

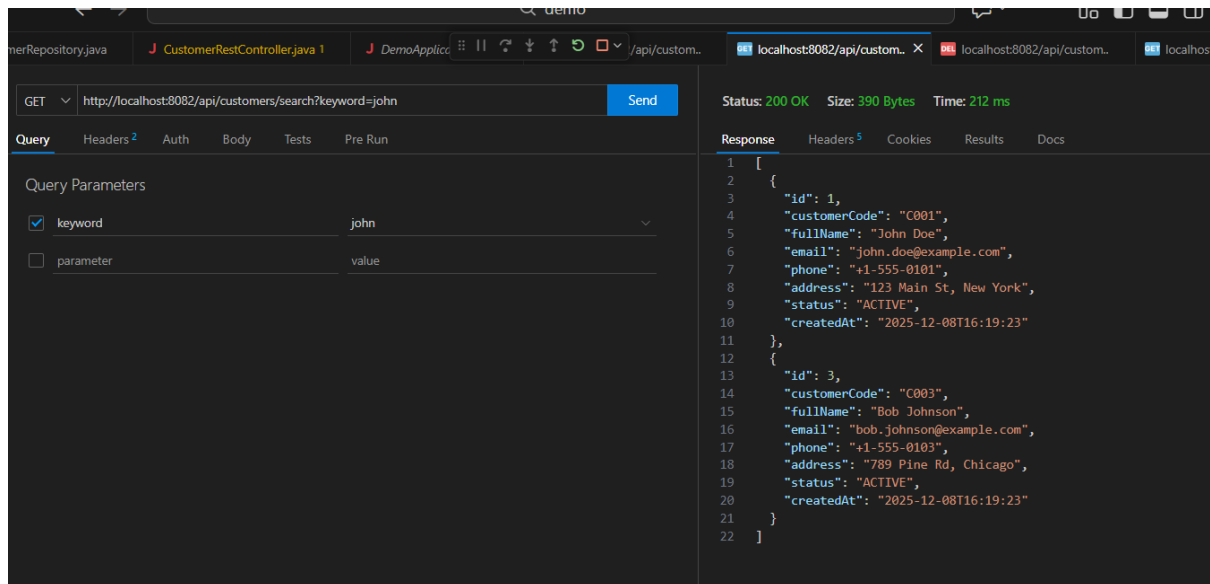
Lab 8 Homework

Name: Nguyễn Hữu Hoàng Nam

StudentID: ITCSIU23028

Exercise 5

Task 5.1



Flow:

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

Task 5.2

GET

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

Query Parameters

parameter	value

Response

```

1  [
2  {
3    "id": 1,
4    "customerCode": "C001",
5    "fullName": "John Doe",
6    "email": "john.doe@example.com",
7    "phone": "+1-555-0101",
8    "address": "123 Main St, New York",
9    "status": "ACTIVE",
10   "createdAt": "2025-12-08T16:19:23"
11  },
12  {
13    "id": 2,
14    "customerCode": "C002",
15    "fullName": "Jane Smith",
16    "email": "jane.smith@example.com",
17    "phone": "+1-555-0102",
18    "address": "456 Oak Ave, Los Angeles",
19    "status": "ACTIVE",
20    "createdAt": "2025-12-08T16:19:23"
21  }
22  ]

```

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

Task 5.3

CustomerRestController.java 1

GET

Status: 200 OK Size: 193 Bytes Time: 12 ms

Query Parameters

	name	email	parameter	value
<input checked="" type="checkbox"/>	name	john		
<input checked="" type="checkbox"/>	email	john.doe		
<input type="checkbox"/>	parameter		parameter	value

Response

```

1  [
2  {
3    "id": 1,
4    "customerCode": "C001",
5    "fullName": "John Doe",
6    "email": "john.doe@example.com",
7    "phone": "+1-555-0101",
8    "address": "123 Main St, New York",
9    "status": "ACTIVE",
10   "createdAt": "2025-12-08T16:19:23"
11  }
12  ]

```

Flow:

