**2022년도 졸업프로젝트**

**팀이름: 황제펭귄**

주제: TLS를 구현을 통한 홈페이지 제작 및 보안 채널 로그인 기능 구현

|  |  |  |  |
| --- | --- | --- | --- |
| **작성자** | 이유민 | **작성일자** | 2022.06.01 |

* **금일 진행 상황**

|  |  |
| --- | --- |
| **진행 상황** | **비고** |
| **진행 중**  **완료**  **필요없는 import 삭제 및 Service 코드 Refactoring**  **AccountService(CheckAccount 추가)**  @Service @RequiredArgsConstructor public class AccountService {   private final AccountRepository accountRepository;  private final MemberService memberService;  private final PasswordEncoder passwordEncoder;  private final TransactionService transactionService;   @Transactional  public void createAccount(AccountCreateReq accountCreateReq) {  String loginId = accountCreateReq.getLoginId();  String password = accountCreateReq.getAccountPassword();   checkPassword(password);  // 전달 받은 아이디를 통해 사용자가 있는지 확인 없다면 사용자가 없다는 예외를 발생.  memberService.checkMember(loginId);   try{  accountRepository.save(accountCreateReq.toEntity(  memberService.getMember(loginId),  passwordEncoder.encode(password),  0L));  }catch (Exception e)  {  e.printStackTrace();  throw new AccountException("계좌생성에 실패했습니다.");  }  }   @Transactional  public void transaction(TransferReq transferReq)  {  String loginId = transferReq.getLoginId();  Long sendAccount = transferReq.getSendAccountNumber();  String password = transferReq.getAccountPassword();   if(transferReq.getMyAccountNumber() == transferReq.getSendAccountNumber()){  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_CURRING*);  }   Account myAccount = accountRepository  .findAccountByAccountNumber(transferReq.getMyAccountNumber())  .orElseThrow(() -> new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_NOT\_FOUND*));   // 사용자의 아이디로부터 자신의 계좌가 맞는지 확인.  checkAccount(myAccount, memberService.getMember(loginId));   // 보내는 계좌 번호가 존재하는지 확인.  if(!accountRepository.existsAccountByAccountNumber(sendAccount)){  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_NOT\_FOUND*);  }  // 사용자의 계좌 비밀번호가 맞는지 확인.  if(!passwordEncoder.matches(password, myAccount.getAccountPassword())) {  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_PASSWORD\_NOT\_MATCH*);  }  // 사용자의 계좌에 충분한 잔액이 있는지 확인.  if(myAccount.getBalance() < transferReq.getBalance()) {  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_BALANCE*);  }  try {  accountRepository.updateMyBalance(transferReq.getBalance(),  myAccount.getAccountNumber());  accountRepository.updateBalance(transferReq.getBalance(),  transferReq.getSendAccountNumber());  // 전달자의 거래내역을 저장  transactionService.saveData(Transaction.*builder*()  .sendAccount(transferReq.getMyAccountNumber())  .toSenderMessage(transferReq.getToSenderMessage())  .receiveAccount(transferReq.getSendAccountNumber())  .toReceiverMessage(transferReq.getToReceiverMessage())  .balance(-transferReq.getBalance())  .transactionDate(new DateConfig().getDateTime())  .build());  // 받는이의 거래내역을 저장  transactionService.saveData(Transaction.*builder*()  .sendAccount(transferReq.getSendAccountNumber())  .toSenderMessage(transferReq.getToSenderMessage())  .receiveAccount(transferReq.getMyAccountNumber())  .toReceiverMessage(transferReq.getToReceiverMessage())  .balance(transferReq.getBalance())  .transactionDate(new DateConfig().getDateTime())  .build());  }catch (Exception e){  e.printStackTrace();  throw new AccountException("계좌이체에 실패했습니다.");  }  }   @Transactional  public List<AccountInquiryRes> inquiry(String loginId) {  // 정확한 사용자를 넘겨줬는지 확인  memberService.checkMember(loginId);   return accountRepository  .findAllByMemberId(memberService.getMember(loginId))  .stream()  .map(Account::toDto)  .collect(Collectors.*toList*());  }   private void checkPassword(String password)  {  // 계좌 비밀번호는 숫자로 6자로 구성되어있다.  Pattern passwordExpression = Pattern.