**Spring Boot Docker Kubernetes – Building Microservices with Spring Boot**

**Step- 1 – Creating Spring Boot Project** <dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

package com.eazybytes.accounts;

@SpringBootApplication

public class AccountsApplication {

public static void main(String[] args) {

SpringApplication.run(AccountsApplication.class, args);

}

}

**Step – 2 – Hello World Rest Controller**

package com.eazybytes.accounts.controller;

@RestController

public class AccountsController {

@GetMapping("sayHello")

public String sayHello(){

return "Hello World!";

}

}

GET <http://localhost:8080/sayHello>  
  
**Step – 3 – Configuring H2 DB and YAML properties**server:

port: 8080

spring:

datasource:

url: jdbc:h2:mem:testdb

driverClassName: org.h2.Driver

username: sa

password: ''

h2:

console:

enabled: true

jpa:

database-platform: org.hibernate.dialect.H2Dialect

hibernate:

ddl-auto: update

show-sql: true

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

);

<http://localhost:8080/h2-console>

**Step – 4 - Spring Data JPA entities, and repositories to interact with DB**

package com.eazybytes.accounts.entity;

import jakarta.persistence.Column;

import jakarta.persistence.MappedSuperclass;

@MappedSuperclass

@Getter @Setter @ToString

public class BaseEntity {

@Column(updatable = false)

private LocalDateTime createdAt;

@Column(updatable = false)

private String createdBy;

@Column(insertable = false)

private LocalDateTime updatedAt;

@Column(insertable = false)

private String updatedBy;

}

package com.eazybytes.accounts.entity;

import jakarta.persistence.\*;

import org.hibernate.annotations.GenericGenerator;

@Entity

@Getter @Setter @ToString @AllArgsConstructor @NoArgsConstructor

public class Customer extends BaseEntity {

@Id

@GeneratedValue(strategy= GenerationType.AUTO,generator="native")

@GenericGenerator(name = "native",strategy = "native")

@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

// There will be a logic to generate 10 digit the account number, since accounts numbers cannot be 1,2,3 etc

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> {

}

**Step – 5 – Creating DTO’s**

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;

private AccountsDto accountsDto;

}

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 – 6 – CREATE API – PART 1**

package com.eazybytes.accounts.service;

public interface IAccountService {

void createAccount(CustomerDto customerDto);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountService {

//No need to @Autowired since there is only single constructor accepting parameters in the class

private AccountsRepository accountsRepository;

private CustomerRepository customerRepository;

@Override

public void createAccount(CustomerDto customerDto) {

}

}

package com.eazybytes.accounts.constants;

public 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 {

// java libraries modelmapper and mapStruct

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 {

**// java libraries modelmapper and mapStruct**

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;

}

}

package com.eazybytes.accounts.controller;

@RestController

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

public class AccountsController {

@GetMapping("sayHello")

public String sayHello(){

return "Hello World!";

}

@PostMapping("/create")

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

return ResponseEntity

.status(HttpStatus.CREATED)

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

}

}

**Step – 7 – CREATE API – PART 2**

package com.eazybytes.accounts.service;

public interface IAccountService {

void createAccount(CustomerDto customerDto);

}

package com.eazybytes.accounts.repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

Optional<Customer> findByMobileNumber(String mobileNumber);

}

package com.eazybytes.accounts.exception;

import com.eazybytes.accounts.dto.ErrorResponseDto;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.context.request.WebRequest;

@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.exception;

@ResponseStatus(value = HttpStatus.BAD\_REQUEST)

public class CustomerAlreadyExistsException extends RuntimeException {

public CustomerAlreadyExistsException(String message) {

super(message);

}

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountService {

//No need to @Autowired since there is only single constructor accepting parameters in the class

private AccountsRepository accountsRepository;

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.setCreatedAt(LocalDateTime.now());

customer.setCreatedBy("Anonymous");

Customer savedCustomer = customerRepository.save(customer);

accountsRepository.save(createNewAccount(savedCustomer));

}

private Accounts createNewAccount(Customer customer) {

Accounts newAccount = new Accounts();

newAccount.setCustomerId(customer.getCustomerId());

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

newAccount.setAccountNumber(randomAccNumber);

newAccount.setAccountType(AccountsConstants.SAVINGS);

newAccount.setBranchAddress(AccountsConstants.ADDRESS);

newAccount.setCreatedAt(LocalDateTime.now());

newAccount.setCreatedBy("Anonymous");

return newAccount;