1. Client sends GET request to `/api/customers/advanced-search?name=john&email=example@email.com&status=active`
2. `CustomerRestController` receives the request
3. Controller calls `customerService.advancedSearch(name, email, status)`
5. Service calls `customerRepository.advancedSearch(name, email, statusEnum)`
6. Repository executes dynamic JPQL query.
8. Repository returns `List<Customer>` entities matching the search criteria
9. Service converts each `Customer` entity to `CustomerResponseDTO` using `convertToResponseDTO()`
10. Service returns `List<CustomerResponseDTO>` to controller
11. Controller wraps the list in `ResponseEntity.ok()` with HTTP 200 status
12. Spring converts DTOs to JSON format
13. Client receives JSON array of customer objects matching the advanced search criteria

EXCERSISE 6

Task 6.1

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `http://localhost:8082/api/customers?page=0&size=1`
- Status:** 200 OK
- Size:** 253 Bytes
- Time:** 220 ms
- Response:**

```
1 {
2   "totalItems": 3,
3   "totalPages": 3,
4   "customers": [
5     {
6       "id": 1,
7       "customerCode": "C001",
8       "fullName": "John Doe",
9       "email": "john.doe@example.com",
10      "phone": "+1-555-0101",
11      "address": "123 Main St, New York",
12      "status": "ACTIVE",
13      "createdAt": "2025-12-08T16:19:23"
14    }
15  ],
16  "currentPage": 0
17 }
```

Flow:

1. Client sends GET request to `/api/customers?page=0&size=10`
2. `CustomerRestController` receives the request at `getAllCustomers()`

3. Controller calls `customerService.getAllCustomers(page, size)`
4. `CustomerServiceImpl` creates a `Pageable` object using `PageRequest.of(page, size)`
5. Service calls `customerRepository.findAll(pageable)`
7. Repository returns `Page<Customer>` object containing customer entities for the requested page plus pagination metadata
8. Service uses `Page.map(this::convertToResponseDTO)` to transform `Page<Customer>` to `Page<CustomerResponseDTO>`
9. Service returns `Page<CustomerResponseDTO>` to controller
10. Controller extracts pagination data from the Page object:
11. Controller builds a `Map<String, Object>` response containing customers list and pagination metadata
12. Controller wraps the map in `ResponseEntity.ok()` with HTTP 200 status
13. Spring converts the Map to JSON format
14. Client receives JSON object with structure: `{ "customers": [...], "currentPage": 0, "totalItems": 50, "totalPages": 5 }`

Task 6.2

The screenshot shows a REST client interface with the following details:

- Request:**
 - Method: GET
 - URL: `http://localhost:8082/api/customers?sortBy=fullName&sortDir=asc`
 - Query Parameters:
 - `sortBy`: `fullName`
 - `sortDir`: `asc`
- Response:**
 - Status: 200 OK
 - Size: 649 Bytes
 - Time: 9 ms
 - JSON Body:

```

10  {
11    "phone": "+1-555-0103",
12    "address": "789 Pine Rd, Chicago",
13    "status": "ACTIVE",
14    "createdAt": "2025-12-08T16:19:23"
15  },
16  {
17    "id": 2,
18    "customerCode": "C002",
19    "fullName": "Jane Smith",
20    "email": "jane.smith@example.com",
21    "phone": "+1-555-0102",
22    "address": "456 Oak Ave, Los Angeles",
23    "status": "ACTIVE",
24    "createdAt": "2025-12-08T16:19:23"
25  },
26  {
27    "id": 1,
28    "customerCode": "C001",
29    "fullName": "John Doe",
30    "email": "john.doe@example.com",
31    "phone": "+1-555-0101",
32    "address": "123 Main St, New York",
33    "status": "ACTIVE"

```

Flow:

1. Client sends GET request to `/api/customers?page=0&size=10&sortBy=fullName&sortDir=asc`
2. `CustomerRestController` receives the request at `getAllCustomers()` method with `@RequestParam` parameters (page default=0, size default=10, sortBy optional, sortDir default="asc")

