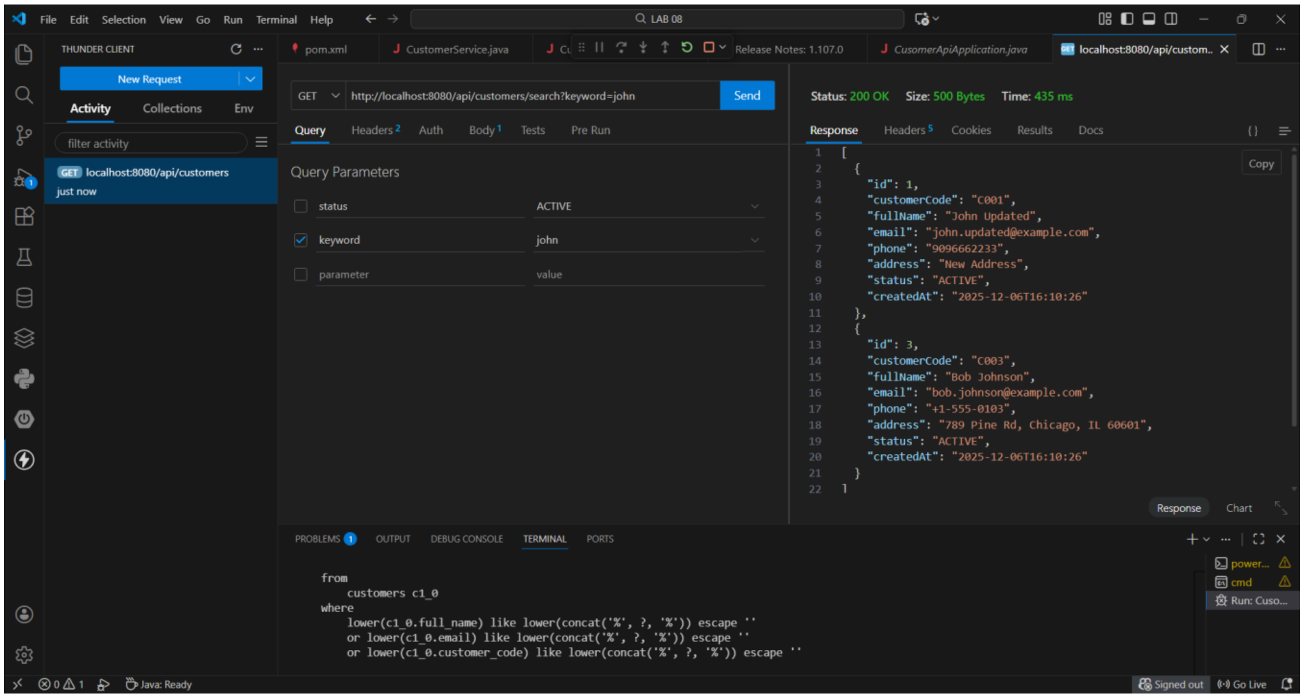
Lê Viễn Phát ITCSIU22213

Web Application Development Lab 08
REST API & DTO PATTERN
Part A – HOMEWORK

I. SEARCH & FILTER ENDPOINTS



GET http://localhost:8080/api/customers/search?keyword=john

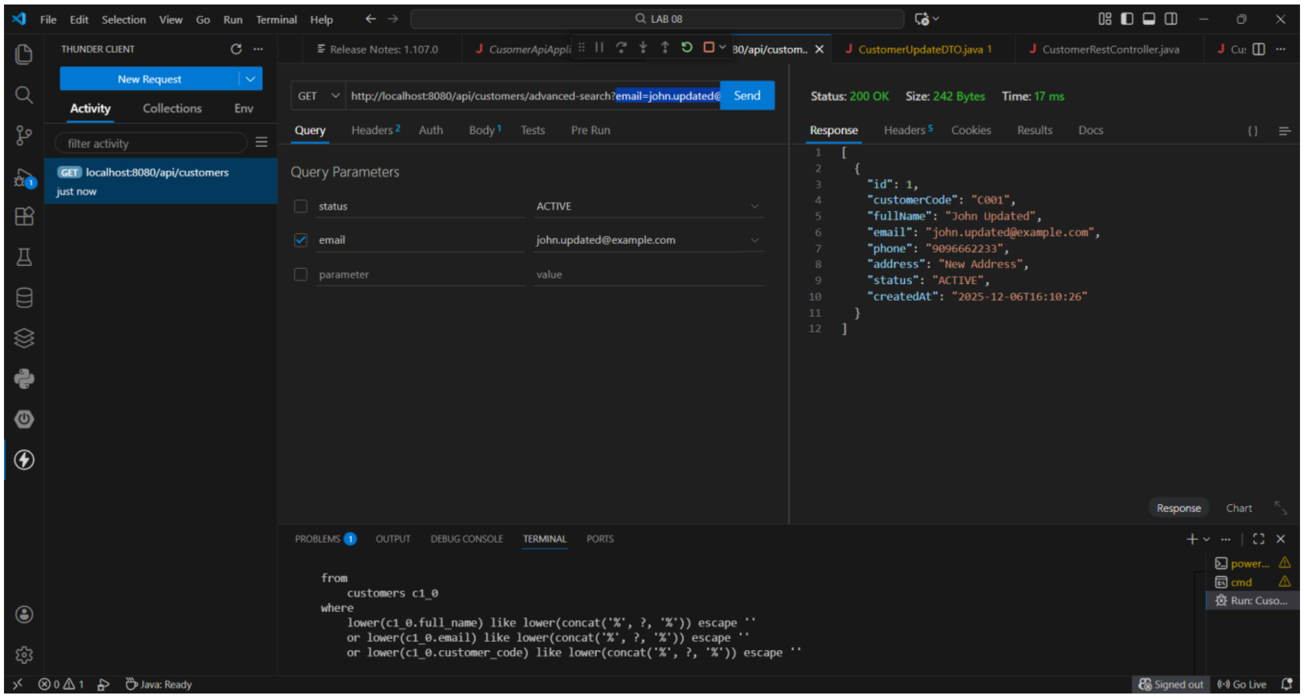
Query Parameters

☐	status	ACTIVE
✓	keyword	john
☐	parameter	value

Status: 200 OK Size: 500 Bytes Time: 435 ms

```
[
  {
    "id": 1,
    "customerCode": "C001",
    "fullName": "John Updated",
    "email": "john.updated@example.com",
    "phone": "9096662233",
    "address": "New Address",
    "status": "ACTIVE",
    "createdAt": "2025-12-06T16:10:26"
  },
  {
    "id": 3,
    "customerCode": "C003",
    "fullName": "Bob Johnson",
    "email": "bob.johnson@example.com",
    "phone": "+1-555-0103",
    "address": "789 Pine Rd, Chicago, IL 60601",
