<https://github.com/eazybytes/microservices/tree/3.2.3>

**Step 12/13 : Spring data JPA entities and repository classes to interact with the tables**Configuring H2 DB  
Standard template for H2 database configuration for all services

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.h2.console.enabled=true

server.port=8080

package com.eazybytes.accounts.entity;

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

@Getter @Setter @ToString

public class BaseEntity {

    @CreatedDate

    @Column(updatable = false)

    private LocalDateTime createdAt;

    @CreatedBy

    @Column(updatable = false)

    private String createdBy;

    @LastModifiedDate

    @Column(insertable = false)

    private LocalDateTime updatedAt;

    @LastModifiedBy

    @Column(insertable = false)

    private String updatedBy;

}

package com.eazybytes.accounts.entity;

@Entity

@Getter @Setter @ToString @AllArgsConstructor @NoArgsConstructor

public class Customer extends  BaseEntity {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name="customer\_id")

    private Long customerId;

    private String name;

    private String email;

    @Column(name="mobile\_number")

    private String mobileNumber;

}

package com.eazybytes.accounts.entity;

@Entity

@Getter @Setter @ToString @AllArgsConstructor @NoArgsConstructor

public class Accounts extends  BaseEntity {

    @Column(name="customer\_id")

    private Long customerId;

    @Column(name="account\_number")

    @Id

    private Long accountNumber;

    @Column(name="account\_type")

    private String accountType;

    @Column(name="branch\_address")

    private String branchAddress;

}

package com.eazybytes.accounts.repository;

@Repository

public interface AccountsRepository extends JpaRepository<Accounts, Long> {

}

package com.eazybytes.accounts.repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

accounts\src\main\resources\schema.sql

CREATE TABLE IF NOT EXISTS `customer` (

  `customer\_id` int AUTO\_INCREMENT  PRIMARY KEY,

  `name` varchar(100) NOT NULL,

  `email` varchar(100) NOT NULL,

  `mobile\_number` varchar(20) NOT NULL,

  `created\_at` date NOT NULL,

  `created\_by` varchar(20) NOT NULL,

  `updated\_at` date DEFAULT NULL,

  `updated\_by` varchar(20) DEFAULT NULL

);

CREATE TABLE IF NOT EXISTS `accounts` (

  `customer\_id` int NOT NULL,

  `account\_number` int AUTO\_INCREMENT  PRIMARY KEY,

  `account\_type` varchar(100) NOT NULL,

  `branch\_address` varchar(200) NOT NULL,

  `created\_at` date NOT NULL,

  `created\_by` varchar(20) NOT NULL,

  `updated\_at` date DEFAULT NULL,

  `updated\_by` varchar(20) DEFAULT NULL

);

accounts\src\main\resources\data.sql

INSERT INTO customer (name,email,mobile\_number,created\_at,created\_by)

VALUES ('Eazy Bytes','tutor@eazybytes.com','9823150966',CURDATE(),'ADMIN');

INSERT INTO accounts (customer\_id,account\_number,account\_type,branch\_address,created\_at,created\_by)

VALUES (1,654321,'Savings','123, Main Street',CURDATE(),'ADMIN');

**Step 15 : Creating DTOs inside accounts microservice**

package com.eazybytes.accounts.dto;

@Data

public class AccountsDto {

    private Long accountNumber;

    private String accountType;

    private String branchAddress;

}

package com.eazybytes.accounts.dto;

@Data

public class CustomerDto {

    private String name;

    private String email;

    private String mobileNumber;

}

package com.eazybytes.accounts.dto;

@Data @AllArgsConstructor

public class ResponseDto {

    private String statusCode;

    private String statusMsg;

}

package com.eazybytes.accounts.dto;

@Data @AllArgsConstructor

public class ErrorResponseDto {

    private  String apiPath;

    private HttpStatus errorCode;

    private  String errorMessage;

    private LocalDateTime errorTime;

}

**Step 16 : CREATE API inside account microservice –   
Part 1 🡺 Add service layer and mapper classes for entity 🡨 🡪 dto**

package com.eazybytes.accounts.constants;

public final class AccountsConstants {

    private AccountsConstants() {

        // restrict instantiation

    }

    public static final String  SAVINGS = "Savings";

    public static final String  ADDRESS = "123 Main Street, New York";

    public static final String  STATUS\_201 = "201";

    public static final String  MESSAGE\_201 = "Account created successfully";

    public static final String  STATUS\_200 = "200";

    public static final String  MESSAGE\_200 = "Request processed successfully";

    public static final String  STATUS\_417 = "417";

    public static final String  MESSAGE\_417\_UPDATE= "Update operation failed. Please try again or contact Dev team";

    public static final String  MESSAGE\_417\_DELETE= "Delete operation failed. Please try again or contact Dev team";

    // public static final String  STATUS\_500 = "500";

    // public static final String  MESSAGE\_500 = "An error occurred. Please try again or contact Dev team";

}

package com.eazybytes.accounts.service;

public interface IAccountsService {

    void createAccount(CustomerDto customerDto);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountsService {

    private AccountRepository accountRepository;

    private CustomerRepository customerRepository;

    @Override

    public void createAccount(CustomerDto customerDto) {

    }

}

package com.eazybytes.accounts.mapper;

public class AccountsMapper {

    public static AccountsDto mapToAccountsDto(Accounts accounts, AccountsDto accountsDto) {

        accountsDto.setAccountNumber(accounts.getAccountNumber());

        accountsDto.setAccountType(accounts.getAccountType());

        accountsDto.setBranchAddress(accounts.getBranchAddress());

        return accountsDto;

