, throws ResourceNotFoundException
5. If found, service calls customerRepository.deleteById(id)
6. Repository deletes the customer record from database
7. Controller wraps the map in ResponseEntity.ok() with HTTP 200 status
8. Spring converts map to JSON format: {"message": "Customer deleted successfully"}
9. Client receives JSON success message

Search API



GET http://localhost:8082/api/customers/search?keyword=john

Query Parameters

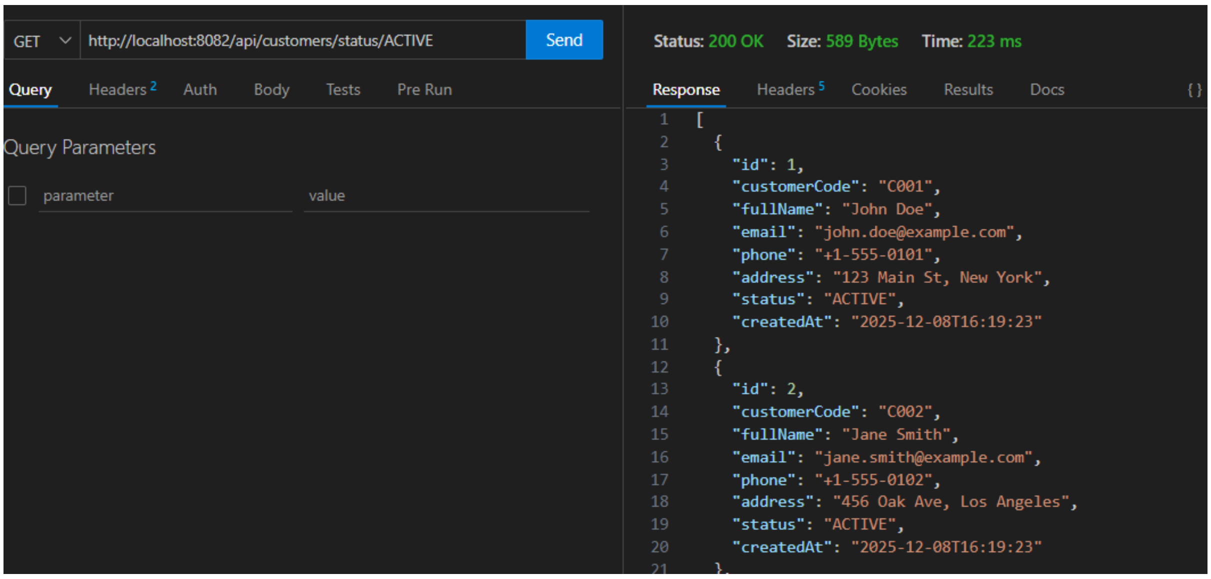
✓ keyword	john
parameter	value

Status: 200 OK Size: 390 Bytes Time: 10 ms

```
[
  {
    "id": 1,
    "customerCode": "C001",
    "fullName": "John Doe",
    "email": "john.doe@example.com",
    "phone": "+1-555-0101",
    "address": "123 Main St, New York",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
  },
  {
    "id": 3,
    "customerCode": "C003",
    "fullName": "Bob Johnson",
    "email": "bob.johnson@example.com",
    "phone": "+1-555-0103",
    "address": "789 Pine Rd, Chicago",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
  }
]
```

1. Client sends GET request to /api/customers/search?keyword={keyword}
2. Controller calls customerService.searchCustomers(keyword)
3. CustomerServiceImpl calls customerRepository.searchCustomers(keyword)
4. Repository executes custom query to search for customers matching the keyword in fullName, email, or customerCode
5. Service converts each Customer entity to CustomerResponseDTO using convertToResponseDTO()
6. Service returns List<CustomerResponseDTO> to controller
7. Controller wraps the list in ResponseEntity.ok() with HTTP 200 status
8. Spring converts DTOs to JSON format
9. Client receives JSON array of matching customer objects

   GET customer by Status API



```
http://localhost:8082/api/customers/status/ACTIVE

Status: 200 OK    Size: 589 Bytes    Time: 223 ms
```

```
[
  {
    "id": 1,
    "customerCode": "C001",
    "fullName": "John Doe",
    "email": "john.doe@example.com",
    "phone": "+1-555-0101",
    "address": "123 Main St, New York",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
  },
  {
    "id": 2,
    "customerCode": "C002",
    "fullName": "Jane Smith",
    "email": "jane.smith@example.com",
    "phone": "+1-555-0102",
    "address": "456 Oak Ave, Los Angeles",
    "status": "ACTIVE",
    "createdAt": "2025-12-08T16:19:23"
```

Flow:

1. Client sends GET request to `/api/customers/status/{status}`
2. `CustomerRestController` receives the request at `getCustomersByStatus()` method with `@PathVariable String status`
3. Controller calls `customerService.getCustomersByStatus(status)`
4. `CustomerServiceImpl` calls `customerRepository.findByStatus(status)`
5. Repository queries database for customers with matching status
6. Repository returns `List<Customer>` entities with the specified status
7. Service converts each `Customer` entity to `CustomerResponseDTO` using `convertToResponseDTO()`
8. Service returns `List<CustomerResponseDTO>` to controller
9. Controller wraps the list in `ResponseEntity.ok()` with HTTP 200 status
10. Spring converts DTOs to JSON format
11. Client receives JSON array of customer objects with the specified status