}

}

package com.eazybytes.accounts.controller;

@RestController

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

@AllArgsConstructor

public class AccountsController {

private IAccountService iAccountsService;

@GetMapping("sayHello")

public String sayHello(){

return "Hello World!";

}

@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));

}

}

POST <http://localhost:8080/api/create>  
{

    "name": "Madan Reddy",

    "email": "tutor@eazybytes",

    "mobileNumber": "4354437687"

}  
  
Again, execute the same request to check the exception message

**Step – 8 – READ API**

package com.eazybytes.accounts.repository;

@Repository

public interface AccountsRepository extends JpaRepository<Accounts, Long> {

Optional<Accounts> findByCustomerId(Long customerId);

}  
  
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.service;

public interface IAccountService {

CustomerDto fetchAccount(String mobileNumber);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountService {

//No need to @Autowired since there is only single constructor accepting parameters in the class

private AccountsRepository accountsRepository;

private CustomerRepository customerRepository;

@Override

public CustomerDto fetchAccount(String mobileNumber) {

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

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

);

Accounts accounts = accountsRepository.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.exception;

@ControllerAdvice

public class GlobalExceptionHandler {

@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.controller;

@RestController

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

@AllArgsConstructor

public class AccountsController {

private IAccountService iAccountsService;

@GetMapping("/fetch")

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

CustomerDto customerDto = iAccountsService.fetchAccount(mobileNumber);

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

}

}  
  
GET <http://localhost:8080/api/fetch?mobileNumber=43544376872>

Test with valid and invalid phone number

**Step – 9 – UPDATE AND DELETE API**

package com.eazybytes.accounts.repository;

@Repository

public interface AccountsRepository extends JpaRepository<Accounts, Long> {

@Transactional

@Modifying

void deleteByCustomerId(Long customerId);

}

package com.eazybytes.accounts.service;

public interface IAccountService {

boolean updateAccount(CustomerDto customerDto);

boolean deleteAccount(String mobileNumber);

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountService {

@Override

public boolean updateAccount(CustomerDto customerDto) {

boolean isUpdated = false;

AccountsDto accountsDto = customerDto.getAccountsDto();

if(accountsDto !=null ){

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

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

);

AccountsMapper.mapToAccounts(accountsDto, accounts);

accounts = accountsRepository.save(accounts);

Long customerId = accounts.getCustomerId();

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

() -> new ResourceNotFoundException("Customer", "CustomerID", customerId.toString())

);

CustomerMapper.mapToCustomer(customerDto,customer);

customerRepository.save(customer);

isUpdated = true;

}

return isUpdated;

}

@Override

public boolean deleteAccount(String mobileNumber) {

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

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

);

accountsRepository.deleteByCustomerId(customer.getCustomerId());

customerRepository.deleteById(customer.getCustomerId());

return true;

}

}

package com.eazybytes.accounts.controller;

@RestController

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

@AllArgsConstructor

public class AccountsController {

@PutMapping("/update")

public ResponseEntity<ResponseDto> updateAccountDetails(@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));

}

}

@DeleteMapping("/delete")

public ResponseEntity<ResponseDto> deleteAccountDetails(@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));

}

}

}

POST <http://localhost:8080/api/create>  
{

    "name": "Madan Reddy",

    "email": "tutor@eazybytes",

    "mobileNumber": "4354437687"

}

GET <http://localhost:8080/api/fetch?mobileNumber=4354437687>  
  
Except **account number** all can be changed  
PUT <http://localhost:8080/api/update>

{

    "name": "Madan Mohan",

    "email": "tutorial@eazybytes",

    "mobileNumber": "43544376871",

    "accountsDto": {

        "accountNumber": 1313869940,

        "accountType": "Savings",

        "branchAddress": "1234 Main Street, New York"

    }

}

DELETE <http://localhost:8080/api/delete?mobileNumber=43544376871>

**Step – 10 – Handle all types of runtime exceptions using global logic inside accounts**package com.eazybytes.accounts.exception;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

public ResponseEntity<ErrorResponseDto> handleGlobalException(Exception 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 this, delete the @AllArgsConstructor from AccountsController.java

~~@AllArgsConstructor~~

public class AccountsController {

private IAccountService iAccountsService;

…  
}  
  
now execute the createAccount endpoint in postman.  
  