    }

    public static Accounts mapToAccounts(AccountsDto accountsDto, Accounts accounts) {

        accounts.setAccountNumber(accountsDto.getAccountNumber());

        accounts.setAccountType(accountsDto.getAccountType());

        accounts.setBranchAddress(accountsDto.getBranchAddress());

        return accounts;

    }

}

package com.eazybytes.accounts.mapper;

public class CustomerMapper {

    public static CustomerDto mapToCustomerDto(Customer customer, CustomerDto customerDto) {

        customerDto.setName(customer.getName());

        customerDto.setEmail(customer.getEmail());

        customerDto.setMobileNumber(customer.getMobileNumber());

        return customerDto;

    }

    public static Customer mapToCustomer(CustomerDto customerDto, Customer customer) {

        customer.setName(customerDto.getName());

        customer.setEmail(customerDto.getEmail());

        customer.setMobileNumber(customerDto.getMobileNumber());

        return customer;

    }

}

**Step 17 : CREATE API inside account microservice  
Part 2 🡺 Add service layer and mapper classes for entity 🡨 🡪 dto**

**Note: Delete data.sql**

package com.eazybytes.accounts;

@SpringBootApplication

public class AccountsApplication {

    public static void main(String[] args) {

        SpringApplication.run(AccountsApplication.class, args);

    }

}

package com.eazybytes.accounts.mapper;

public class CustomerMapper {

    public static CustomerDto mapToCustomerDto(Customer customer, CustomerDto customerDto) {

        customerDto.setName(customer.getName());

        customerDto.setEmail(customer.getEmail());

        customerDto.setMobileNumber(customer.getMobileNumber());

        return customerDto;

    }

    public static Customer mapToCustomer(CustomerDto customerDto, Customer customer) {

        customer.setName(customerDto.getName());

        customer.setEmail(customerDto.getEmail());

        customer.setMobileNumber(customerDto.getMobileNumber());

**customer.setCreatedAt(LocalDateTime.now());**

**customer.setCreatedBy("ADMIN");**

        return customer;

    }

}

package com.eazybytes.accounts.repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long>{

    Optional<Customer> findByMobileNumber(String mobileNumber);

}

package com.eazybytes.accounts.exception;

@ResponseStatus(value = HttpStatus.BAD\_REQUEST)

public class CustomerAlreadyExistsException extends RuntimeException {

    public CustomerAlreadyExistsException(String message) {

        super(message);

    }

}

package com.eazybytes.accounts.exception;

@ControllerAdvice

public class GlobalExceptionHandler  extends ResponseEntityExceptionHandler {

    @ExceptionHandler(CustomerAlreadyExistsException.class)

    public ResponseEntity<ErrorResponseDto> handleCustomerAlreadyExistsException(CustomerAlreadyExistsException exception,

                                                                                 WebRequest webRequest){

        ErrorResponseDto errorResponseDTO = new ErrorResponseDto(

                webRequest.getDescription(false),

                HttpStatus.BAD\_REQUEST,

                exception.getMessage(),

                LocalDateTime.now()

        );

        return new ResponseEntity<>(errorResponseDTO, HttpStatus.BAD\_REQUEST);

    }

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountsService {

    private AccountRepository accountRepository;

    private CustomerRepository customerRepository;

    @Override

    public void createAccount(CustomerDto customerDto) {

        Customer customer = CustomerMapper.mapToCustomer(customerDto, new Customer());

        Optional<Customer> optionalCustomer = customerRepository.findByMobileNumber(customerDto.getMobileNumber());

        if(optionalCustomer.isPresent()) {

            throw new CustomerAlreadyExistsException("Customer already registered with given mobileNumber "

                    +customerDto.getMobileNumber());

        }

        Customer savedCustomer = customerRepository.save(customer);

        accountRepository.save(createNewAccount(savedCustomer));

    }

     private Accounts createNewAccount(Customer customer) {

        Accounts newAccount = new Accounts();

        newAccount.setCustomerId(customer.getCustomerId());

        long randomAccNumber = 1000000000L + new Random().nextInt(900000000);

        newAccount.setCreatedAt(LocalDateTime.now());

        newAccount.setCreatedBy("ADMIN");

        newAccount.setAccountNumber(randomAccNumber);

        newAccount.setAccountType(AccountsConstants.SAVINGS);

        newAccount.setBranchAddress(AccountsConstants.ADDRESS);

        return newAccount;

    }

}

package com.eazybytes.accounts.controller;

@RestController

@AllArgsConstructor

@RequestMapping(path = "/api", produces = "application/json")

public class AccountsController {

    private IAccountsService iAccountsService;

    @PostMapping("/create")

    public ResponseEntity<ResponseDto> createAccount(@RequestBody CustomerDto customerDto) {

         iAccountsService.createAccount(customerDto);

