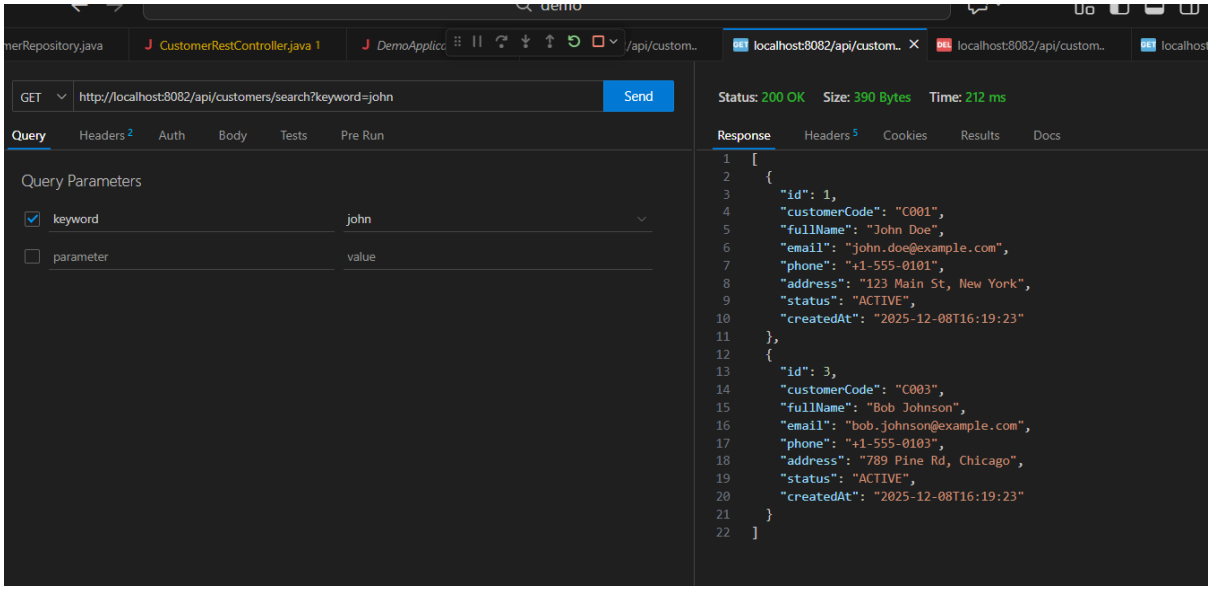
Lab 8 Homework
Name: Nguyễn Hữu Hoàng Nam