POST <http://localhost:8080/api/create>  
{

    "name": "Madan Reddy",

    "email": "tutor@eazybytes",

    "mobileNumber": "4354437687"

}

**Step – 11 – Perform input data validations inside accounts MS**<dependency>

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

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

</dependency>  
This library has all the annotations to do the validation.  
  
package com.eazybytes.accounts.dto;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotEmpty;

import jakarta.validation.constraints.Pattern;

import jakarta.validation.constraints.Size;

@Data

public class CustomerDto {

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

@Size(min = 5, max = 30, message = "The length of the customer name should be between 3 and 30")

private String name;

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

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

private String email;

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

private String mobileNumber;

private AccountsDto accountsDto;

}

package com.eazybytes.accounts.dto;

@Data

public class AccountsDto {

@NotEmpty(message = "AccountNumber can not be a null or empty")

@Pattern(regexp="(^$|[0-9]{10})",message = "AccountNumber must be 10 digits")

private Long accountNumber;

@NotEmpty(message = "AccountType can not be a null or empty")

private String accountType;

@NotEmpty(message = "BranchAddress can not be a 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

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

@AllArgsConstructor

@Validated

public class AccountsController {

@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 must be 10 digits")

String mobileNumber) {

…

}

@PutMapping("/update")

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

…

}

}

@DeleteMapping("/delete")

public ResponseEntity<ResponseDto> deleteAccountDetails(@RequestParam

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

String mobileNumber) {

…

}

}

}

package com.eazybytes.accounts.exception;

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@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);

}

…

}

POST <http://localhost:8080/api/create>  
{

    "name": "Madan Reddy",

    "email": "tutor",

    "mobileNumber": "4354437687"

}

{

    "name": "Madan Reddy",

    "email": "tutor@eazybytes",

    "mobileNumber": "43544376871"

}

{

    "name": "M",

    "email": "tutor@eazybytes",

    "mobileNumber": "4354437686"

}

**Step – 11 – Update audit columns using spring data**package com.eazybytes.accounts.entity;

import jakarta.persistence.EntityListeners;

import org.springframework.data.annotation.CreatedBy;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedBy;

import org.springframework.data.annotation.LastModifiedDate;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

@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.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");

}

}

package com.eazybytes.accounts.service.impl;

@Service

@AllArgsConstructor

public class AccountsServiceImpl implements IAccountService {

private AccountsRepository accountsRepository;

private CustomerRepository customerRepository;

@Override

public void createAccount(CustomerDto customerDto) {

…

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

~~customer.setCreatedBy("Anonymous");~~

…

}

private Accounts createNewAccount(Customer customer) {

…

~~newAccount.setCreatedAt(LocalDateTime.now());~~

~~newAccount.setCreatedBy("Anonymous");~~

…

}

…

}  
  
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);

}

}

Test the create and update endpoints and make sure the auditing columns are updated  
  
**Step – 11 – Documentation of REST API using springdoc openAPI**<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.2.0</version>

</dependency>  
  
<http://localhost:8080/swagger-ui/index.html>

**Step – 12 – Enhancing documentation of REST API using @OpenAPIDefinition**package com.eazybytes.accounts;

import io.swagger.v3.oas.annotations.\*;

@SpringBootApplication

@EnableJpaAuditing(auditorAwareRef = "auditAwareImpl")

**@OpenAPIDefinition(**

**info = @Info(**

**title = "Accounts microservice REST API Documentation",**

**description = "EazyBank Accounts microservice REST API Documentation",**

**version = "v1",**

**contact = @Contact(**

**name = "Madan Reddy",**

**email = "tutor@eazybytes.com",**

**url = "https://www.eazybytes.com"**

**),**

**license = @License(**

**name = "Apache 2.0",**

**url = "https://www.eazybytes.com"**

**)**

**),**

**externalDocs = @ExternalDocumentation(**

**description = "EazyBank Accounts microservice REST API Documentation",**

**url = "https://www.eazybytes.com/swagger-ui.html"**

**)**

**)**

public class AccountsApplication {

….