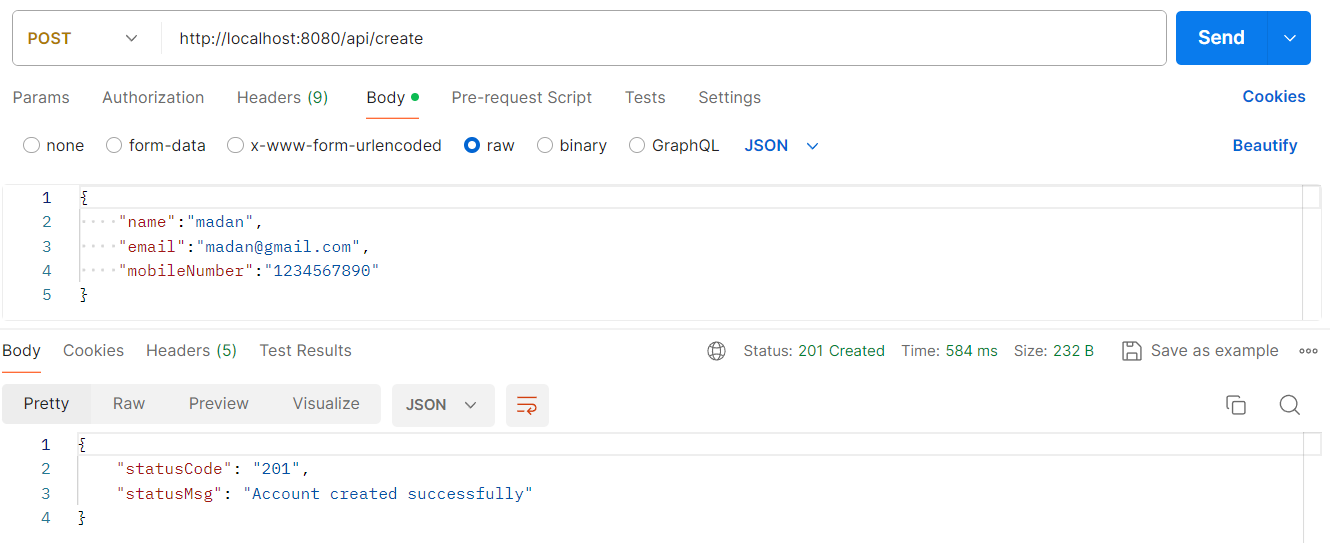
        return ResponseEntity

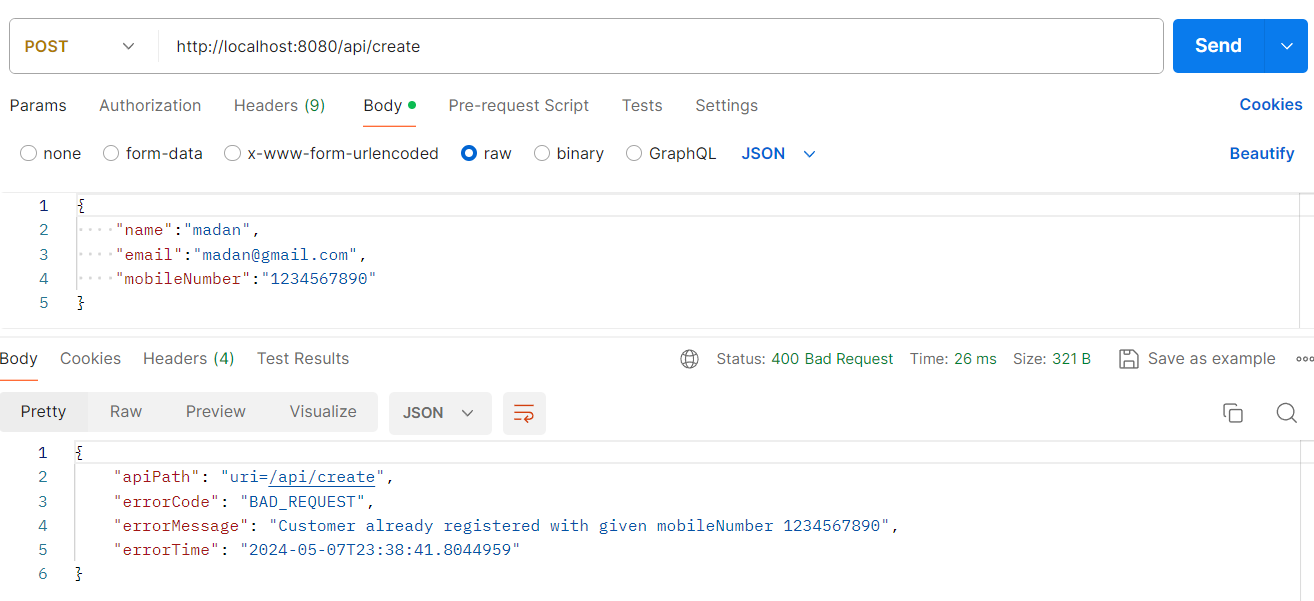
               .status(HttpStatus.CREATED)

               .body(new ResponseDto(AccountsConstants.STATUS\_201, AccountsConstants.MESSAGE\_201));

    }

}





**Step 18 : READ API inside account microservice –**package com.eazybytes.accounts.exception;

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException{

    public ResourceNotFoundException(String resourceName, String fieldName, String fieldValue) {

        super(String.format("%s not found with the given input data %s : %s", resourceName, fieldName, fieldValue));

    }

}

package com.eazybytes.accounts.exception;

@ControllerAdvice

public class GlobalExceptionHandler  extends ResponseEntityExceptionHandler {

\*\*\*

    @ExceptionHandler(ResourceNotFoundException.class)

    public ResponseEntity<ErrorResponseDto> handleResourceNotFoundException(ResourceNotFoundException exception,