StudentID: ITCSIU23028

Exercise 5

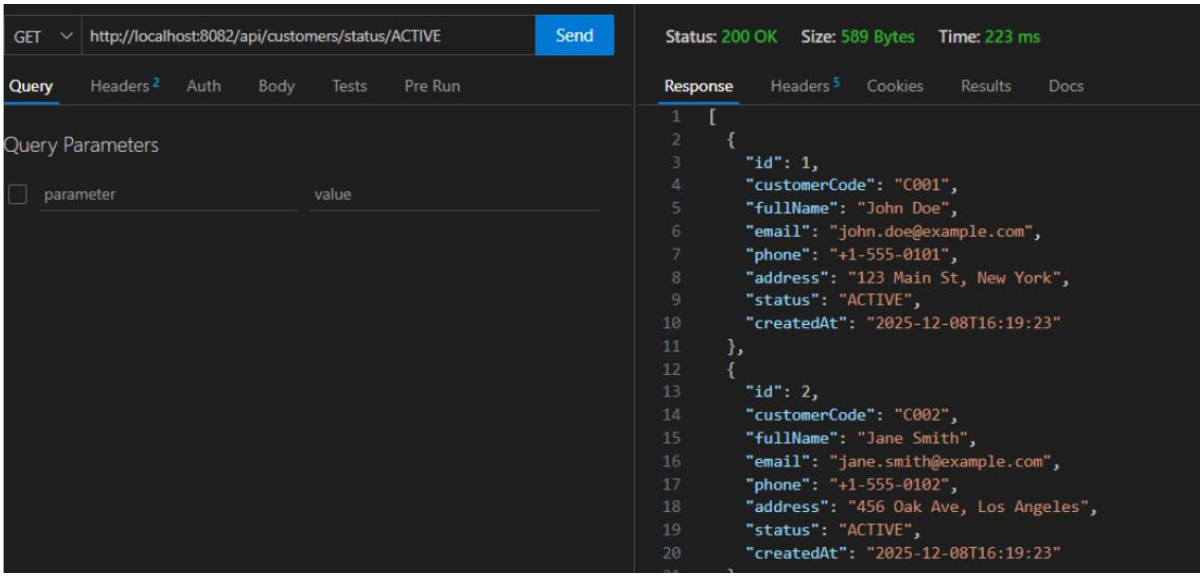
Task 5.1



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

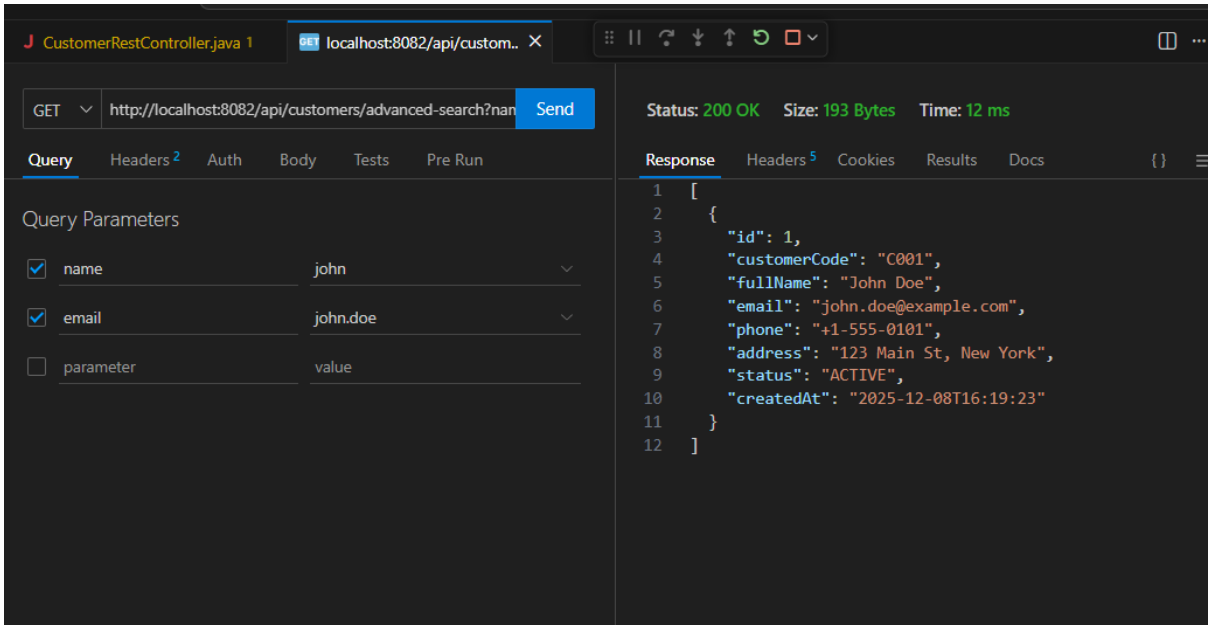
Task 5.2



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



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

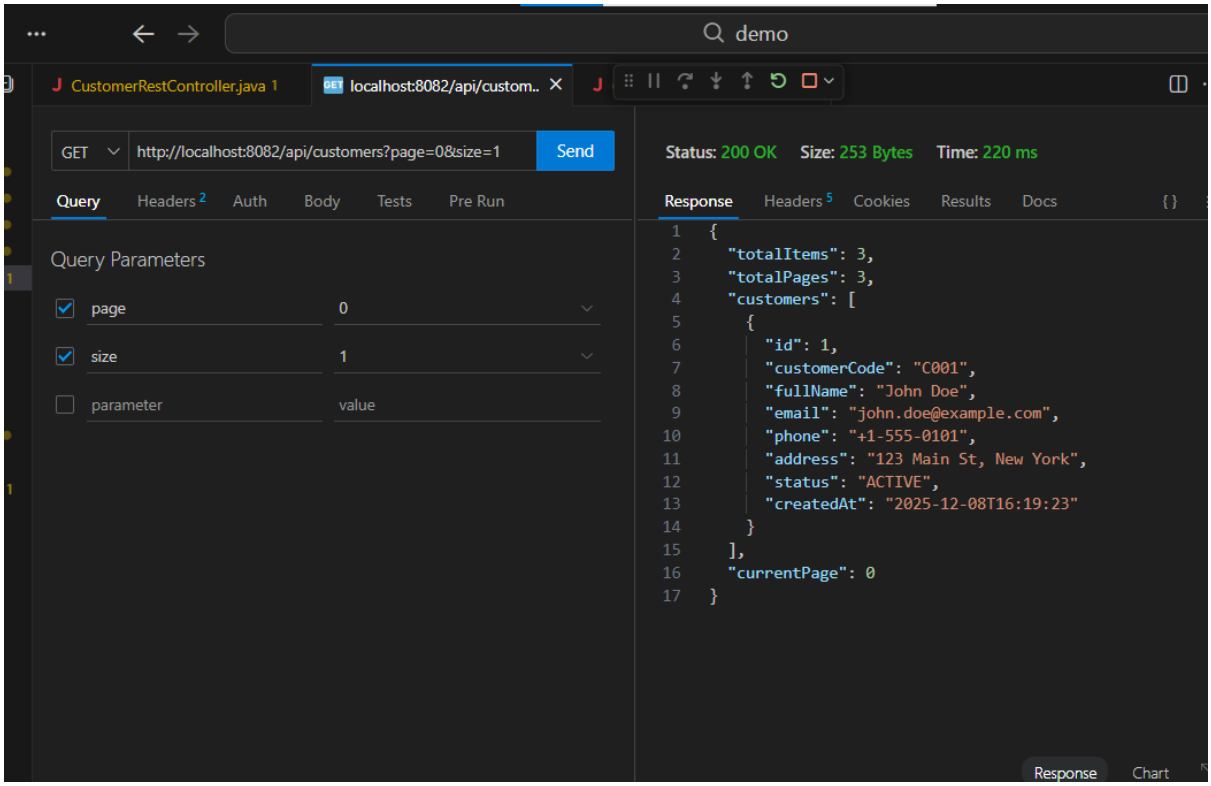
