```
        @Override
        protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
HttpHeaders headers, HttpStatus status, WebRequest request) {

                String errorMessage =
ex.getBindingResult().getAllErrors().stream().map(ObjectError::getDefaultMessage).collect(Collectors.joini
ng(", "));

                LOG.error("Request body validation failed. {}", errorMessage);

                ErrorResponse response = getErrorResponse(TypeEnum.ERROR, "400", "Please provide missing
fields in moreInfo", "Request body", errorMessage);

                return new ResponseEntity<>(response, headers, status);

        }


donorAccountIdentifierGroupList =
Arrays.stream(cacScLinksInquiry.getDonorComponentIdentifierGroup()).filter(Objects::nonNull).map(this::get
donorAccountIdentifier).collect(Collectors.toList());



SignerGroup[] arraySignerObj =
Arrays.stream(signerGroupArray).filter(Objects::nonNull).map(this::getFilteredSignergroup).toArray(SignerG
roup[]::new);

List<ErrorMessage> errorMessageList = errorMessage.stream().filter(e -> e.getErrorCode() != null &&
!e.getErrorCode().isEmpty()).collect(Collectors.toList());


List<CACOrSCLinkStatusGroupDTO> cacOrSCLinkStatus =
Arrays.stream(cacOrSCLinkStatusList).filter(Objects::nonNull).map(this::
getCACOrSCLinkStatusGroup).collect(Collectors.toList());

@Autowired
        OriginConfiguration originConfiguration;

        public static void main(String[] args) {
                TimeZone.setDefault(TimeZone.getTimeZone("CST6CDT"));
                SpringApplication.run(USRAOServicingOrder.class, args);
        }

        @Bean
    public WebMvcConfigurer corsConfigurer()
    {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/**").allowedOrigins(originConfiguration.getAllowedOrigin().stream().toArray(String[]
::new));
            }
        };
    }



Boolean isDuplicatePresent = vpProductsInfoList.stream()
        .anyMatch(s -> s.getCgAcctClassCod().equals(obj.getCgAcctClassCod())
        && s.getCgVpStrTxt().equals(vpRecord.getDpVpCod())&& s.getCgAcctTypeCod() ==
obj.getCgAcctTypeCod()&& s.getCgMktPlaceTxt().equals(obj.getCgMktPlaceTxt())&&
s.getCgBnkPkgTypCod().equals(obj.getCgBnkPkgTypCod()));


Map<String, String> countryCodeMap = countryCodes.stream().map(e -> new Locale("", e))
                .collect(Collectors.toMap(e -> e.getISO3Country(), e -> e.getDisplayCountry()));
```

```
        String errorMessage =
ex.getBindingResult().getAllErrors().stream().map(ObjectError::getDefaultMessage)
                                    .collect(Collectors.joining(", "));


saveCustomerFormsRequest.getCustomerFormList().stream().map(CustomerForm::new)
                                    .forEach(e -> mongoTemplate.save(e));

List<EodCommentDTO> commentsList = eodCommentsList.stream().map(EodCommentDTO::new)
                                    .collect(Collectors.toList());

result = staticFormList.stream().map(form -> updatePdfBytes(form, builder))
                .reduce((result1, result2) -> (result1 && result2)).orElse(false);



List<CustomerFormDTO> formList = customerFormList.stream().map(this::getCustomerForm)
                                        .map(CustomerFormDTO::new).collect(Collectors.toList());



List<String> cins = applicantList.stream().filter(e -> (e.getCin() != null && !e.getCin().equals("")))
                                    .map(ApplicantDTO::getCin).collect(Collectors.toList());

Map<String, List<ProductDTO>> valuePropProdPkgMap = aoRequest.getApplication().getProduct().stream()
                                    .filter(s -> null != s.getValuePropositionCode())
                                    .collect(Collectors.groupingBy(ProductDTO::getValuePropositionCode));

Map<String, ApplicantDTO> applicantMap =
applicantList.stream().collect(Collectors.toMap(ApplicantDTO::getCin, e -> e));


applicant.getEmployment().stream().map(e -> getEmploymentReferencesGroup(e,
applicant.getActionCode())).toArray(EmploymentReferencesGroup[]::new)

applicant.getFinancialReference().stream().map(this::getFinancialReferencesGroup).filter(Objects::nonNull)
.collect(Collectors.toCollection(ArrayList::new))

applicant.getEmail().stream().map(this::getEmailGroup).toArray(EmailGroup[]::new)

ccdg.setProgramEnrollmentIdentifier(product.getEcrmPromotionalOffer().getEcrmOffer().stream().anyMatch(e->
e.getCampaignId().startsWith("MRCVP"))?"1":"");

checkingAccountNumber = productList.stream().filter(PackageUtility::isCheckingProduct)
        .collect(Collectors.toMap(ProductDTO::getBankingPackageNumber, ProductDTO::getAccountNumber));

private static CardGroup getCurrentCardGroup(CardGroup[] cardGroup) {
                return Arrays.asList(cardGroup).stream().filter(e -> e.getCurrentLevelOfIssuance() !=
null).reduce((e1, e2) -> Integer.parseInt(e1.getCurrentLevelOfIssuance()) >
Integer.parseInt(e2.getCurrentLevelOfIssuance()) ? e1 : e2).orElse(null);