                                                                                 WebRequest webRequest){

        ErrorResponseDto errorResponseDTO = new ErrorResponseDto(

                webRequest.getDescription(false),

                HttpStatus.NOT\_FOUND,

                exception.getMessage(),

                LocalDateTime.now()

        );

        return new ResponseEntity<>(errorResponseDTO, HttpStatus.NOT\_FOUND);

    }

}

package com.eazybytes.accounts.service;

public interface IAccountsService {

    void createAccount(CustomerDto customerDto);

    CustomerDto fetchAccount(String mobileNumber);

}

package com.eazybytes.accounts.repository;

@Repository

public interface AccountRepository extends JpaRepository<Accounts, Long>{

    Optional<Accounts> findByCustomerId(Long customerId);

}

package com.eazybytes.accounts.dto;

@Data

public class CustomerDto {

    private String name;

    private String email;

    private String mobileNumber;

    private AccountsDto accountsDto;

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountsService {

    private AccountRepository accountRepository;

    private CustomerRepository customerRepository;

\*\*\*

    @Override

    public CustomerDto fetchAccount(String mobileNumber) {

        Customer customer = customerRepository.findByMobileNumber(mobileNumber).orElseThrow(

            () -> new ResourceNotFoundException("Customer", "mobileNumber", mobileNumber)

        );

        Accounts accounts = accountRepository.findByCustomerId(customer.getCustomerId()).orElseThrow(

            () -> new ResourceNotFoundException("Account", "customerId", customer.getCustomerId().toString())

        );

        CustomerDto customerDto = CustomerMapper.mapToCustomerDto(customer, new CustomerDto());

        customerDto.setAccountsDto(AccountsMapper.mapToAccountsDto(accounts, new AccountsDto()));

        return customerDto;

    }

}

package com.eazybytes.accounts.controller;

@RestController

@AllArgsConstructor

@RequestMapping(path = "/api", produces = "application/json")

public class AccountsController {

    private IAccountsService iAccountsService;

\*\*\*

    @GetMapping("/fetch")

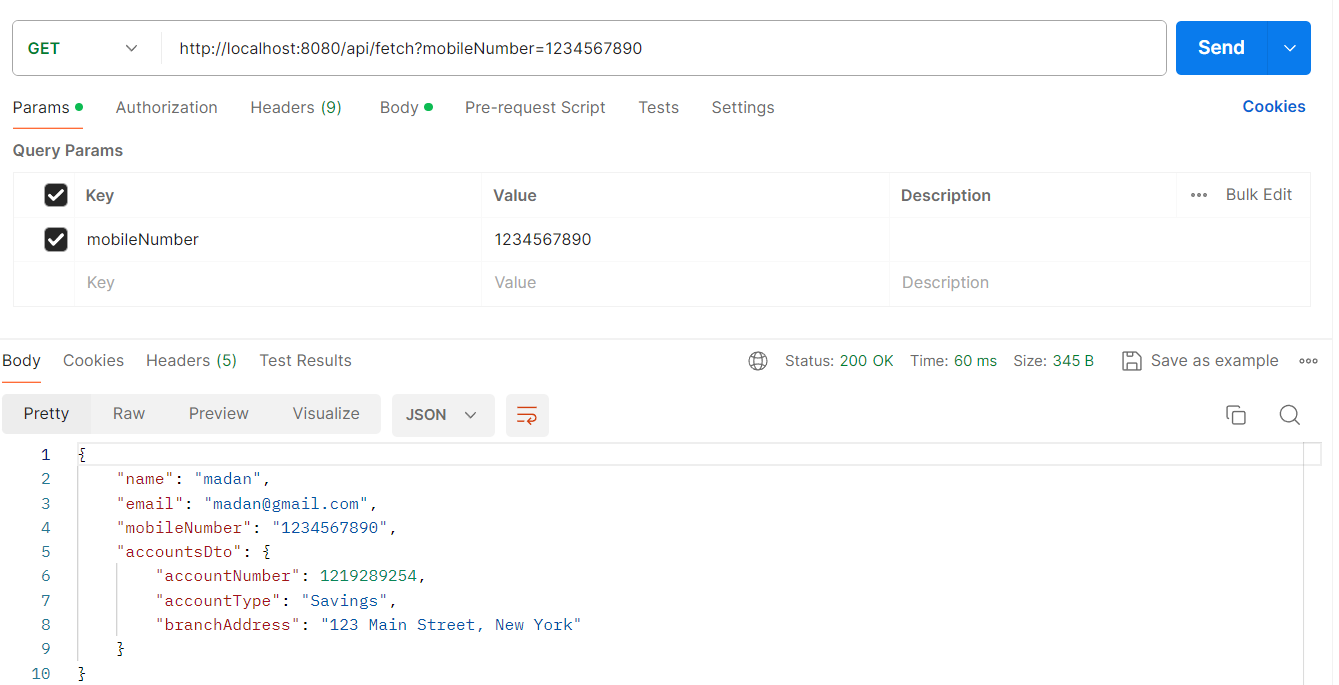
    public ResponseEntity<CustomerDto> fetchAccountDetails(@RequestParam String mobileNumber) {