    "status": "ACTIVE",
    "createdAt": "2025-12-06T16:10:26"
  }
]
```

```
    from
        customers c1_0
    where
        lower(c1_0.full_name) like lower(concat('%', ?, '%')) escape ''
        or lower(c1_0.email) like lower(concat('%', ?, '%')) escape ''
        or lower(c1_0.customer_code) like lower(concat('%', ?, '%')) escape ''
```



GET http://localhost:8080/api/customers/advanced-search?email=john.updated@

Query Parameters

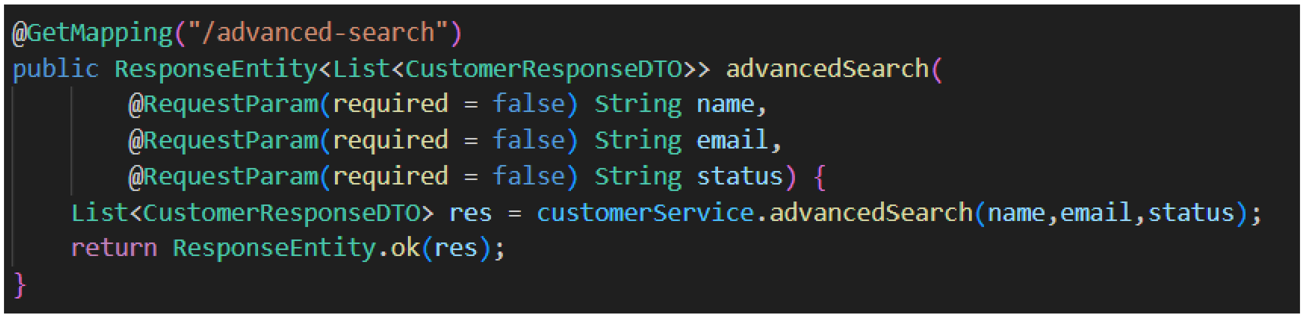
☐	status	ACTIVE
✓	email	john.updated@example.com
☐	parameter	value

Status: 200 OK Size: 242 Bytes Time: 17 ms

```
[
  {
    "id": 1,
    "customerCode": "C001",
    "fullName": "John Updated",
    "email": "john.updated@example.com",
    "phone": "9096662233",
    "address": "New Address",
    "status": "ACTIVE",
    "createdAt": "2025-12-06T16:10:26"
  }
]
```

