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

**Step 1 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

package com.eazybytes.accounts.model;

@MappedSuperclass

@Setter @Getter @ToString

public class BaseEntity {

    @Column(updatable = false)

    private LocalDateTime createdAt;

    @Column(insertable = false)

    private String createdBy;

    @Column(updatable = false)

    private LocalDateTime updatedAt;

    @Column(insertable = false)

    private String updatedBy;

}

package com.eazybytes.accounts.model;

@Entity

@Getter @Setter @ToString @AllArgsConstructor @NoArgsConstructor

public class Customer extends BaseEntity{

    @Column(name = "customer\_id")

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private int customerId;

    @Column(name = "name")

    private String name;

    @Column(name = "email")

    private String email;

    @Column(name = "mobile\_number")

    private String mobileNumber;

}

package com.eazybytes.accounts.model;

@Entity

@Getter @Setter @ToString @AllArgsConstructor @NoArgsConstructor

public class Accounts extends BaseEntity{

    @Column(name = "account\_number")

    @Id

    private long accountNumber;

    @Column(name = "customer\_id")

    private int customerId;

    @Column(name = "account\_type")

    private String accountType;

    @Column(name = "branch\_address")

    private String branchAddress;

}

package com.eazybytes.accounts.repository;

@Repository

public interface AccountsRepository extends CrudRepository<Accounts, Long>{

}

package com.eazybytes.accounts.repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Integer> {

   // There are many method in this interface

}

**Step 2 – Create DTO classes**

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 statusMessage;

}

package com.eazybytes.accounts.dto;

@Data @AllArgsConstructor

public class ErrorResponseDto {

    private String apiPath;

    private HttpStatus errorCode;

    private String errorMessage;

    private LocalDateTime errorTime;

}

**Step 3 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.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(CustomerDto 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;

    }

}

package com.eazybytes.accounts.service;

public interface IAccountsService {

    void createAccount(CustomerDto customerDto);

}

package com.eazybytes.accounts.service.impl;

@Service

public class AccountsServiceImpl implements IAccountsService {

    @Autowired

    private AccountsRepository accountsRepository;

    @Autowired

    private CustomerRepository customerRepository;

    @Override

    public void createAccount(CustomerDto customerDto) {

        throw new UnsupportedOperationException("Unimplemented method 'createAccount'");

    }

}

package com.eazybytes.accounts.controller;

@RestController

@RequestMapping(path = "/api", produces = MediaType.APPLICATION\_JSON\_VALUE)

public class AccountsController {

    @PostMapping("/create")

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

        return ResponseEntity

                .status(HttpStatus.CREATED)

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

    }

}

**Step 4 CREATE API inside account microservice –   
Part 2 🡺 Add service layer and mapper classes for entity 🡨 🡪 dto**

@SpringBootApplication

@ComponentScans({@ComponentScan("com.eazybytes.accounts.controller,com.**eazybytes.accounts.service"**)})

@EnableJpaRepositories("com.eazybytes.accounts.repository")

@EntityScan("com.eazybytes.accounts.model")

public class AccountsApplication {

    public static void main(String[] args) {

        SpringApplication.run(AccountsApplication.class, args);

    }

}

package com.eazybytes.accounts.repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Integer> {

    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 {

    @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

public class AccountsServiceImpl implements IAccountsService {

    @Autowired

    private AccountsRepository accountsRepository;

    @Autowired

    private CustomerRepository customerRepository;

    @Override

    public void createAccount(CustomerDto customerDto) {

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

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

        if(optionalCustomer.isPresent()){

           throw new CustomerAlreadyExistsException("Customer already exists with mobile number " +   
 customerDto.getMobileNumber());

        }

        Customer savedCustomer = customerRepository.save(customer);

        accountsRepository.save(createNewAccount(savedCustomer));

    }

    private Accounts createNewAccount(Customer customer){

        Accounts newAccount = new Accounts();

        newAccount.setCustomerId(customer.getCustomerId());

        long randomAccNumber = ThreadLocalRandom.current().nextLong(100000000, 999999999+1);

        newAccount.setAccountNumber(randomAccNumber);

        newAccount.setAccountType(AccountsConstants.SAVINGS);

        newAccount.setBranchAddress(AccountsConstants.ADDRESS);

        return newAccount;

    }

}

package com.eazybytes.accounts.controller;

@RestController

@RequestMapping(path = "/api", produces = MediaType.APPLICATION\_JSON\_VALUE)

public class AccountsController {

    @Autowired

    private IAccountsService iAccountsService;

    @PostMapping("/create")

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

        iAccountsService.createAccount(customerDto);