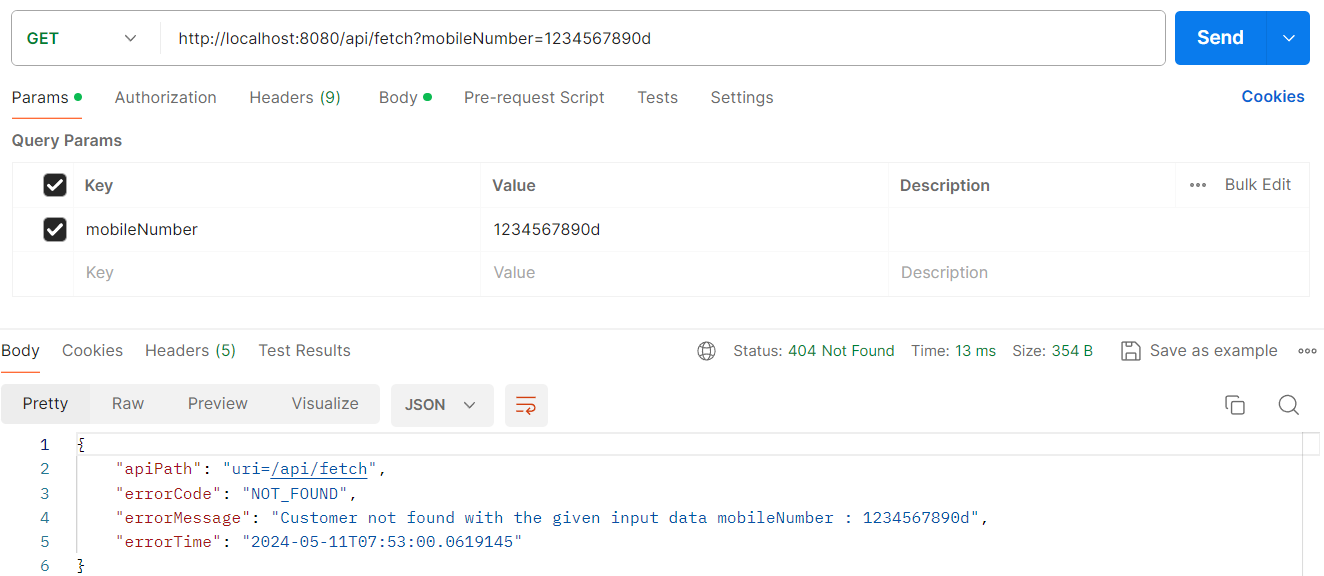
        CustomerDto customerDto = iAccountsService.fetchAccount(mobileNumber);

        return ResponseEntity.status(HttpStatus.OK).body(customerDto);

    }

}





**Step 19 : UPDATE API inside account microservice**

package com.eazybytes.accounts.service;

public interface IAccountsService {

    void createAccount(CustomerDto customerDto);

    CustomerDto fetchAccount(String mobileNumber);

    boolean updateAccount(CustomerDto customerDto);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountsService {

    private AccountRepository accountRepository;

    private CustomerRepository customerRepository;

   \*\*\*

    @Override

    public boolean updateAccount(CustomerDto customerDto) {

        boolean isUpdated = false;

        AccountsDto accountsDto = customerDto.getAccountsDto();

        if(accountsDto != null) {

            Accounts accounts = accountRepository.findById(accountsDto.getAccountNumber()).orElseThrow(

                () -> new ResourceNotFoundException("Account", "AccountNumber", String.valueOf(accountsDto.getAccountNumber()))

            );

            AccountsMapper.mapToAccounts(accountsDto, accounts);

            accounts = accountRepository.save(accounts);

            Long customerId = accounts.getCustomerId();

            Customer customer = customerRepository.findById(customerId).orElseThrow(

                () -> new ResourceNotFoundException("Customer", "CustomerId", String.valueOf(customerId))

            );

            CustomerMapper.mapToCustomer(customerDto, customer);

            customerRepository.save(customer);

            isUpdated = true;

        }

        return isUpdated;

    }

}

package com.eazybytes.accounts.controller;

@RestController

@AllArgsConstructor

@RequestMapping(path = "/api", produces = "application/json")

public class AccountsController {

    private IAccountsService iAccountsService;

   \*\*\*

    @PutMapping("/update")

    public ResponseEntity<ResponseDto> updateAccount(@RequestBody CustomerDto customerDto) {

        boolean isUpdated = iAccountsService.updateAccount(customerDto);

        if(isUpdated) {

            return ResponseEntity

                    .status(HttpStatus.OK)

                    .body(new ResponseDto(AccountsConstants.STATUS\_200, AccountsConstants.MESSAGE\_200));

