@RestController

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

@Validated

public class AccountsController {

private static final Logger logger = LoggerFactory.getLogger(AccountsController.class);

private final IAccountsService iAccountsService;

public AccountsController(IAccountsService iAccountsService) {

this.iAccountsService = iAccountsService;

}

@Value("${build.version}")

private String buildVersion;

@Autowired

private Environment environment;

@Autowired

private AccountsContactInfoDto accountsContactInfoDto;

@Operation(

summary = "Create Account REST API",

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

)

@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) {

iAccountsService.createAccount(customerDto);

return ResponseEntity

.status(HttpStatus.CREATED)

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

}

@Operation(

summary = "Fetch Account Details REST API",

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

)

@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) {

CustomerDto customerDto = iAccountsService.fetchAccount(mobileNumber);

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

}

@Operation(

summary = "Update Account Details REST API",

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

)

@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) {

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));

}

}

@Operation(

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

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

)

@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) {

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));

}

}

@Operation(

summary = "Get Build information",

description = "Get Build information that is deployed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@Retry(name = "getBuildInfo",fallbackMethod = "getBuildInfoFallback")

@GetMapping("/build-info")

public ResponseEntity<String> getBuildInfo() {

logger.debug("getBuildInfo() method Invoked");

return ResponseEntity

.status(HttpStatus.OK)

.body(buildVersion);

}

public ResponseEntity<String> getBuildInfoFallback(Throwable throwable) {

logger.debug("getBuildInfoFallback() method Invoked");

return ResponseEntity

.status(HttpStatus.OK)

.body("0.9");

}

@Operation(

summary = "Get Java version",

description = "Get Java versions details that is installed into accounts microservice"

)

@ApiResponses({

@ApiResponse(

responseCode = "200",

description = "HTTP Status OK"

),

@ApiResponse(

responseCode = "500",

description = "HTTP Status Internal Server Error",

content = @Content(

schema = @Schema(implementation = ErrorResponseDto.class)

)

)

}

)

@RateLimiter(name= "getJavaVersion", fallbackMethod = "getJavaVersionFallback")

@GetMapping("/java-version")

public ResponseEntity<String> getJavaVersion() {

return ResponseEntity

.status(HttpStatus.OK)

.body(environment.getProperty("JAVA\_HOME"));

}

public ResponseEntity<String> getJavaVersionFallback(Throwable throwable) {

return ResponseEntity

.status(HttpStatus.OK)

.body("Java 17");

}

@Operation(

summary = "Get Contact Info",

description = "Contact Info details that can be reached out in case of any issues"

)

@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("/contact-info")

public ResponseEntity<AccountsContactInfoDto> getContactInfo() {

return ResponseEntity

.status(HttpStatus.OK)

.body(accountsContactInfoDto);

}A screenshot of a computer

Description automatically generated

@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(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);

}

@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);

}

@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);

}

@ResponseStatus(value = HttpStatus.BAD\_REQUEST)

public class CustomerAlreadyExistsException extends RuntimeException {

public CustomerAlreadyExistsException(String message) {

super(message);

}

}

@Repository

public interface AccountsRepository extends JpaRepository<Accounts, Long> {

Optional<Accounts> findByCustomerId(Long customerId);

@Transactional

@Modifying

void deleteByCustomerId(Long customerId);

}