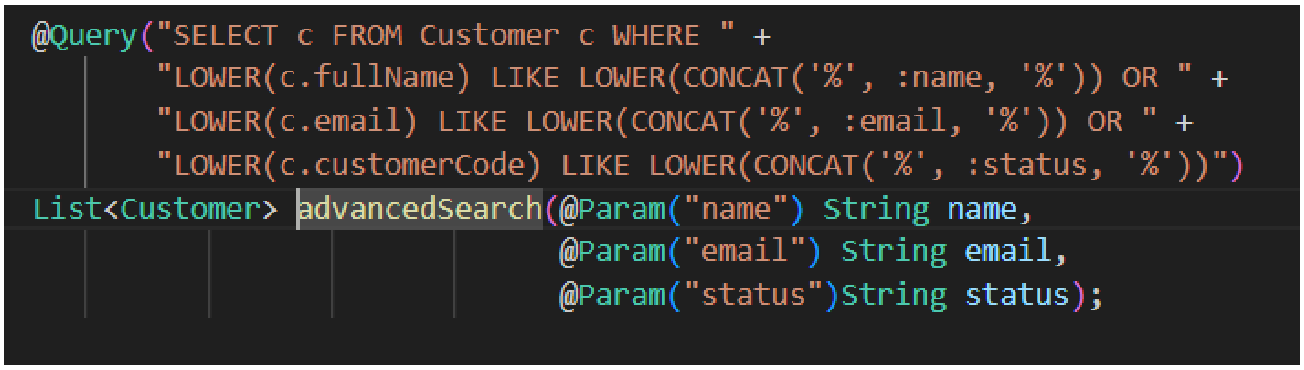
```
    from
        customers c1_0
    where
        lower(c1_0.full_name) like lower(concat('%', ?, '%')) escape ''
        or lower(c1_0.email) like lower(concat('%', ?, '%')) escape ''
        or lower(c1_0.customer_code) like lower(concat('%', ?, '%')) escape ''
```

- The endpoints are implemented in the REST controller



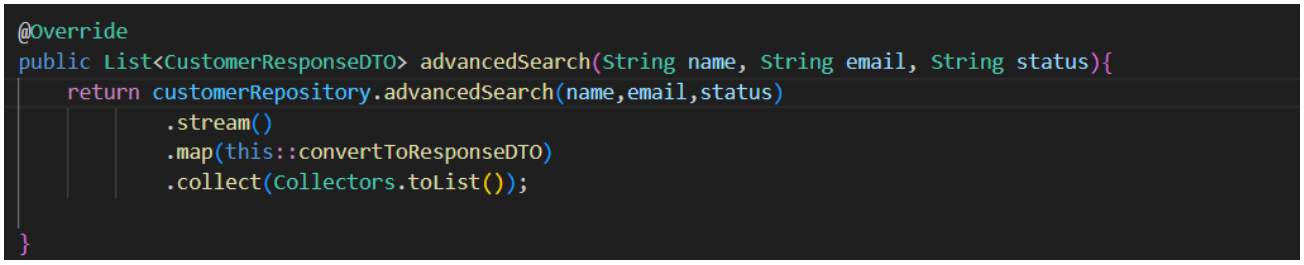
```
@GetMapping("/advanced-search")
public ResponseEntity<List<CustomerResponseDTO>> advancedSearch(
        @RequestParam(required = false) String name,
        @RequestParam(required = false) String email,
        @RequestParam(required = false) String status) {
    List<CustomerResponseDTO> res = customerService.advancedSearch(name,email,status);
    return ResponseEntity.ok(res);
}
```