*compile*("[0-9]{6}");  if(!passwordExpression.matcher(password).matches()){  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_PASSWORD\_FORMAT*);  }  }   private void checkAccount(Account account, Member member){  if(!accountRepository.findAllByMemberId(member).contains(account)){  throw new AccountException("전달받은 계좌는 사용자의 계좌가 아닙니다.");  }  }   public void checkAccount(Long accountNumber){  if(!accountRepository.existsAccountByAccountNumber(accountNumber)){  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_NOT\_FOUND*);  }  }   public Account getAccount(Long accountNumber){  return accountRepository.findAccountByAccountNumber(accountNumber)  .orElseThrow(() -> new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_NOT\_FOUND*));  } }  **CardController,CardRequestRes,CardRepository,CardListController**  **-주석 삭제**  **-비활성화된 Import 삭제**  **CardService(CheckMember 사용)**  @Service @RequiredArgsConstructor public class CardService {   private final CardRepository cardRepository;  private final MemberService memberService;  private final AccountService accountService;   @Transactional  public void createCard(CardCreateReq cardCreateReq) {   String loginId = cardCreateReq.getLoginId();  Long accountNumber = cardCreateReq.getAccountNumber();   try {  cardRepository.save(cardCreateReq.toEntity(  memberService.getMember(loginId),  accountService.getAccount(accountNumber))  );  } catch (Exception e) {  e.printStackTrace();  throw new CardException("카드 생성에 실패했습니다.");  }  }   @Transactional  public List<CardRequestRes> cardList(String loginId) {  // 정확한 사용자를 넘겨줬는지 확인  memberService.checkMember(loginId);   return cardRepository  .findAllByMemberId(memberService.getMember(loginId))  .stream()  .map(Card::toDto)  .collect(Collectors.*toList*());  }   }  **CounselService(CheckMember사용)**  @Service @RequiredArgsConstructor public class CounselService {   private final CounselRepository counselRepository;  private final MemberService memberService;   @Transactional  public void createCounsel(CounselCreateReq counselCreateReq) {  String loginId = counselCreateReq.getLoginId();  // 먼저 해당 사용자가 있는 지 검증  memberService.checkMember(loginId);  try {  counselRepository.save(counselCreateReq.toEntity(  memberService.getMember(loginId))  );  } catch (Exception e) {  e.printStackTrace();  throw new CounselException("상담글 생성에 실패했습니다.");  }  }   @Transactional  public List<CounselInquiryRes> listAllCounsel(String loginId) {  Member member = memberService.getMember(loginId);   return counselRepository.findAllByMemberId(member)  .stream()  .map(Counsel::toDto)  .collect(Collectors.*toList*());  }   @Transactional  public Counsel getSingleCounsel (Long id) {  if(!counselRepository.existsById(id)){  throw new CounselException(ExceptionMessages.*ERROR\_COUNSEL\_NOT\_EXIST*);  }  return counselRepository  .findById(id)  .orElseThrow(() -> new CounselException(ExceptionMessages.*ERROR\_UNDEFINED*));  }   @Transactional  public void updateCounsel(CounselUpdateReq counselUpdateReq) {  if(!counselRepository.existsById(counselUpdateReq.getId())){  throw new CounselException(ExceptionMessages.*ERROR\_EVENT\_NOT\_EXIST*);  }  Counsel counsel = counselRepository.getById(counselUpdateReq.getId());  if(!counsel.getMemberId().getLoginId().equals(counselUpdateReq.getLoginId())){  throw new CounselException(ExceptionMessages.*ERROR\_COUNSEL\_UNAUTHORIZED\_ACCESS*);  }  try {  counselRepository.updateCounsel(counselUpdateReq.getId(),  counselUpdateReq.getTitle(),  counselUpdateReq.getContent(),  new DateConfig().getDateTime());  }catch (Exception e){  e.printStackTrace();  throw new CounselException("상담글 업데이트에 실패했습니다.");  }  }   @Transactional  public void deleteCounsel(Long id, String loginId) {  if(!counselRepository.existsById(id)){  throw new CounselException(ExceptionMessages.*ERROR\_COUNSEL\_NOT\_EXIST*);  }  Counsel counsel = counselRepository.getById(id);  if(!counsel.getMemberId().getLoginId().equals(loginId)){  throw new CounselException(ExceptionMessages.