        } else {}

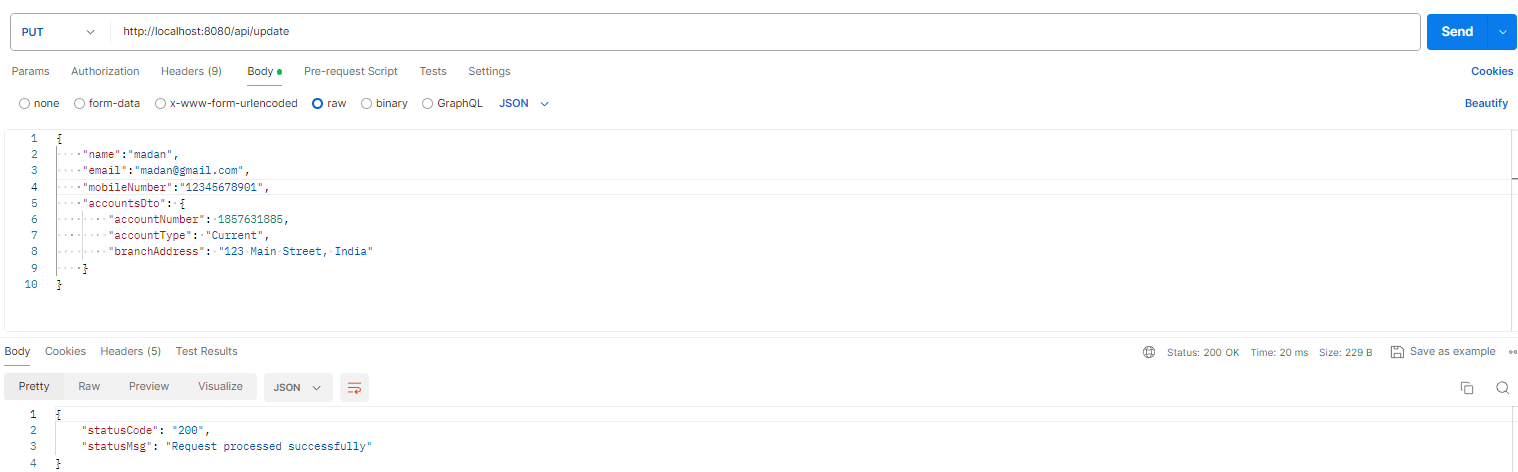
        return ResponseEntity

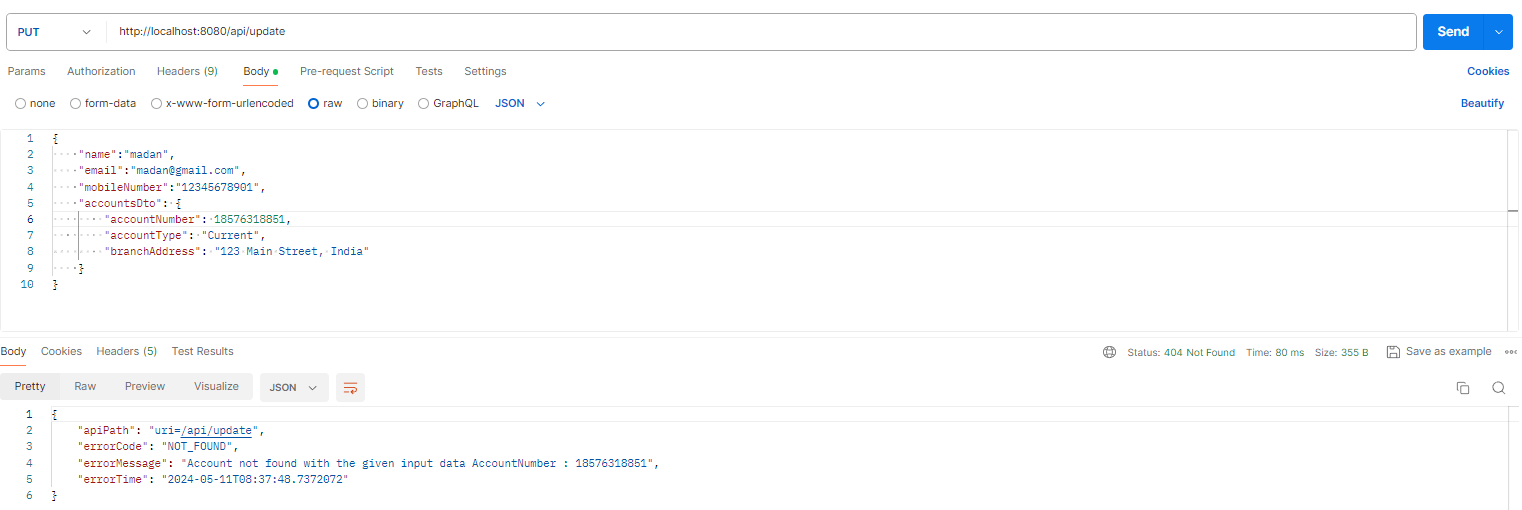
                .status(HttpStatus.EXPECTATION\_FAILED)

                .body(new ResponseDto(AccountsConstants.STATUS\_417, AccountsConstants.MESSAGE\_417\_UPDATE));

        }

}





**Step 20 : DELETE API inside account microservice**

package com.eazybytes.accounts.service;

public interface IAccountsService {

    void createAccount(CustomerDto customerDto);

    CustomerDto fetchAccount(String mobileNumber);

    boolean updateAccount(CustomerDto customerDto);

    boolean deleteAccount(String mobileNumber);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountsService {

    private AccountRepository accountRepository;

    private CustomerRepository customerRepository;

    \*\*\*

    @Override

    public boolean deleteAccount(String mobileNumber) {

        Customer customer = customerRepository.findByMobileNumber(mobileNumber).orElseThrow(

            () -> new ResourceNotFoundException("Customer", "mobileNumber", mobileNumber)

        );

        accountRepository.deleteByCustomerId(customer.getCustomerId());

        customerRepository.deleteById(customer.getCustomerId());

        return true;

    }

}

package com.eazybytes.accounts.repository;

@Repository

public interface AccountRepository extends JpaRepository<Accounts, Long>{

    Optional<Accounts> findByCustomerId(Long customerId);

    @Transactional

    @Modifying

    void deleteByCustomerId(Long customerId);

}

package com.eazybytes.accounts.controller;

@RestController

@AllArgsConstructor

@RequestMapping(path = "/api", produces = "application/json")

public class AccountsController {

    private IAccountsService iAccountsService;