- CustomerService methods are mapped to Repository methods, the query is like so



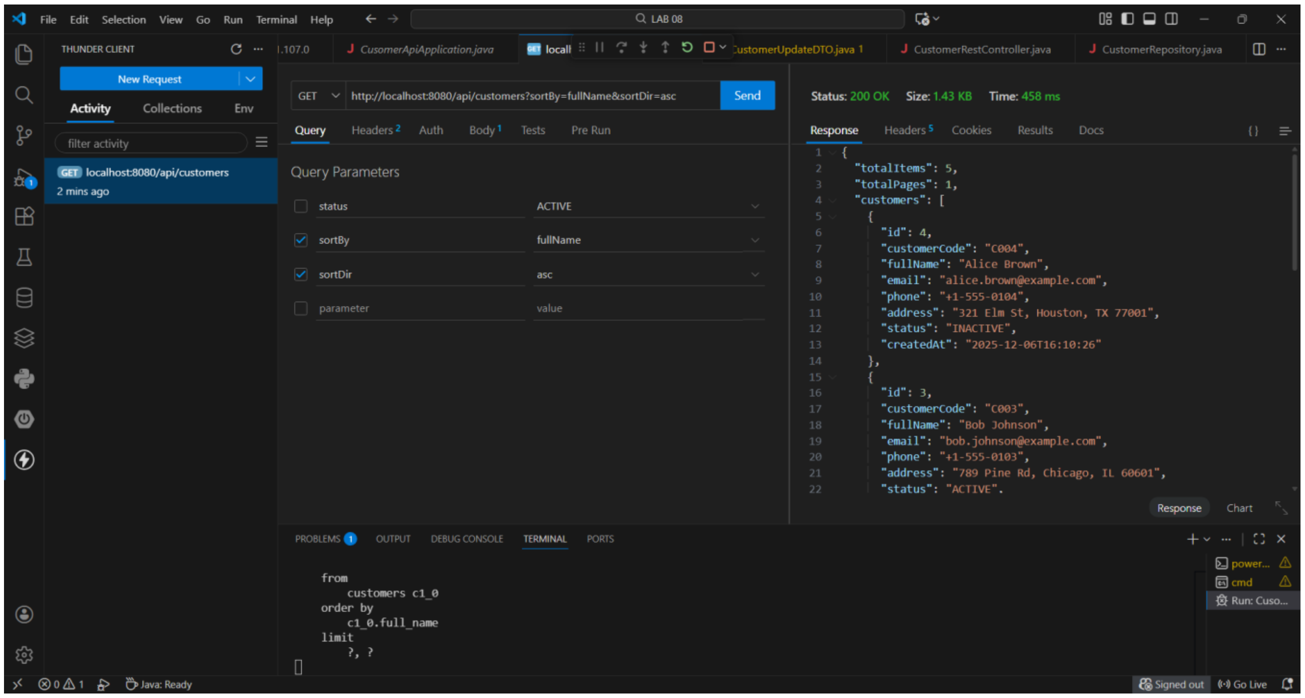
```
@Query("SELECT c FROM Customer c WHERE " +
        "LOWER(c.fullName) LIKE LOWER(CONCAT('%', :name, '%')) OR " +
        "LOWER(c.email) LIKE LOWER(CONCAT('%', :email, '%')) OR " +
        "LOWER(c.customerCode) LIKE LOWER(CONCAT('%', :status, '%'))")
List<Customer> advancedSearch(@Param("name") String name,
                               @Param("email") String email,
                               @Param("status")String status);
```

- With the list of customers, the CustomerService convert them in to a response DTO



```
@Override
public List<CustomerResponseDTO> advancedSearch(String name, String email, String status){
    return customerRepository.advancedSearch(name,email,status)
            .stream()
            .map(this::convertToResponseDTO)
            .collect(Collectors.toList());
}
```