11. Controller wraps the list in `ResponseEntity.ok()` with HTTP 200 status

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



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:

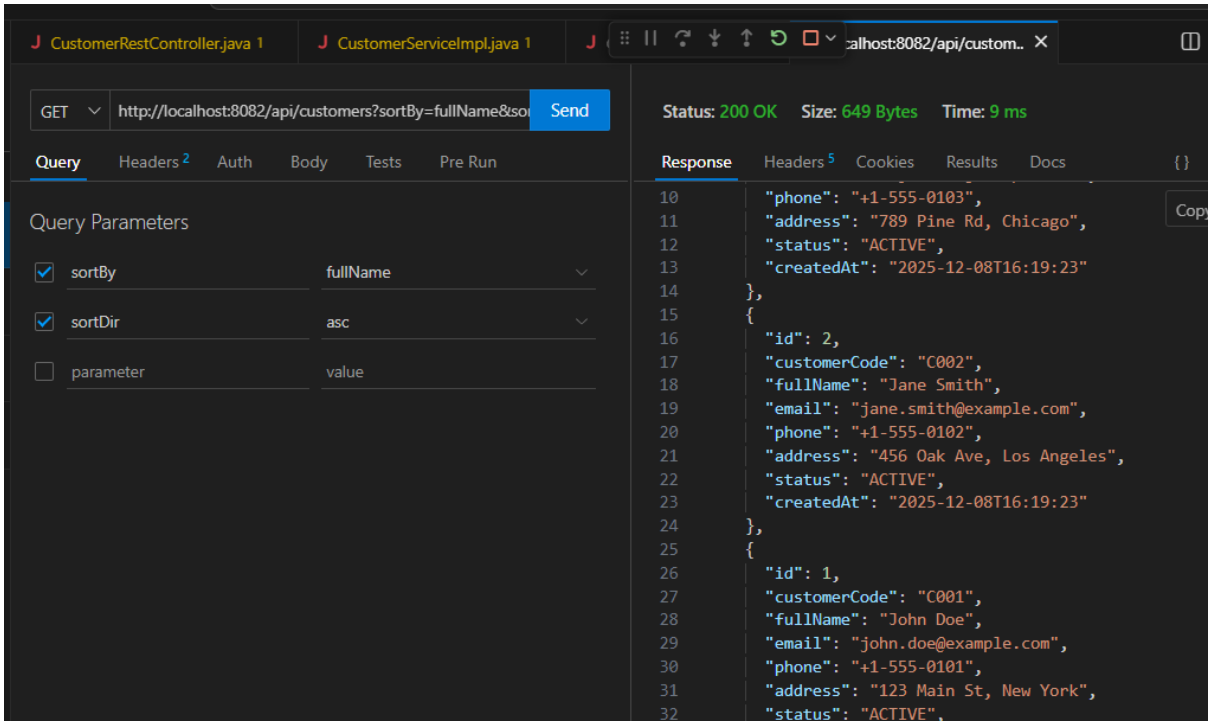
11. Controller builds a `Map<String, Object>` response containing customers list and pagination metadata