  \*\*\*

    @DeleteMapping("/delete")

    public ResponseEntity<ResponseDto> deleteAccount(@RequestParam String mobileNumber) {

        boolean isDeleted = iAccountsService.deleteAccount(mobileNumber);

        if(isDeleted) {

            return ResponseEntity

                    .status(HttpStatus.OK)

                    .body(new ResponseDto(AccountsConstants.STATUS\_200, AccountsConstants.MESSAGE\_200));

        } else {

        return ResponseEntity

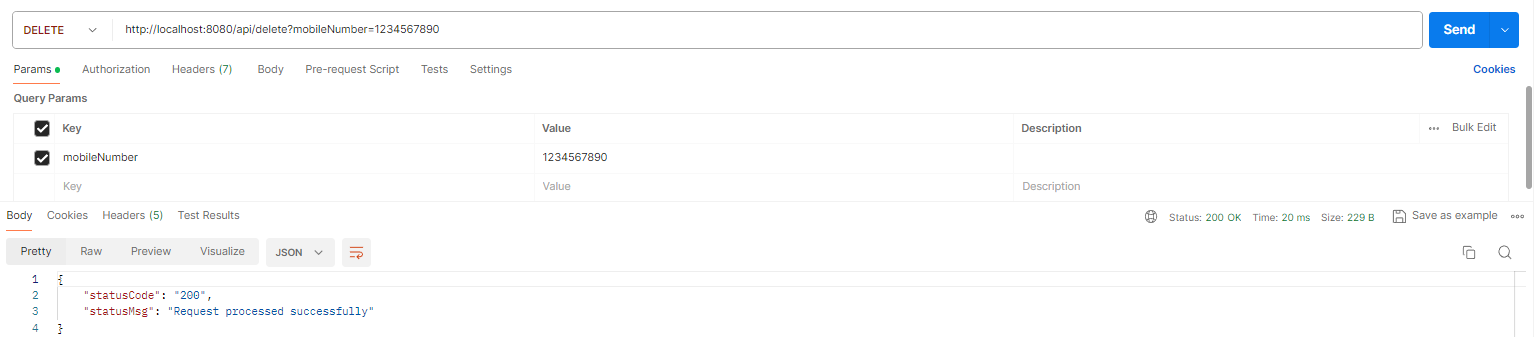
                .status(HttpStatus.EXPECTATION\_FAILED)

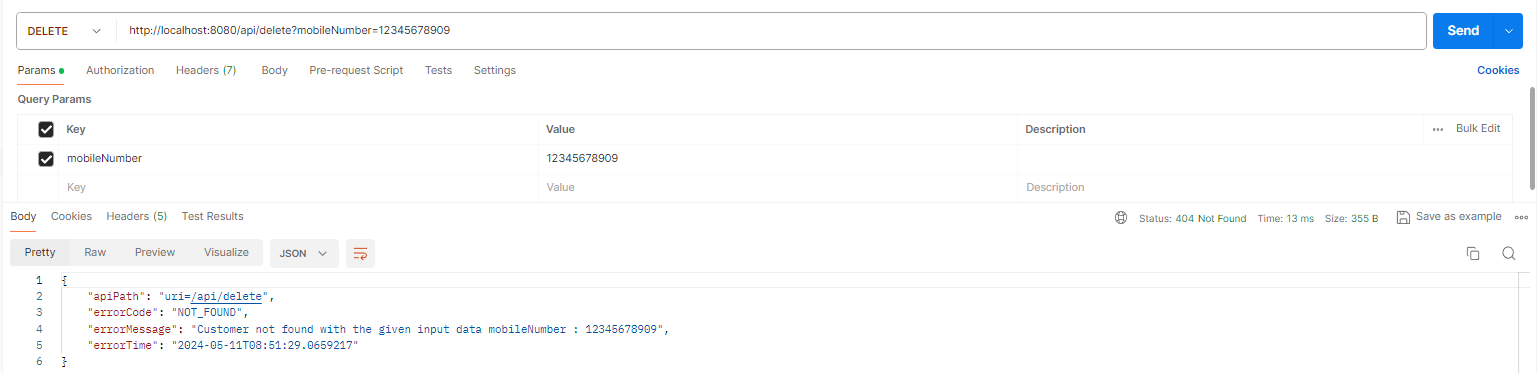
                .body(new ResponseDto(AccountsConstants.STATUS\_417, AccountsConstants.MESSAGE\_417\_DELETE));

        }

    }

}





**Step 21 : Handle all types of runtime exceptions using global logic inside accounts**

package com.eazybytes.accounts.exception;

@ControllerAdvice

public class GlobalExceptionHandler  extends ResponseEntityExceptionHandler {

    @ExceptionHandler(RuntimeException.class)

    public ResponseEntity<ErrorResponseDto> handleGlobalException(RuntimeException exception,

                                                                                 WebRequest webRequest){

        ErrorResponseDto errorResponseDTO = new ErrorResponseDto(

                webRequest.getDescription(false),

                HttpStatus.INTERNAL\_SERVER\_ERROR,

                exception.getMessage(),

                LocalDateTime.now()

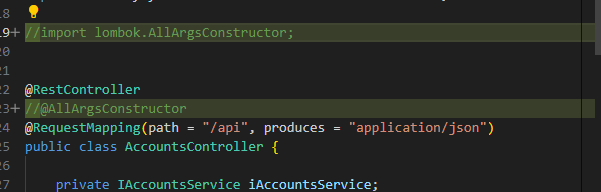
        );

        return new ResponseEntity<>(errorResponseDTO, HttpStatus.INTERNAL\_SERVER\_ERROR);