II. PAGINATION & SORTING



```
GET http://localhost:8080/api/customers?sortBy=fullName&sortDir=asc
```

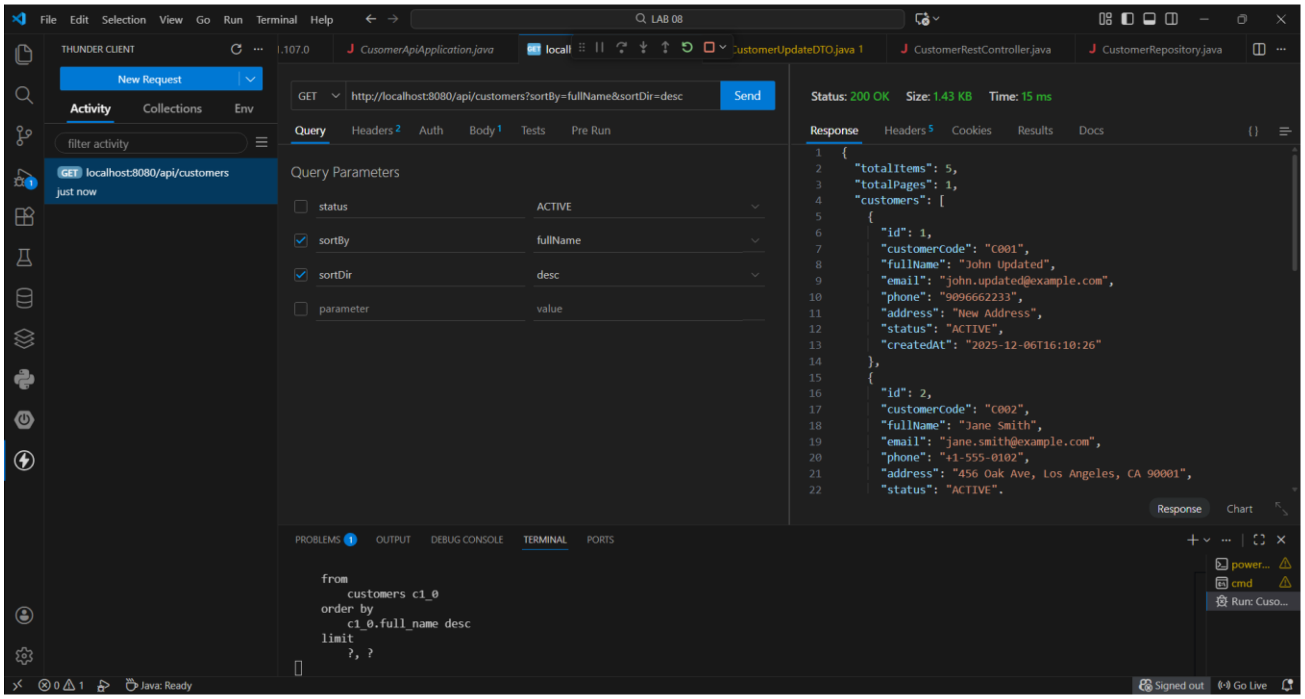
Query Parameters

☐ status	ACTIVE
✓ sortBy	fullName
✓ sortDir	asc
☐ parameter	value

Status: 200 OK Size: 1.43 KB Time: 458 ms

```
{
  "totalItems": 5,
  "totalPages": 1,
  "customers": [
    {
      "id": 4,
      "customerCode": "C004",
      "fullName": "Alice Brown",
      "email": "alice.brown@example.com",
      "phone": "+1-555-0104",
      "address": "321 Elm St, Houston, TX 77001",
      "status": "INACTIVE",
      "createdAt": "2025-12-06T16:10:26"
    },
    {
      "id": 3,
      "customerCode": "C003",
      "fullName": "Bob Johnson",
      "email": "bob.johnson@example.com",
      "phone": "+1-555-0103",
      "address": "789 Pine Rd, Chicago, IL 60601",
      "status": "ACTIVE".
```

```
    from
        customers c1_0
    order by
        c1_0.full_name
    limit
        ?, ?
```



```
GET http://localhost:8080/api/customers?sortBy=fullName&sortDir=desc
```

Query Parameters

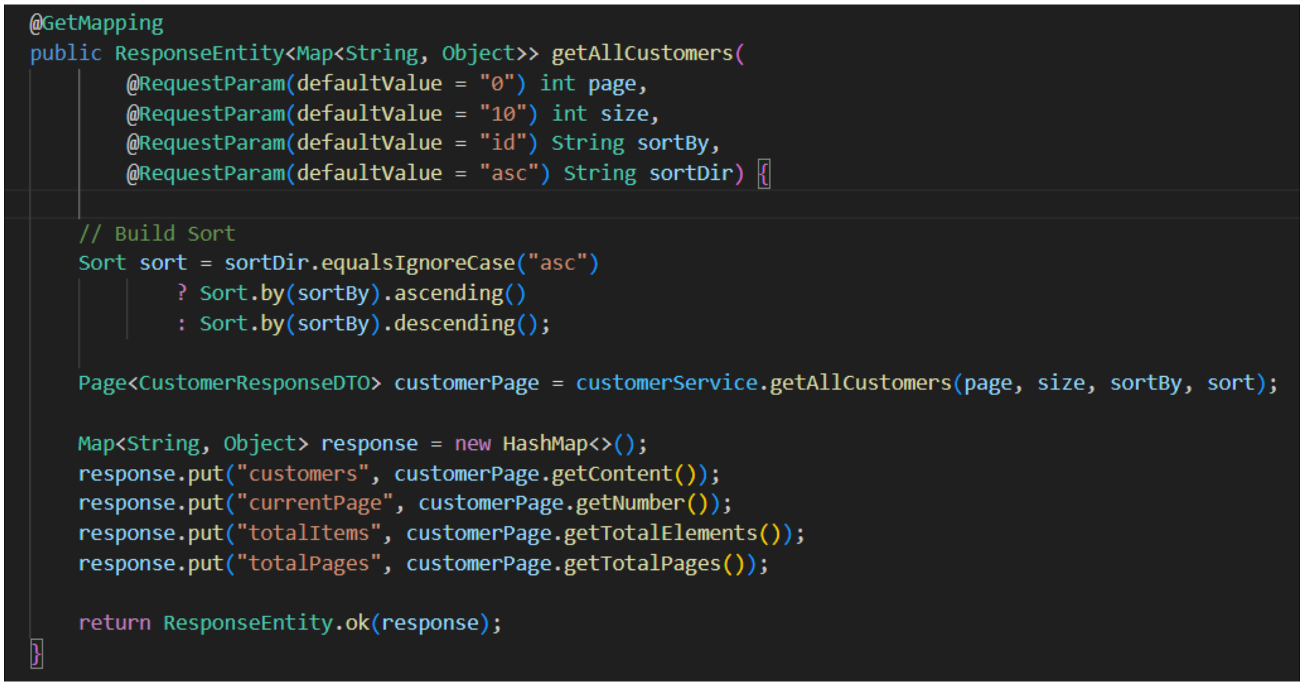
☐ status	ACTIVE
✓ sortBy	fullName
✓ sortDir	desc
☐ parameter	value