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



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

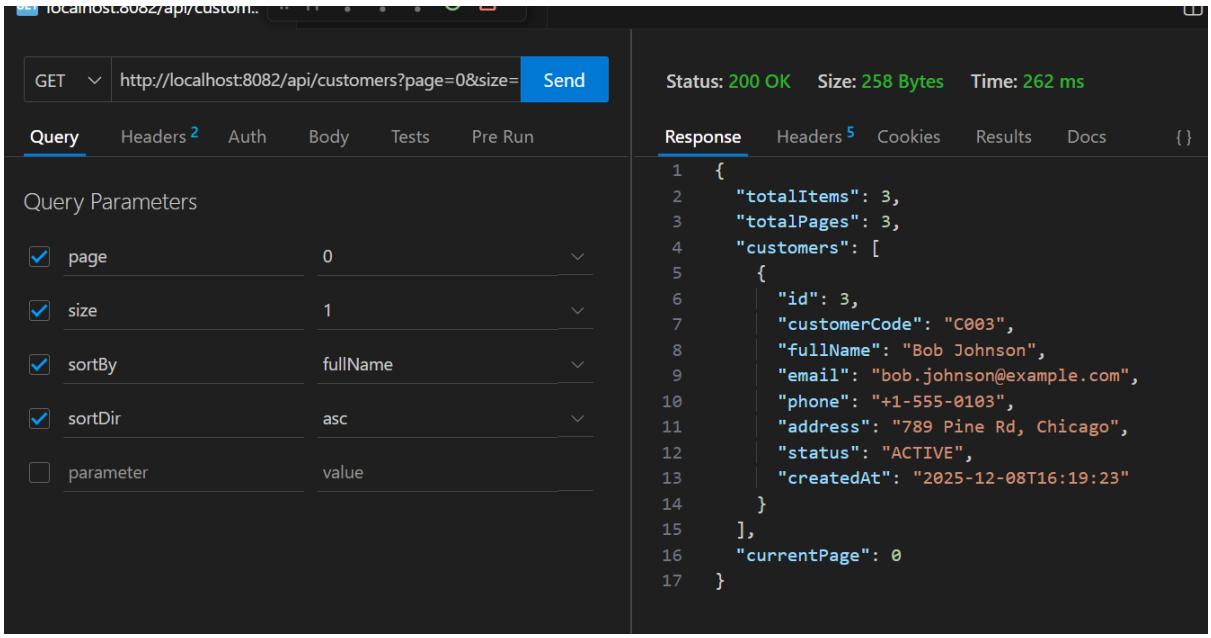
12. Controller extracts pagination data from the Page object and builds a `Map<String, Object>` response