}<http://localhost:8080/swagger-ui/index.html>  
 **Step – 13 – Enhancing documentation of REST API using @Tag, @Operation and @ApiResponse  
  
Changes for @Tag**@Tag(

name = "CRUD REST APIs for Accounts in EazyBank",

description = "CRUD REST APIs in EazyBank to CREATE, UPDATE, FETCH AND DELETE account details"

)

…

public class AccountsController {  
 …  
}

**Changes for @Operation**

…

public class AccountsController {

…

@Operation(

summary = "Create Account REST API",

description = "REST API to create new Customer & Account inside EazyBank"

)

@PostMapping("/create")

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

…

}

@Operation(

summary = "Fetch Account Details REST API",

description = "REST API to fetch Customer & Account details based on a mobile number"

)

@GetMapping("/fetch")

public ResponseEntity<CustomerDto> fetchAccountDetails(@RequestParam

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

String mobileNumber) {

…

}

@Operation(

summary = "Update Account Details REST API",

description = "REST API to update Customer & Account details based on a account number"

)

@PutMapping("/update")

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

…

}

@Operation(

summary = "Delete Account & Customer Details REST API",

description = "REST API to delete Customer & Account details based on a mobile number"

)

@DeleteMapping("/delete")

public ResponseEntity<ResponseDto> deleteAccountDetails(@RequestParam

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

String mobileNumber) {

…

}

}

**Changes for @ApiResponse**package com.eazybytes.accounts.controller;

….

public class AccountsController {

….

@ApiResponses({

@ApiResponse(

responseCode = "201",

description = "HTTP Status CREATED"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error"

)

}

)

@PostMapping("/create")

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

….

}

….

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error"

)

}

)

@GetMapping("/fetch")

public ResponseEntity<CustomerDto> fetchAccountDetails(@RequestParam

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

String mobileNumber) {

….

}

….

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "417",

description = "Expectation Failed"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error"

)

}

)

@PutMapping("/update")

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

….

}

….

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "417",

description = "Expectation Failed"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error"

)

}

)

@DeleteMapping("/delete")

public ResponseEntity<ResponseDto> deleteAccountDetails(@RequestParam

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

String mobileNumber) {

….

}

}

<http://localhost:8080/swagger-ui/index.html>

**Step – 14 – Enhancing documentation of REST API using @Schema – Part 1**package com.eazybytes.accounts.dto;

@Schema(

name = "Customer",

description = "Schema to hold Customer and Account information"

)

public class CustomerDto {

@Schema(

description = "Name of the customer", example = "Eazy Bytes"

)

…

private String name;

@Schema(

description = "Email address of the customer", example = "tutor@eazybytes.com"

)

…

private String email;

@Schema(

description = "Mobile Number of the customer", example = "9345432123"

)

…

private String mobileNumber;

@Schema(

description = "Account details of the Customer"

)

private AccountsDto accountsDto;

}package com.eazybytes.accounts.dto;

Data

@Schema(

name = "Accounts",

description = "Schema to hold Account information"

)

public class AccountsDto {

@Schema(

description = "Account Number of Eazy Bank account", example = "3454433243"

)

…

private Long accountNumber;

@Schema(

description = "Account type of Eazy Bank account", example = "Savings"

)

…

private String accountType;

@Schema(

description = "Eazy Bank branch address", example = "123 NewYork"

)

…

private String branchAddress;

}

package com.eazybytes.accounts.dto;

@Schema(

name = "Response",

description = "Schema to hold successful response information"

)

public class ResponseDto {

@Schema(

description = "Status code in the response", example = "200"

)

private String statusCode;

@Schema(

description = "Status message in the response", example = "Request Processed successfully"

)

private String statusMsg;

}

package com.eazybytes.accounts.dto;

@Schema(

name = "ErrorResponse",

description = "Schema to hold error response information"

)

public class ErrorResponseDto {

@Schema(

description = "API path invoked by client"

)

private String apiPath;

@Schema(

description = "Error code representing the error happened"

)

private HttpStatus errorCode;

@Schema(

description = "Error message representing the error happened"

)

private String errorMessage;

@Schema(

description = "Time representing when the error happened"

)

private LocalDateTime errorTime;

}

package com.eazybytes.accounts.controller;

…

public class AccountsController {

….

@ApiResponses({

@ApiResponse(

responseCode = "201",

description = "HTTP Status CREATED"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@PostMapping("/create")

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

…

}

…

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@GetMapping("/fetch")

public ResponseEntity<CustomerDto> fetchAccountDetails(@RequestParam

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

String mobileNumber) {

…

}

…

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "417",

description = "Expectation Failed"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@PutMapping("/update")

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

…

}

…

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "417",

description = "Expectation Failed"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@DeleteMapping("/delete")

public ResponseEntity<ResponseDto> deleteAccountDetails(@RequestParam

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

String mobileNumber) {

…

}

}

**Step – 15 – Enhancing documentation of REST API using @Schema – Part 2  
  
  
  
  
Step – 16 – Important Annotations and Classes that supports building REST services**