3. Controller checks if `sortBy` parameter is provided and not empty
4. If sortBy exists, controller creates a `Sort` object:
 - If sortDir is "asc": `Sort.by(sortBy).ascending()`
 - If sortDir is "desc": `Sort.by(sortBy).descending()`
5. Controller calls `customerService.getAllCustomers(page, size, sort)` if sorting is applied, or `customerService.getAllCustomers(page, size)` if no sorting
6. `CustomerServiceImpl` creates a `Pageable` object using `PageRequest.of(page, size, sort)` with sorting criteria included
7. Service calls `customerRepository.findAll(pageable)`
8. Repository queries database with LIMIT, OFFSET, and ORDER BY clauses based on the Pageable parameters
9. Repository returns `Page<Customer>` object containing sorted customer entities for the requested page plus pagination metadata
10. Service uses `Page.map(this::convertToResponseDTO)` to transform `Page<Customer>` to `Page<CustomerResponseDTO>`
11. Service returns `Page<CustomerResponseDTO>` to controller
12. Controller extracts pagination data from the Page object and builds a `Map<String, Object>` response
13. Controller wraps the map in `ResponseEntity.ok()` with HTTP 200 status
14. Spring converts the Map to JSON format
15. Client receives JSON object with sorted and paginated customer data: { "customers": [...sorted list...], "currentPage": 0, "totalItems": 50, "totalPages": 5 }

Task 6.3

The screenshot shows a REST client interface with the following details:

- Request:**
 - Method: GET
 - URL: `http://localhost:8082/api/customers?page=0&size=1&sortBy=fullName&sortDir=asc`
 - Query Parameters:

Parameter	Value
page	0
size	1
sortBy	fullName
sortDir	asc
parameter	value
- Response:**
 - Status: 200 OK
 - Size: 258 Bytes
 - Time: 262 ms
 - JSON Body:


```

1 {
2   "totalItems": 3,
3   "totalPages": 3,
4   "customers": [
5     {
6       "id": 3,
7       "customerCode": "C003",
8       "fullName": "Bob Johnson",
9       "email": "bob.johnson@example.com",
10      "phone": "+1-555-0103",
11      "address": "789 Pine Rd, Chicago",
12      "status": "ACTIVE",
13      "createdAt": "2025-12-08T16:19:23"
14    }
15  ],
16  "currentPage": 0
17 }
```