        }



AtomicInteger index = new AtomicInteger(0);

productList.stream().filter(PackageUtility::isCheckingAndSavingProduct)
                                    .forEach(e ->
e.setAccountNumber(accountNumberList.get(index.getAndIncrement())));


                checkingAccountNumber = productList.stream().filter(PackageUtility::isCheckingProduct)
                                    .collect(Collectors.toMap(ProductDTO::getBankingPackageNumber,
ProductDTO::getAccountNumber));

                List<String> cins = applicantList.stream().filter(e -> (e.getCin() != null &&
!e.getCin().equals(""))).map(ApplicantDTO::getCin).collect(Collectors.toList());
```

```
        if(signerGroup.stream().map(SignerGroup::getGCIFCustomerNumber).allMatch(e -> isSignerPresent(e,
customerInformationList)))

        private boolean isSignerPresent(String gcifId, List<GetCustomerDetailInformation>
customerInformationList)
        {
                return customerInformationList.stream().anyMatch(e ->
gcifId.equals(e.getGCIFCustomerNumber()));
        }

private List<GetCustomerDetailInformation> getSignerInformationOfProduct(List<CinDetailDTO> cinDetails,
Map<String, GetCustomerDetailInformation> customerDetailMap)
        {
                return
cinDetails.stream().map(CinDetailDTO::getCin).map(customerDetailMap::get).collect(Collectors.toList());
        }


filteredList = filteredList.stream().filter(s -> null != s.getDomicileBranchState()
        && s.getDomicileBranchState().equalsIgnoreCase(governingState))
                                        .collect(Collectors.toList());



public List<AccountCinList> getAccNumberForFilenet(List<FilenetDocument> imageCaptureList) {

                List<AccountCinList> accountCinList = new ArrayList<>();

                imageCaptureList.stream().forEach(e -> {
                        AccountCinList accountCin = new AccountCinList();
                        if (e.getAccountNumber() != null && !e.getAccountNumber().isEmpty() &&
e.getAccountNumber().contains(",")) {
                                accountCin.setAccountNumber(e.getAccountNumber().split(",")[0].trim());
                        } else {
                                accountCin.setAccountNumber(e.getAccountNumber());
                        }
                        accountCin.setCin(e.getCin());
                        accountCinList.add(accountCin);
                });

                return accountCinList;

        }




        @Bean
        public Map<String, String> countryCodeMap() {

                List<String> countryCodes = Arrays.asList(Locale.getISOCountries());
                Map<String, String> countryCodeMap = countryCodes.stream().map(e -> new Locale("", e))
                                .collect(Collectors.toMap(e -> e.getISO3Country(), e ->
e.getDisplayCountry()));
                return countryCodeMap;
        }



User user = users.stream().filter(e-> e.getId().equalsIgnoreCase("2")).collect(onlyElement());

List<E7538> lst4604Acc = Arrays.asList(tps4604Response.get().getE5999().getE7538()).stream().filter(a ->
a.getE7000().equals(accountDetails.getAccountNumber())).filter(b ->
b.getE7296().equalsIgnoreCase("1")).collect(Collectors.toList());
```

```
List<GcifTransactionDTO> gcifTransactionList = gcifTransactions.stream().map(GcifTransactionDTO::new)
                                .collect(Collectors.toList());
```

```
        List<CustomerFormDTO> formList = customerFormList.stream().map(this::getCustomerForm)
                                        .map(CustomerFormDTO::new).collect(Collectors.toList());
```

```
        public CustomerForm getCustomerForm(CustomerForm customerForm) {
                if (customerForm.getStaticFormId() != null && customerForm.getStaticFormFlag() != null) {
                        String staticCustomerFormId = customerForm.getStaticFormId();
                        Query staticFormQuery = new
Query(Criteria.where("staticCustomerFormId").is(staticCustomerFormId));
                        StaticCustomerForm staticCustomerForm = mongoTemplate.findOne(staticFormQuery,
StaticCustomerForm.class);
                        if (staticCustomerForm != null) {
                                customerForm.setPdfBytes(staticCustomerForm.getPdfBytes());
                                if(customerForm.getDisplayOrderPreAo() == null) {

        customerForm.setDisplayOrderPreAo(staticCustomerForm.getDisplayOrderPreAo());
                                }
                                if(customerForm.getDisplayOrderPostAo() == null) {

        customerForm.setDisplayOrderPostAo(staticCustomerForm.getDisplayOrderPostAo());
                                }
                        }
                }
```

```
        envelopeStatusresponse  = envelopeStatusRequest.getRecipientStatusList().stream()
                                        .map(e -> getDocusignRecipientStatus(envelopeId, e))
