1. **Get all customers:**

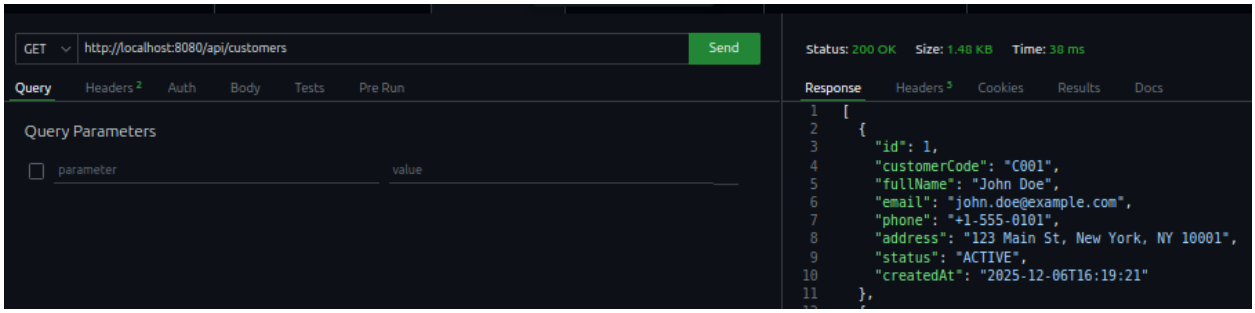
- Client sends GET request to controller with endpoint '/customers'.
- Controller maps the request to getAllCustomers() and calls customerService.getAllCustomers().

```java
// 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.

```java
@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.



2. **Get customer by id:**

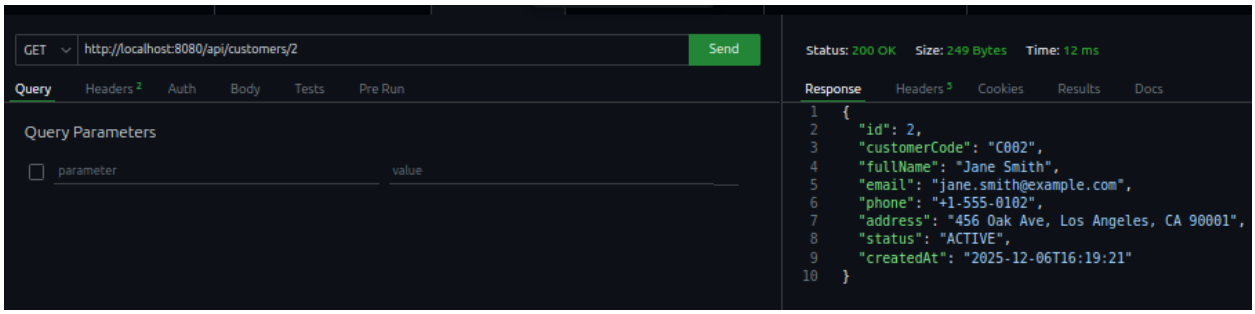
- Client sends GET request to controller with endpoint '/customers/id'.
- Controller maps the request to getAllCustomers() and calls customerService.getCustomerById().

```java
// 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.

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

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



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().

```java
// 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);
}
```

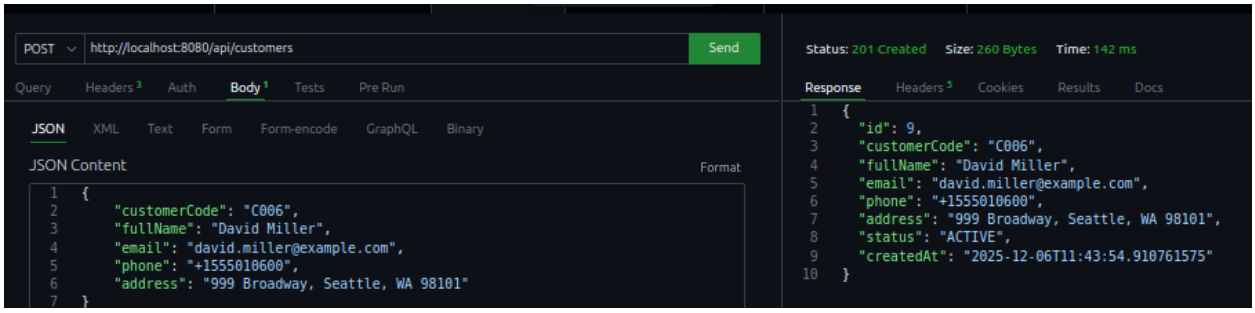
- Service converts the JSON to Entity, then calls Repository to query the database.
- Repository returns an entity to service.
- Service gets the entity and maps it to a ResponseDTO.