Exercise 7

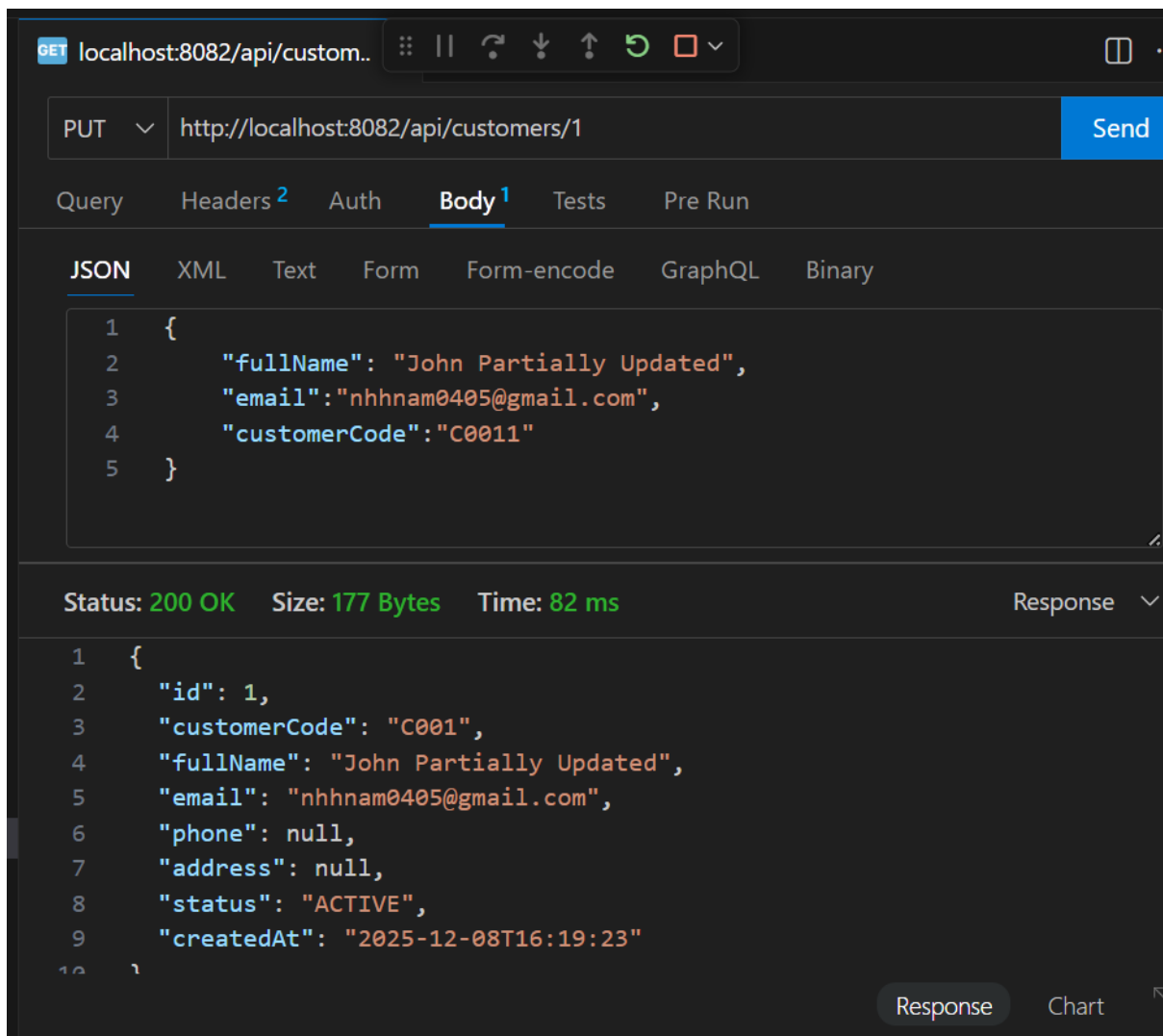
Task 7.1

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** http://localhost:8082/api/customers/1
- Buttons:** Send
- Tabs:** Query, Headers², Auth, **Body¹**, Tests, Pre Run
- Body Format:** JSON (selected), XML, Text, Form, Form-encode, GraphQL, Binary
- Request Body (JSON):**

```
1 {
2   "fullName": "John Partially Updated"
3 }
```
- Status:** 200 OK
- Size:** 205 Bytes
- Time:** 233 ms
- Response Tab:** Response (selected), Chart
- Response Body (JSON):**

```
1 {
2   "id": 1,
3   "customerCode": "C001",
4   "fullName": "John Partially Updated",
5   "email": "john.doe@example.com",
6   "phone": "+1-555-0101",
7   "address": "123 Main St, New York",
8   "status": "ACTIVE",
9   "createdAt": "2025-12-08T16:19:23"
10 }
```

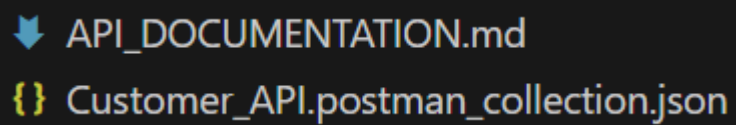


Flow:

1. Client sends PUT/PATCH request to `/api/customers/1` with complete customer data in JSON body
2. `CustomerRestController` receives the request
3. Spring validates the request DTO
4. Controller calls `customerService.updateCustomer(id, requestDTO)`
6. Repository queries database for customer with the specified ID
7. If customer not found, repository returns empty Optional and service throws `ResourceNotFoundException`
8. If customer found, service checks if email is being changed to an existing one by calling `customerRepository.existsByEmail()`
9. If email already exists for another customer, service throws `DuplicateResourceException`
10. Service updates all fields of existing customer entity
11. Service does NOT update customerCode (immutable field)

12. Service calls `customerRepository.save(existingCustomer)` to persist changes
13. Repository executes UPDATE query in database and returns updated Customer entity
14. Service converts Customer entity to CustomerResponseDTO using `convertToResponseDTO()`
15. Service returns CustomerResponseDTO to controller
16. Controller wraps DTO in `ResponseEntity.ok()` with HTTP 200 status
17. Spring converts DTO to JSON format
18. Client receives JSON object with all updated customer details

Exercise 8



↓ API_DOCUMENTATION.md
{ } Customer_API.postman_collection.json