13. Controller wraps the map in `ResponseEntity.ok()` with HTTP 200 status

14. Spring converts the Map to JSON format

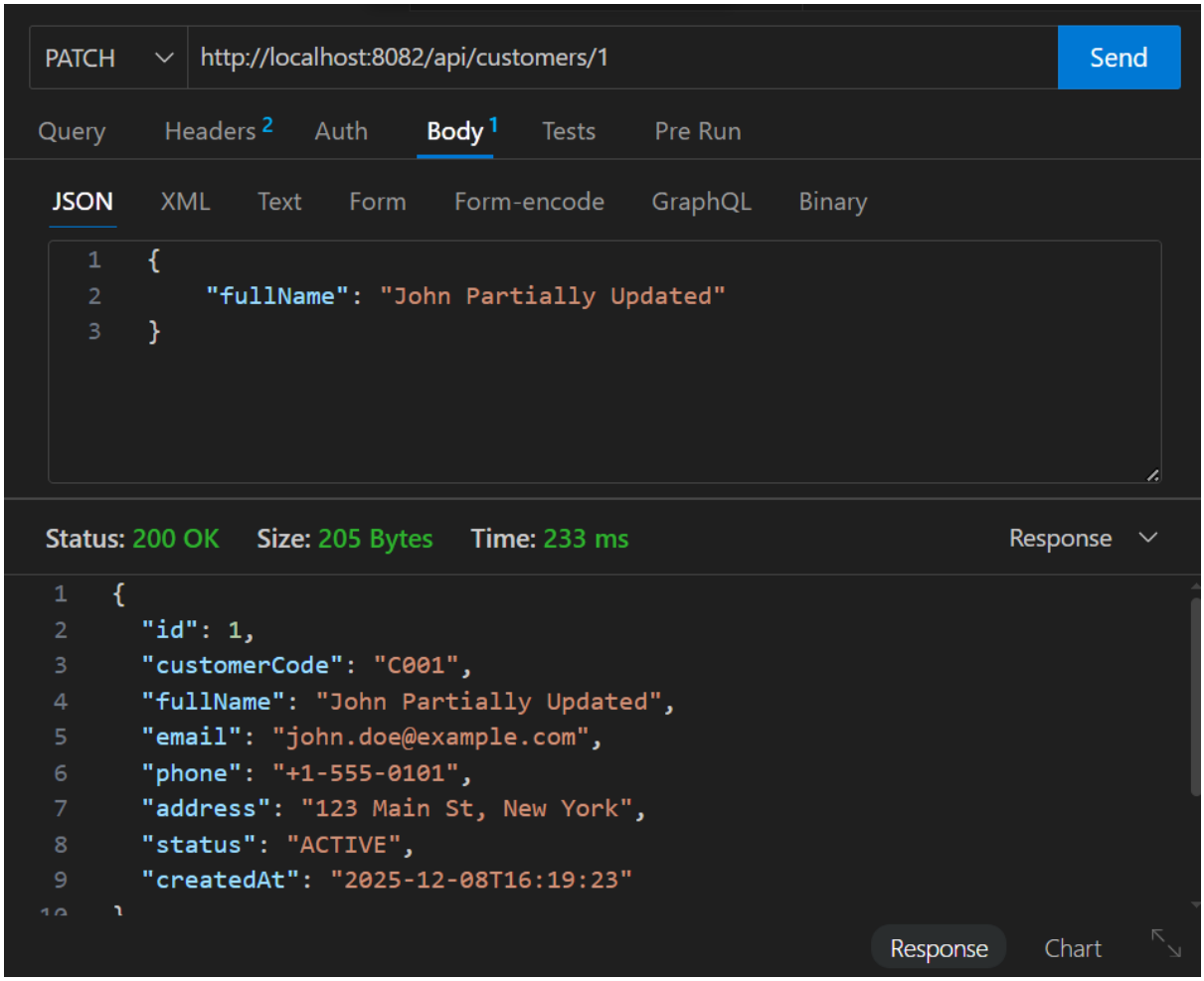
15. Client receives JSON object with sorted and paginated customer data: { "customers": [...sorted list...], "currentPage": 0, "totalItems": 50, "totalPages": 5 }