```java
// 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.



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().

```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);
}
```

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

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

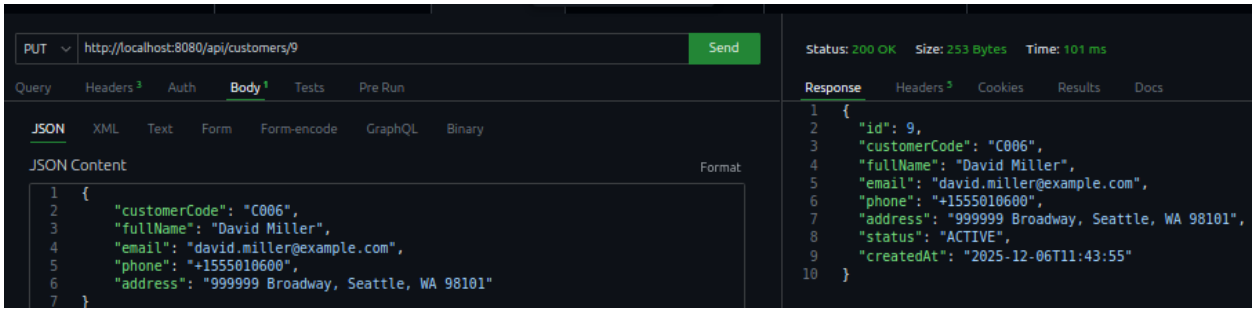
- Service update the fields off the entity.
- Service calls Repository to save the entity and convert it to a ResponseDTO.

```java
    // 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.



5. **Delete customer:**

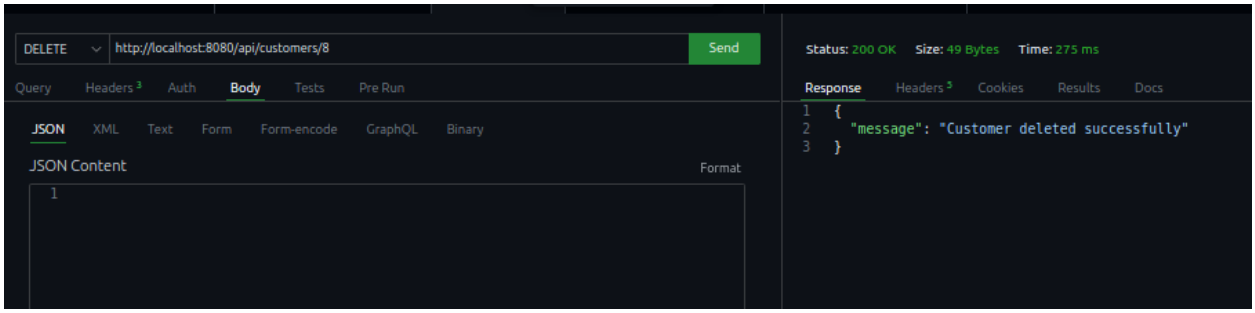
- Client sends DELETE request to controller with endpoint '/customers/id'.
- Controller maps the request to deleteCustomer() and calls customerService.deleteCustomer().

```java
// 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.

```java
@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.



6. **Search customer by keyword:**

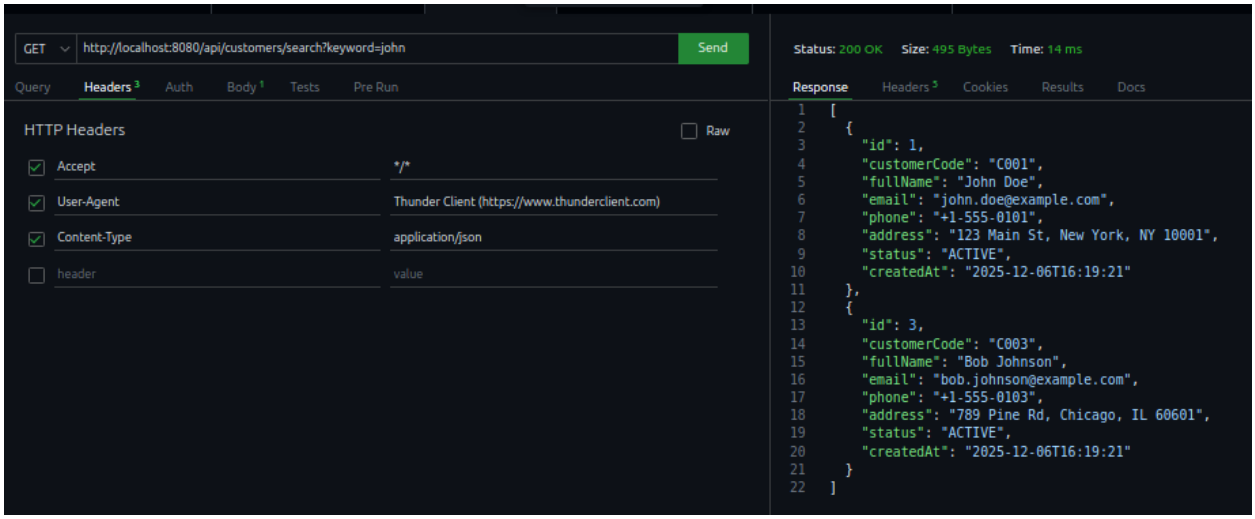
- Client sends GET request to controller with endpoint '/customers/search?keyword='.
- Controller maps the request to searchCustomers() and calls customerService.searchCustomers().