Status: 200 OK Size: 1.43 KB Time: 15 ms

```
{
  "totalItems": 5,
  "totalPages": 1,
  "customers": [
    {
      "id": 1,
      "customerCode": "C001",
      "fullName": "John Updated",
      "email": "john.updated@example.com",
      "phone": "9096662233",
      "address": "New Address",
      "status": "ACTIVE",
      "createdAt": "2025-12-06T16:10:26"
    },
    {
      "id": 2,
      "customerCode": "C002",
      "fullName": "Jane Smith",
      "email": "jane.smith@example.com",
      "phone": "+1-555-0102",
      "address": "456 Oak Ave, Los Angeles, CA 90001",
      "status": "ACTIVE".
```

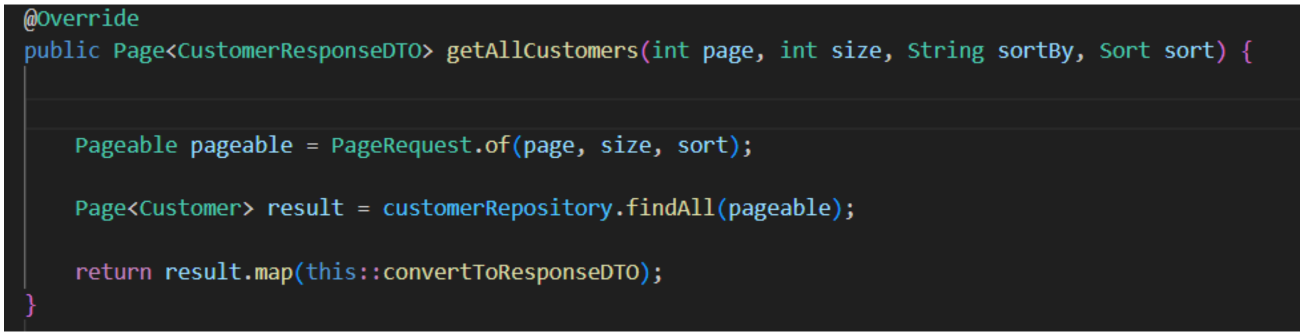
```
    from
        customers c1_0
    order by
        c1_0.full_name desc
    limit
        ?, ?
```

- The REST controller mapping takes in 4 parameters and it handles the sorting directly.



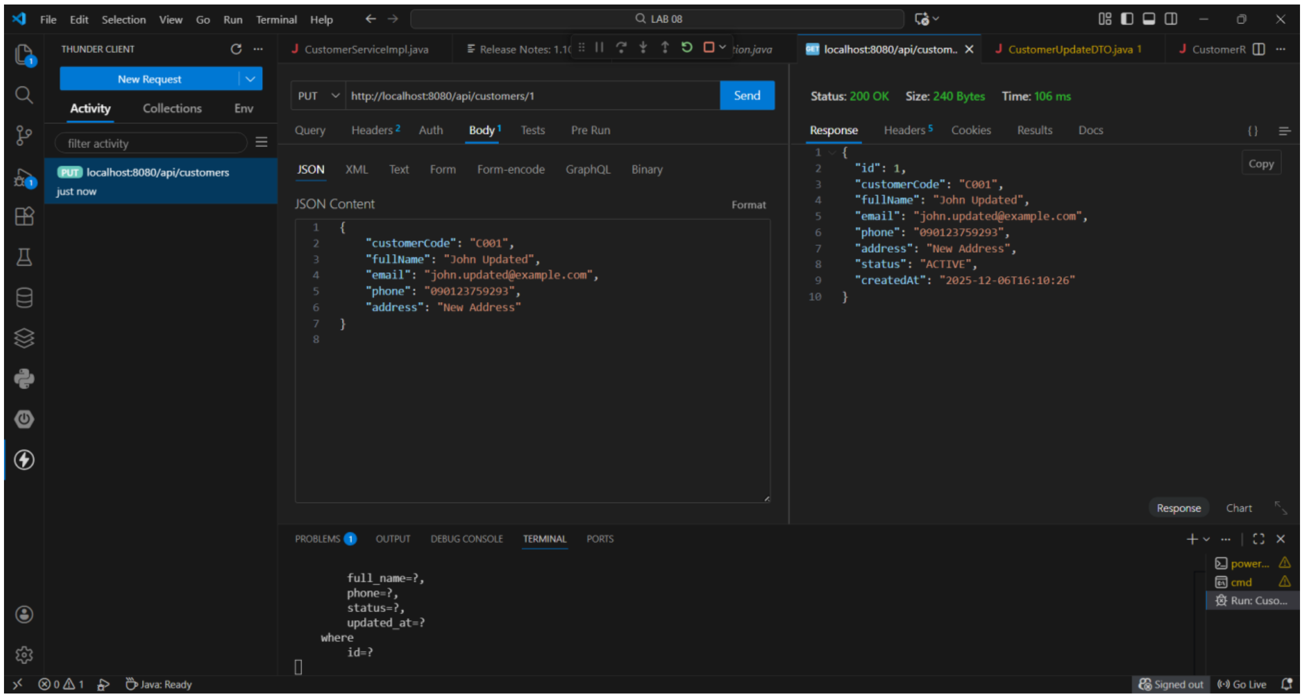
```java
@GetMapping
public ResponseEntity<Map<String, Object>> getAllCustomers(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "10") int size,
        @RequestParam(defaultValue = "id") String sortBy,
        @RequestParam(defaultValue = "asc") String sortDir) {

    // Build Sort
    Sort sort = sortDir.equalsIgnoreCase("asc")
            ? Sort.by(sortBy).ascending()
            : Sort.by(sortBy).descending();

    Page<CustomerResponseDTO> customerPage = customerService.getAllCustomers(page, size, sortBy, sort);

    Map<String, Object> response = new HashMap<>();
    response.put("customers", customerPage.getContent());
    response.put("currentPage", customerPage.getNumber());
    response.put("totalItems", customerPage.getTotalElements());
    response.put("totalPages", customerPage.getTotalPages());

    return ResponseEntity.ok(response);
}
```

