

## 1. Get all customers:

- Client sends GET request to controller with endpoint '/customers'.
- Controller maps the request to getAllCustomers() and calls customerService.getAllCustomers().

```
// GET all customers
@GetMapping
public ResponseEntity<List<CustomerResponseDTO>> getAllCustomers() {
    List<CustomerResponseDTO> customers = customerService.getAllCustomers();
    return ResponseEntity.ok(customers);
}
```

- Service calls Repository to query the database.
- Repository returns an entity list to service.
- Service gets the entity and maps it to a ResponseDTO.

```
@Override
public List<CustomerResponseDTO> getAllCustomers() {
    return customerRepository.findAll()
        .stream()
        .map(this::convertToResponseDTO)
        .collect(Collectors.toList());
}
```

- Service returns the DTO to Controller.
- Controller returns a response with the DTO converted to JSON format to the client.

Query		Headers <sup>2</sup>		Auth		Body		Tests		Pre Run		Send				Status: 200 OK Size: 1.48 KB Time: 38 ms			
Query Parameters																Response Headers <sup>3</sup> Cookies Results Docs			
parameter		value														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, NY 10001", 9 "status": "ACTIVE", 10 "createdAt": "2025-12-06T16:19:21" 11 }, 12 ]			

## 2. Get customer by id:

- Client sends GET request to controller with endpoint '/customers/{id}'.
- Controller maps the request to getAllCustomers() and calls customerService.getCustomerById().

```
// GET customer by ID
@GetMapping("/{id}")
public ResponseEntity<CustomerResponseDTO> getCustomerById(@PathVariable Long id) {
    CustomerResponseDTO customer = customerService.getCustomerById(id);
    return ResponseEntity.ok(customer);
}
```

- Service calls Repository to query the database.
- Repository returns an entity to service.
- Service gets the entity and maps it to a RequestDTO.

```

@Override
public CustomerResponseDTO getCustomerById(Long id) {
    Customer customer = customerRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Customer not found with id: " + id));
    return convertToResponseDTO(customer);
}

```

- Service returns the DTO to Controller.
- Controller returns a response with the DTO converted to JSON format to the client.

Query		Headers				Body				Tests				Pre Run				Send			
GET		http://localhost:8080/api/customers/2																Send			
Query Parameters																					
<input type="checkbox"/> parameter		value																			

Status: 200 OK Size: 249 Bytes Time: 12 ms

Response Headers: Content-Type: application/json

Response Body:

```

1  {
2      "id": 2,
3      "customerCode": "C002",
4      "fullName": "Jane Smith",
5      "email": "jane.smith@example.com",
6      "phone": "+1-555-0102",
7      "address": "456 Oak Ave, Los Angeles, CA 90001",
8      "status": "ACTIVE",
9      "createdAt": "2025-12-06T16:19:21"
10 }

```

### 3. Create customer:

- Client sends POST request to controller with endpoint '/customers' and a RequestBody with the customer information in JSON format.
- Controller maps the request to createCustomer() and calls customerService.createCustomer().

```

// POST create new customer
@PostMapping
public ResponseEntity<CustomerResponseDTO> createCustomer(@Valid @RequestBody CustomerRequestDTO requestDTO) {
    CustomerResponseDTO createdCustomer = customerService.createCustomer(requestDTO);
    return ResponseEntity.status(HttpStatus.CREATED).body(createdCustomer);
}

```

- Service converts the JSON to Entity, then calls Repository to query the database.
- Repository returns an entity to service.
- Service gets the entity and maps it to a ResponseDTO.

```

// Convert DTO to Entity
Customer customer = convertToEntity(requestDTO);

// Save to database
Customer savedCustomer = customerRepository.save(customer);

// Convert Entity to Response DTO
return convertToResponseDTO(savedCustomer);

```

- Service returns the DTO to Controller.
- Controller returns a response with the status CREATED and the new customer in JSON body.

```

1 {
2   "id": 9,
3   "customerCode": "C006",
4   "fullName": "David Miller",
5   "email": "david.miller@example.com",
6   "phone": "+1555010600",
7   "address": "999 Broadway, Seattle, WA 98101"
8 }
9 "status": "ACTIVE",
10 "createdAt": "2025-12-06T11:43:54.910761575"

```

#### 4. Update customer:

- Client sends PUT request to controller with endpoint '/customers/{id}' and a RequestBody with the customer information in JSON format.
- Controller maps the request to updateCustomer() and calls customerService.updateCustomer().

```

// 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);
}

```

- Service calls Repository to check if the customer exist and create a customer entity.

```

@Override
public CustomerResponseDTO updateCustomer(Long id, CustomerRequestDTO requestDTO) {
    Customer existingCustomer = customerRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Customer not found with id: " + id));

```

- Service update the fields off the entity.
- Service calls Repository to save the entity and convert it to a ResponseDTO.

```

// Update fields
existingCustomer.setFullName(requestDTO.getFullName());
existingCustomer.setEmail(requestDTO.getEmail());
existingCustomer.setPhone(requestDTO.getPhone());
existingCustomer.setAddress(requestDTO.getAddress());

// Don't update customerCode (immutable)

Customer updatedCustomer = customerRepository.save(existingCustomer);
return convertToResponseDTO(updatedCustomer);
}

```

- Service returns the DTO to Controller.
- Controller returns a response with ok status and the updated customer in JSON body.