                                        .map(e -> new
DocusignRecipientStatusDTO(e,envelopeStatus)).collect(Collectors.toList());
```

```
saveEcrmPromotionsRequest.getEcrmPromotions().stream().map(e ->
this.getEcrmPromtions(saveEcrmPromotionsRequest, e)).forEach(e -> mongoTemplate.save(e));
```

```
Map<String, List<ProductDTO>> valuePropProdPkgMap = aoRequest.getApplication().getProduct().stream()
.filter(s -> null != s.getValuePropositionCode())
        .collect(Collectors.groupingBy(ProductDTO::getValuePropositionCode));
```

```
private boolean checkForBranchConfiguration(List<BranchFeatureDTO> branchFeatureList, String channelName)
{
                return branchFeatureList.stream().filter(e ->
e.getChannel().equalsIgnoreCase(channelName)).map(e -> (e.getSwitchFlag() != null &&
e.getSwitchFlag().equals("B"))).collect(Collectors.reducing((a, b) -> (a || b))).orElse(true);
        }
```

```
primaryCin = customerDetailMap.keySet().stream().findFirst().orElse(null);
```

```
highRiskGroup.setHRCFlagsGroup(Arrays.stream(highRisk.getHrcflagsGroup()).map(e -> {
                                HRCFlagsGroup hRCFlagsGroup = new HRCFlagsGroup();
```

```
                                hRCFlagsGroup.setHighRiskCode(e.getHighRiskCode());
                                return hRCFlagsGroup;
                        }).toArray(HRCFlagsGroup[]::new));

ProductDTO product = branchAORequest.getApplication().getProduct().stream().filter(
                                                        e -> e.getCinDetails().stream().anyMatch(cin ->
cin.getCin().equals(applicant.getCin())))
                                                        .findFirst().orElse(null);


auc.setOtherRealEstateGroup(applicant.getRealEstate().stream().map(this::getOtherRealEstateGroup).collect(
Collectors.toCollection(ArrayList::new)));



ProductDTO product = branchAORequest.getApplication().getProduct().stream().filter(
                                                        e -> e.getCinDetails().stream().anyMatch(cin ->
cin.getCin().equals(applicant.getCin())))
                                                        .findFirst().orElse(null);


AtomicInteger index = new AtomicInteger(0);
productList.stream().filter(PackageUtility::isCheckingAndSavingProduct)
                                        .forEach(e ->
e.setAccountNumber(accountNumberList.get(index.getAndIncrement())));


        private static CardGroup getCurrentCardGroup(CardGroup[] cardGroup) {
                return Arrays.asList(cardGroup).stream().filter(e -> e.getCurrentLevelOfIssuance() !=
null).reduce((e1, e2) -> Integer.parseInt(e1.getCurrentLevelOfIssuance()) >
Integer.parseInt(e2.getCurrentLevelOfIssuance()) ? e1 : e2).orElse(null);
        }




@Component
public class CacheEvictionScheduler {

        private static final Logger LOG = LoggerFactory.getLogger(CacheEvictionScheduler.class.getName());

        @Autowired
        private CacheManager cacheManager;

        @Scheduled(cron = "0 0 23 * * ?")
        public void evictCacheValues() {

                try {
                        LOG.info("Service: XD-D-CustomerVerification - Cache Evication Started ");

                        List<String> evictCacheNames = Arrays.asList("userDefinition");

                        evictCacheNames.parallelStream().forEach(cacheName ->
cacheManager.getCache(cacheName).clear());

                        LOG.info("Service: XD-D-CustomerVerification - Cache Evication Completed ");
                } catch (Exception e) {
                        LOG.error("Service: XD-D-CustomerVerification - Cache Evication Failed ", e);
                }

        }

}



origin:
```

```
  allowed-origin:
    - https://sd-03cb-02a8.nam.nsroot.net
    - https://sit1.usretailaccountopening.citigroup.net
```

```
@Configuration
@ConfigurationProperties(prefix = "origin")
public class OriginConfiguration {

        private List<String> allowedOrigin;

        public List<String> getAllowedOrigin() {
                return allowedOrigin;
        }

        public void setAllowedOrigin(List<String> allowedOrigin) {
                this.allowedOrigin = allowedOrigin;
        }

}
```

```
        @Bean
    public WebMvcConfigurer corsConfigurer()
    {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/**").allowedOrigins(originConfiguration.getAllowedOrigin().stream().toArray(String[]
::new));
            }
        };
    }
```

```
spring:
  profiles: DEV
  datasource:
    url: ccccc
    username: cc
    password: cc
    driver-class-name: oracle.jdbc.driver.OracleDriver
    testOnBorrow: true
    testWhileIdle: true
    timeBetweenEvictionRunsMillis: 60000
    minEvictableIdleTimeMillis: 40000
    validationQuery: SELECT 1 FROM DUAL
    max-active: 150
    max-idle: 25
    max-wait: 8000
  batch:
    initializer:
      enabled: false
    job:
      enabled: false
```