The CustomerService will build the ResponseDTO using Pageable for pagination, and findAll() as the CRUD operation.



```java
@Override
public Page<CustomerResponseDTO> getAllCustomers(int page, int size, String sortBy, Sort sort) {


    Pageable pageable = PageRequest.of(page, size, sort);

    Page<Customer> result = customerRepository.findAll(pageable);

    return result.map(this::convertToResponseDTO);
}
```

III. PARTIAL UPDATE WITH PATCH



PUT http://localhost:8080/api/customers/1

Status: 200 OK Size: 240 Bytes Time: 106 ms

JSON Content

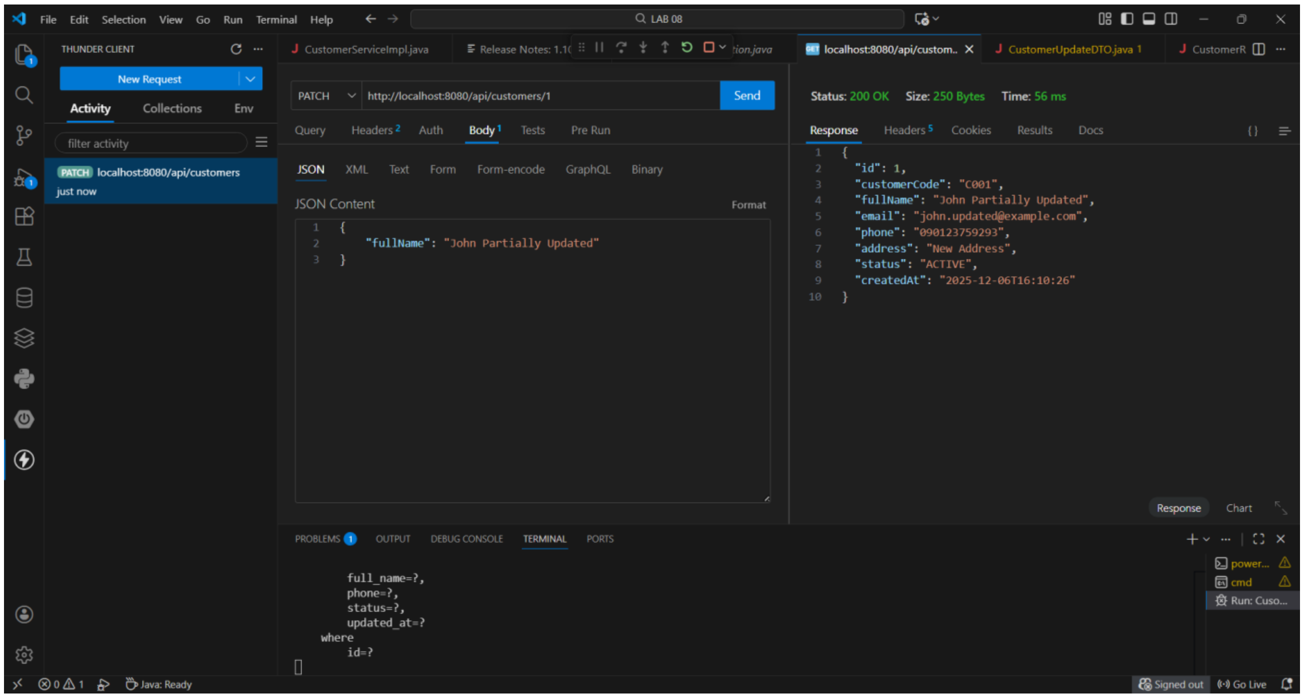
```json
{
    "customerCode": "C001",
    "fullName": "John Updated",
    "email": "john.updated@example.com",
    "phone": "090123759293",
    "address": "New Address"
}
```