The screenshot shows a POST request to `http://localhost:8080/api/customers/9`. The Body tab is selected, showing a JSON payload:

```

1 {
2   "id": 9,
3   "customerCode": "C006",
4   "fullName": "David Miller",
5   "email": "david.miller@example.com",
6   "phone": "+1555010600",
7   "address": "999999 Broadway, Seattle, WA 98101"
}

```

The response status is 200 OK, size is 253 Bytes, and time is 101 ms. The response body is identical to the request body.

## 5. Delete customer:

- Client sends DELETE request to controller with endpoint '/customers/id'.
- Controller maps the request to deleteCustomer() and calls customerService.deleteCustomer().

```

// DELETE customer
@DeleteMapping("/{id}")
public ResponseEntity<Map<String, String>> deleteCustomer(@PathVariable Long id) {
    customerService.deleteCustomer(id);
    Map<String, String> response = new HashMap<>();
    response.put("message", "Customer deleted successfully");
    return ResponseEntity.ok(response);
}

```

- Service calls repository to check if a customer with the id exists and delete the customer.

```

@Override
public void deleteCustomer(Long id) {
    if (!customerRepository.existsById(id)) {
        throw new ResourceNotFoundException("Customer not found with id: " + id);
    }
    customerRepository.deleteById(id);
}

```

- Controller returns a response with the status ok.

The screenshot shows a DELETE request to `http://localhost:8080/api/customers/8`. The Body tab is selected, showing an empty JSON payload:

```

1

```

The response status is 200 OK, size is 49 Bytes, and time is 275 ms. The response body contains the message "Customer deleted successfully".

## 6. Search customer by keyword:

- Client sends GET request to controller with endpoint '/customers/search?keyword='.
- Controller maps the request to searchCustomers() and calls customerService.searchCustomers().

```
// GET search customers
@GetMapping("/search")
public ResponseEntity<List<CustomerResponseDTO>> searchCustomers(@RequestParam String keyword) {
    List<CustomerResponseDTO> customers = customerService.searchCustomers(keyword);
    return ResponseEntity.ok(customers);
}
```

- Service calls Repository to query the database.
- Repository returns an entity to service.
- Service gets the entity and maps it to a ResponseDTO.

```
@Override
public List<CustomerResponseDTO> searchCustomers(String keyword) {
    return customerRepository.searchCustomers(keyword)
        .stream()
        .map(this::convertToResponseDTO)
        .collect(Collectors.toList());
}
```

- Service returns the DTO to Controller.
- Controller returns a response with the DTO with status ok and JSON body to the client.

The screenshot shows a Thunder Client interface. The URL is set to `http://localhost:8080/api/customers/search?keyword=john`. The Headers tab is selected, showing the following configuration:

Header	Value
Accept	/*
User-Agent	Thunder Client (https://www.thunderclient.com)
Content-Type	application/json
header	value

The Response tab shows the JSON data returned by the server:

```

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, NY 10001",
9     "status": "ACTIVE",
10    "createdAt": "2025-12-06T16:19:21"
11  },
12  {
13    "id": 3,
14    "customerCode": "C003",
15    "fullName": "Bob Johnson",
16    "email": "bob.johnson@example.com",
17    "phone": "+1-555-0103",
18    "address": "789 Pine Rd, Chicago, IL 60601",
19    "status": "ACTIVE",
20    "createdAt": "2025-12-06T16:19:21"
21  }
22 ]

```

## 7. Validation error:

- Client sends a request (POST or PUT) to controller with invalid data in RequestBody.
- Controller validates the RequestBody using `@Valid` annotation and catches `MethodArgumentNotValidException`.

```
@PostMapping
public ResponseEntity<CustomerResponseDTO> createCustomer(@Valid @RequestBody CustomerRequest)
```

- GlobalExceptionHandler intercepts the exception and calls `handleValidationException()`.
- Handler extracts field errors from `BindingResult` and creates a list of validation error details.
- Handler creates an `ErrorResponseDTO` with status `BAD_REQUEST` and the validation error details.
- Handler returns the `ErrorResponseDTO` to Controller.

```

// Handle Validation Errors (400)
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<ErrorResponseDTO> handleValidationException(
    MethodArgumentNotValidException ex,
    WebRequest request) {

    List<String> details = new ArrayList<>();
    for (FieldError error : ex.getBindingResult().getFieldErrors()) {
        details.add(error.getField() + ": " + error.getDefaultMessage());
    }

    ErrorResponseDTO error = new ErrorResponseDTO(
        HttpStatus.BAD_REQUEST.value(),
        error: "Validation Failed",
        message: "Invalid input data",
        request.getDescription(false).replace("uri=", ""))
    );
    error.setDetails(details);

    return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
}

```

- Controller returns a response with status 400 (BAD\_REQUEST) and the error details in JSON body to the client.