    }

    \*\*\*

}

To test the global exception handler change the controller as below   




**Step 22 : Perform validation on the input data inside accounts microservice**

Required dependency:  
<dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-validation</artifactId>

</dependency>

package com.eazybytes.accounts.dto;

@Data

public class CustomerDto {

    @NotEmpty(message = "Name cannot be null or empty")

    @Size(min = 5, max = 30, message = "The length of the customer nam eshould be between 5 and 30 characters")

    private String name;

    @NotEmpty(message = "Email cannot be null or empty")

    @Email(message = "Email address should be valid value")

    private String email;

    @Pattern(regexp = "($|[0-9]{10})", message = "Mobile number should be 10 digits value")

    private String mobileNumber;

    private AccountsDto accountsDto;

}

package com.eazybytes.accounts.dto;

@Data

public class AccountsDto {

    @NotEmpty(message = "Account number cannot be null or empty")

    @Pattern(regexp = "([0-9]{10})", message = "Account number should be 10 digits value")

    private Long accountNumber;

    @NotEmpty(message = "Account type cannot be null or empty")

    private String accountType;

    @NotEmpty(message = "Branch address cannot be null or empty")

    private String branchAddress;

}

package com.eazybytes.accounts.controller;

import org.springframework.validation.annotation.Validated;

import jakarta.validation.Valid;

import jakarta.validation.constraints.Pattern;

@RestController

@AllArgsConstructor

@RequestMapping(path = "/api", produces = "application/json")

**@Validated**

public class AccountsController {

    private IAccountsService iAccountsService;

    @PostMapping("/create")

    public ResponseEntity<ResponseDto> createAccount(**@Valid** @RequestBody CustomerDto customerDto) {

          \*\*\*

    }

    @GetMapping("/fetch")

    public ResponseEntity<CustomerDto> fetchAccountDetails(@RequestParam

**@Pattern(regexp = "($|[0-9]{10})", message = "Mobile number should be 10 digits value")**

                                                           String mobileNumber) {

\*\*\*

    }

    @PutMapping("/update")

    public ResponseEntity<ResponseDto> updateAccount(**@Valid** @RequestBody CustomerDto customerDto) {

        \*\*\*

    }

    @DeleteMapping("/delete")

    public ResponseEntity<ResponseDto> deleteAccount(@RequestParam

**@Pattern(regexp = "($|[0-9]{10})", message = "Mobile number should be 10 digits value")**

                                                     String mobileNumber) {

        \*\*\*

        }

    }

}

package com.eazybytes.accounts.exception;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatusCode;

import org.springframework.validation.FieldError;

import org.springframework.validation.ObjectError;

import org.springframework.web.bind.MethodArgumentNotValidException;

import java.util.HashMap;

import java.util.Map;

import java.util.List;

@ControllerAdvice

public class GlobalExceptionHandler  extends ResponseEntityExceptionHandler {

    @Override

    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,

                        HttpHeaders headers, HttpStatusCode status, WebRequest request) {

        Map<String, String> validationErrors = new HashMap<>();

        List<ObjectError> validationErrorList = ex.getBindingResult().getAllErrors();

        validationErrorList.forEach((error) -> {

            String fieldName = ((FieldError) error).getField();

            String validationMsg = error.getDefaultMessage();

            validationErrors.put(fieldName, validationMsg);

        });

        return new ResponseEntity<>(validationErrors, HttpStatus.BAD\_REQUEST);

    }

    @ExceptionHandler(RuntimeException.class)

    public ResponseEntity<ErrorResponseDto> handleGlobalException(RuntimeException exception,

                                                                                 WebRequest webRequest){

        ErrorResponseDto errorResponseDTO = new ErrorResponseDto(

                webRequest.getDescription(false),

                HttpStatus.INTERNAL\_SERVER\_ERROR,

                exception.getMessage(),

                LocalDateTime.now()

        );

        return new ResponseEntity<>(errorResponseDTO, HttpStatus.INTERNAL\_SERVER\_ERROR);

    }

\*\*\*

}



**Step 23 : Update audit columns using Spring Data**

package com.eazybytes.accounts.entity;

@MappedSuperclass

**@EntityListeners(AuditingEntityListener.class)**

@Getter @Setter @ToString

public class BaseEntity {

**@CreatedDate**

    @Column(updatable = false)

    private LocalDateTime createdAt;

**@CreatedBy**

    @Column(updatable = false)

    private String createdBy;

**@LastModifiedDate**

    @Column(insertable = false)

    private LocalDateTime updatedAt;

**@LastModifiedBy**

    @Column(insertable = false)

    private String updatedBy;

}

package com.eazybytes.accounts;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing(auditorAwareRef = "auditAwareImpl")

public class AccountsApplication {

    public static void main(String[] args) {

        SpringApplication.run(AccountsApplication.class, args);

    }

}

package com.eazybytes.accounts.audit;

import org.springframework.data.domain.AuditorAware;

import org.springframework.stereotype.Component;

@Component("auditAwareImpl")

public class AuditAwareImpl implements AuditorAware<String> {

    @Override

    public Optional<String> getCurrentAuditor() {

        return Optional.of("ACCOUNTS\_MS");

    }

}

Execute Create and Update endpoints:   