```java
// 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.

```java
@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.



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.

```java
@PostMapping
public ResponseEntity<CustomerResponseDTO> createCustomer(@Valid @RequestBody CustomerReque
```

- 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.

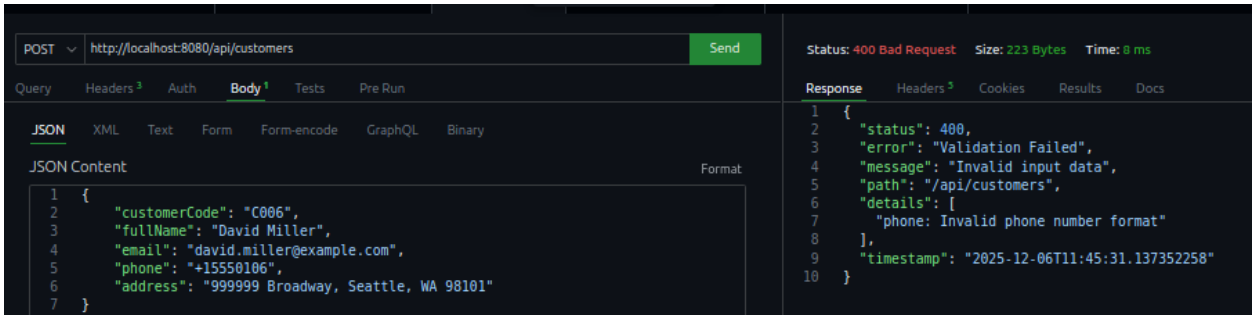
```java
// Handle Validation Errors (400)
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<ErrorResponseDTO> handleValidationException(
        MethodArgumentNotValidException ex,
        WebRequest request) {

    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.



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.

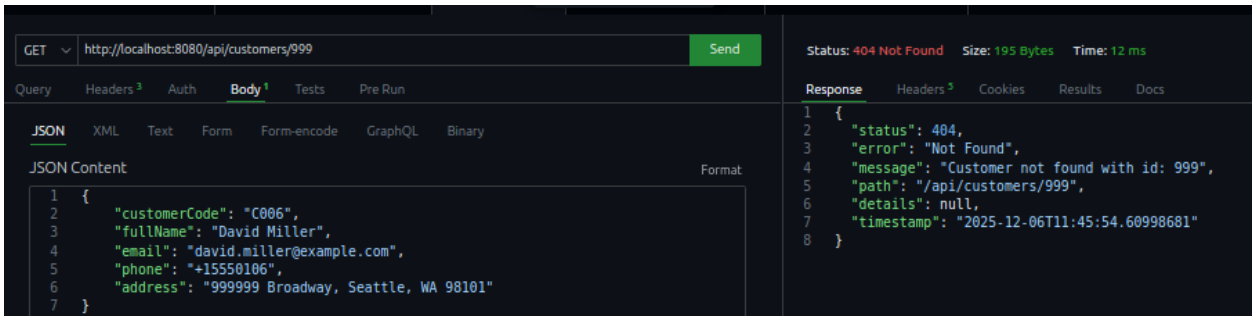
- GlobalExceptionHandler intercepts the exception and calls handleResourceNotFoundException().
- Handler creates an ErrorResponseDTO with status NOT_FOUND.
- Handler returns the ErrorResponseDTO to Controller.

```java
// 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.



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.

```java
// 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.