The screenshot shows a Postman request to `http://localhost:8080/api/customers`. The request method is `POST`. The `Body` tab is selected, showing a JSON payload:

```

1  {
2      "customerCode": "C006",
3      "fullName": "David Miller",
4      "email": "david.miller@example.com",
5      "phone": "+15550106",
6      "address": "999999 Broadway, Seattle, WA 98101"
7  }

```

The response status is `400 Bad Request`, size is `223 Bytes`, and time is `8 ms`. The response body is:

```

1  {
2      "status": 400,
3      "error": "Validation Failed",
4      "message": "Invalid input data",
5      "path": "/api/customers",
6      "details": [
7          "phone: Invalid phone number format"
8      ],
9      "timestamp": "2025-12-06T11:45:31.137352258"
10 }

```

## 8. Resource not found:

- Client sends a request (GET, PUT, or DELETE) to controller with an invalid customer id.
- Controller maps the request and calls the corresponding customerService method.
- Service calls Repository to query the database by id.
- Repository returns empty Optional because customer doesn't exist.
- Service throws ResourceNotFoundException.
- GlobalExceptionHandler intercepts the exception and calls `handleResourceNotFoundException()`.
- Handler creates an ErrorResponseDTO with status NOT\_FOUND.
- Handler returns the ErrorResponseDTO to Controller.

```

// Handle ResourceNotFoundException (404)
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<ErrorResponseDTO> handleResourceNotFoundException(
    ResourceNotFoundException ex,
    WebRequest request) {

    ErrorResponseDTO error = new ErrorResponseDTO(
        HttpStatus.NOT_FOUND.value(),
        error: "Not Found",
        ex.getMessage(),
        request.getDescription(false).replace("uri=", ""))
);

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
}

```

- Controller returns a response with status 404 (NOT\_FOUND) and the error details in JSON body to the client.

GET	http://localhost:8080/api/customers/999	Send						
Query	Headers 3	Auth	<u>Body</u> 1	Tests	Pre Run			
JSON	XML	Text	Form	Form-encode	GraphQL	Binary		
JSON Content				Format				
<pre> 1 { 2     "customerCode": "C006", 3     "fullName": "David Miller", 4     "email": "david.miller@example.com", 5     "phone": "+15550106", 6     "address": "999999 Broadway, Seattle, WA 98101" 7 }</pre>								
				Response	Headers 3	Cookies	Results	Docs
				Status: 404 Not Found   Size: 195 Bytes   Time: 12 ms <pre> 1 { 2     "status": 404, 3     "error": "Not Found", 4     "message": "Customer not found with id: 999", 5     "path": "/api/customers/999", 6     "details": null, 7     "timestamp": "2025-12-06T11:45:54.60998681" 8 }</pre>				

## 9. Duplicate resource:

- Client sends a POST request to create a customer with existing customerCode or email.
- Controller maps the request to createCustomer() and calls customerService.createCustomer().
- Service calls Repository to check if customerCode or email already exists.
- Repository returns true for existsByCustomerCode() or existsByEmail().
- Service throws DuplicateResourceException.
- GlobalExceptionHandler intercepts the exception and calls handleDuplicateResourceException().
- Handler creates an ErrorResponseDTO with status CONFLICT.
- Handler returns the ErrorResponseDTO to Controller.

```

// Handle DuplicateResourceException (409)
@ExceptionHandler(DuplicateResourceException.class)
public ResponseEntity<ErrorResponseDTO> handleDuplicateResourceException(
    DuplicateResourceException ex,
    WebRequest request) {

    ErrorResponseDTO error = new ErrorResponseDTO(
        HttpStatus.CONFLICT.value(),
        error: "Conflict",
        ex.getMessage(),
        request.getDescription(false).replace("uri=", ""))
);

    return new ResponseEntity<>(error, HttpStatus.CONFLICT);
}

```

- Controller returns a response with status 409 (CONFLICT) and the error details in JSON body to the client.

The screenshot shows a POST request to `http://localhost:8080/api/customers`. The request body contains a JSON object representing a customer. The response is a 409 Conflict status with a JSON error message indicating that a customer code already exists.

Body	Response
<pre>1 {   "customerCode": "C006",   "fullName": "David Miller",   "email": "david.miller@example.com",   "phone": "+1555010600",   "address": "99999 Broadway, Seattle, WA 98101" }</pre>	<pre>1 {   "status": 409,   "error": "Conflict",   "message": "Customer code already exists: C006",   "path": "/api/customers",   "details": null,   "timestamp": "2025-12-06T11:46:35.742996658" }</pre>