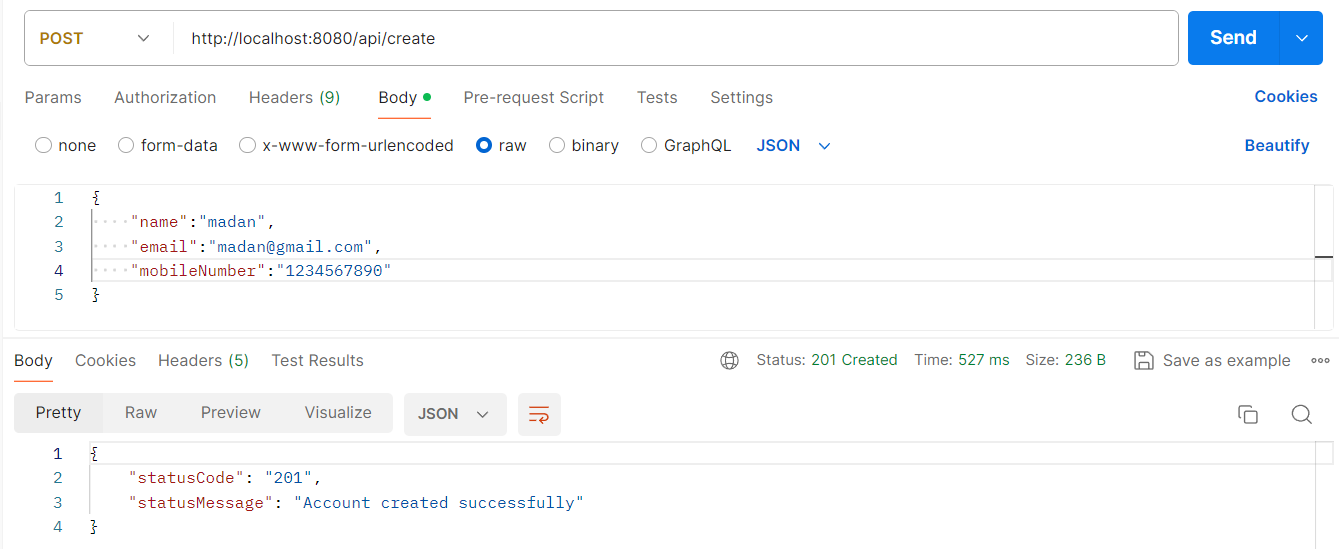
        return ResponseEntity

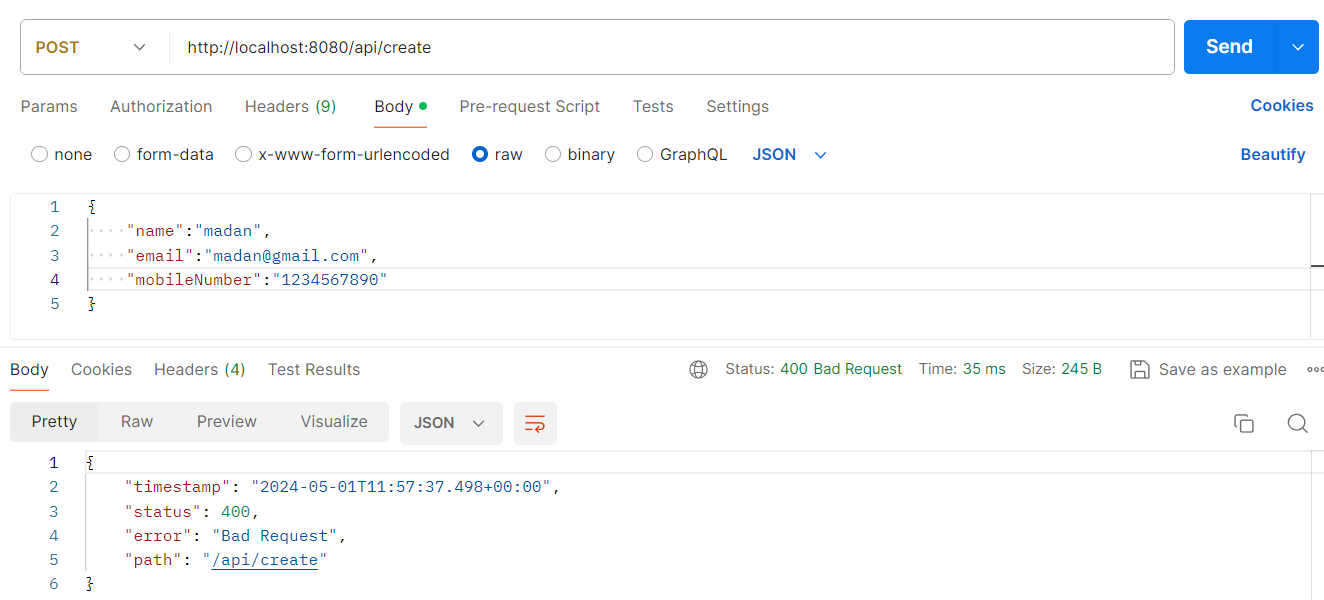
                .status(HttpStatus.CREATED)

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

    }

}





**Step 5 ???**