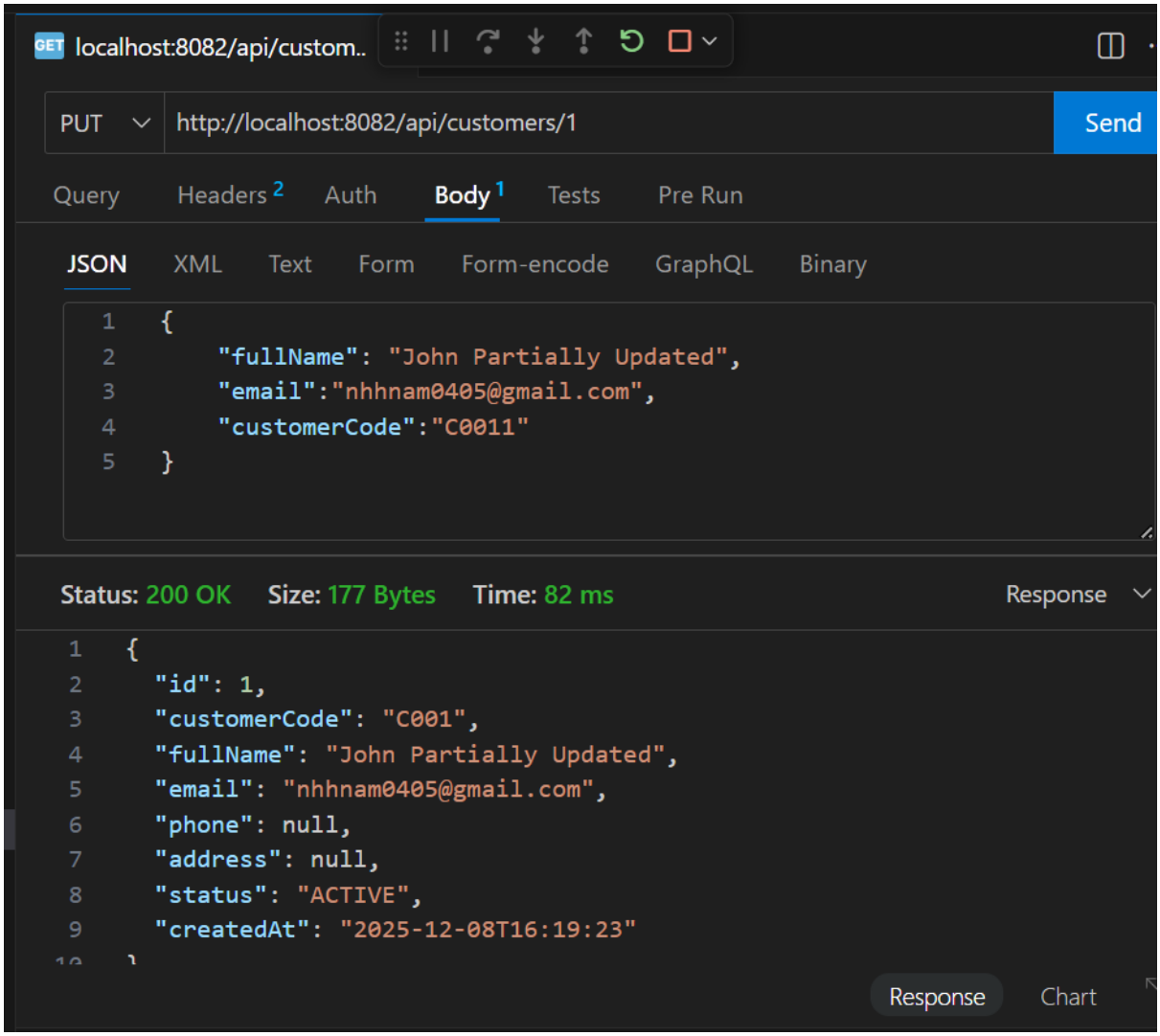
Task 6.3



Exercise 7

Task 7.1





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 requestDTO

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