Response

```json
{
  "id": 1,
  "customerCode": "C001",
  "fullName": "John Updated",
  "email": "john.updated@example.com",
  "phone": "090123759293",
  "address": "New Address",
  "status": "ACTIVE",
  "createdAt": "2025-12-06T16:10:26"
}
```

```
        full_name=?,
        phone=?,
        status=?,
        updated_at=?
    where
        id=?
```

Testing PUT



PATCH http://localhost:8080/api/customers/1

Status: 200 OK Size: 250 Bytes Time: 56 ms

JSON Content

```json
{
    "fullName": "John Partially Updated"
}
```

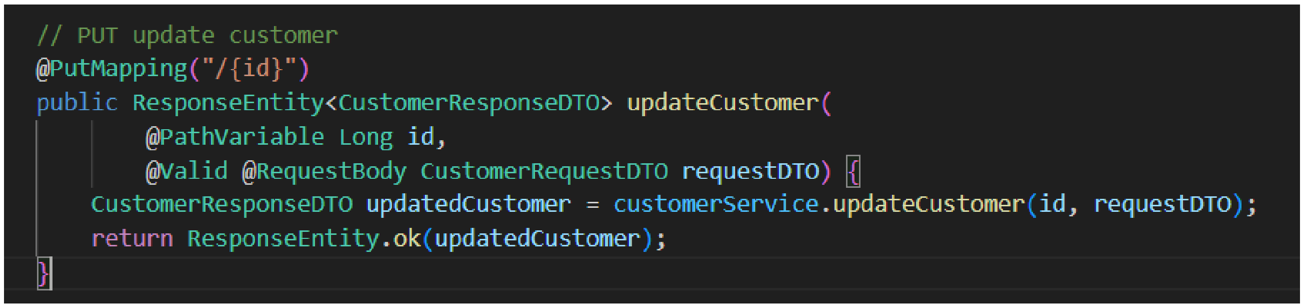
Response

```json
{
  "id": 1,
  "customerCode": "C001",
  "fullName": "John Partially Updated",
  "email": "john.updated@example.com",
  "phone": "090123759293",
  "address": "New Address",
  "status": "ACTIVE",
  "createdAt": "2025-12-06T16:10:26"
}
```

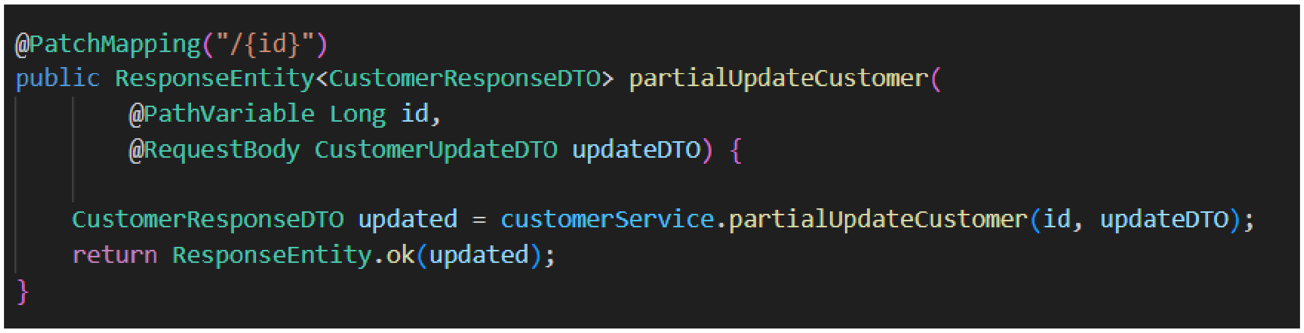
```
        full_name=?,
        phone=?,
        status=?,
        updated_at=?
    where
        id=?
```

Testing PATCH

- Both the PUT and PATCH methods use the save CRUD operation, the main difference is that while the PUT method is meant for a complete update, PATCH only update a specific field of fields, this is reflected in their use of DTO.



```java
// PUT update customer
@PutMapping("/{id}")
public ResponseEntity<CustomerResponseDTO> updateCustomer(
        @PathVariable Long id,
        @Valid @RequestBody CustomerRequestDTO requestDTO) {
    CustomerResponseDTO updatedCustomer = customerService.updateCustomer(id, requestDTO);
    return ResponseEntity.ok(updatedCustomer);
}
```



```java
@PatchMapping("/{id}")
public ResponseEntity<CustomerResponseDTO> partialUpdateCustomer(
        @PathVariable Long id,
        @RequestBody CustomerUpdateDTO updateDTO) {

    CustomerResponseDTO updated = customerService.partialUpdateCustomer(id, updateDTO);
    return ResponseEntity.ok(updated);
}
```