*ERROR\_COUNSEL\_UNAUTHORIZED\_ACCESS*);  }  try{  counselRepository.deleteById(id);  }catch (Exception e){  e.printStackTrace();  throw new CounselException("상담글 삭제에 실패했습니다.");  }  } }  **MemberService(checkMember,getMember추가)**  @Service @RequiredArgsConstructor public class MemberService {   private final MemberRepository memberRepository;  private final PasswordEncoder passwordEncoder;  private final HttpSession httpSession;   @Transactional  public void register(MemberRegisterReq memberRegisterRequest) {  String loginId = memberRegisterRequest.getLoginId();  String email = memberRegisterRequest.getEmail();  String phoneNumber = memberRegisterRequest.getPhoneNumber();  String password1 = memberRegisterRequest.getPassword1();  String password2 = memberRegisterRequest.getPassword2();   if(memberRegisterRequest.checkNull()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_CREATE\_FORM\_HAS\_NULL*);  }   if(memberRegisterRequest.checkLength()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_LENGTH\_LIMIT*);  }   checkMember(loginId);  // 사용자의 아이디와 비밀번호 검증  checkLoginId(loginId);  checkMemberPassword(password1,password2);   if(memberRepository.findByEmail(email).isPresent()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_EMAIL\_DUPLICATE*);  }  if(memberRepository.findByPhoneNumber(phoneNumber).isPresent()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_PHONENUMBER\_DUPLICATE*);  }  else {  try{  memberRepository.save(memberRegisterRequest.toEntity(passwordEncoder.encode(password1), Role.*USER*));  }catch (Exception e){  // Exception 이 발생한 이유와 위치는 어디에서 발생했는지 전체적인 단계를 다 출력합니다.  e.printStackTrace();  throw new MemberException("회원가입에 실패했습니다.");  }  }  }   @Transactional  public Member login(MemberLoginReq memberLoginReq)  {  Member member = memberRepository  .findByLoginId(memberLoginReq.getLoginId())  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*));  if (!passwordEncoder.matches(memberLoginReq.getPassword(), member.getPassword())){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_PASSWORD*);  }   memberRepository.updateLoginDate(new DateConfig().getDateTime(), memberLoginReq.getLoginId());  // HttpSession Or JWT 도입예정.   // 임시적으로 세션을 등록  httpSession.setAttribute("user", new MemberSessionDto(member));   return memberRepository  .findByLoginId(memberLoginReq.getLoginId())  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_UNDEFINED*));  }   private void checkMemberPassword(String password1, String password2) {  // Password 규칙은 영문자, 특수문자를 포함 8~20이하이다.  Pattern passwordExpression = Pattern.*compile*("^(?=.\*[a-zA-Z])(?=.\*\\d)(?=.\*\\W).{8,20}$");  if (!passwordExpression.matcher(password1).matches()) {  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_PASSWORD\_FORMAT*);  } else if (!password1.equals(password2)) {  throw new MemberException("입력한 비밀번호가 서로 다릅니다.");  }  }  private void checkLoginId(String loginId) {  // 시작은 영문으로만,{영문, 숫자} 으로만 이루어진 5 ~ 12자 이하이다.  Pattern nameExpression = Pattern.*compile*("^[a-zA-Z]{1}[a-zA-Z0-9]{4,11}$");  if (!nameExpression.matcher(loginId).matches()) {  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_ID\_FORMAT*);  }  }   public void checkMember(String loginId){  if(memberRepository.findByLoginId(loginId).isEmpty()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*);  }  }   public Member getMember(String loginId){  return memberRepository.findByLoginId(loginId)  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*));  }   }  **TransactionService(checkAccount사용,saveData추가)**  @Service @RequiredArgsConstructor public class TransactionService {   private final AccountService accountService;  private final TransactionRepository transactionRepository;   @Transactional  public List<TransactionInquiryRes> showTransactions(TransactionInquiryReq transactionInquiryReq){  // 사용자의 계좌가 없다면 예외를 발생.  accountService.checkAccount(transactionInquiryReq.getAccountNumber());   return transactionRepository.getAllBySendAccount(transactionInquiryReq.getAccountNumber()  , transactionInquiryReq.getStartDate()  , transactionInquiryReq.getEndDate())  .stream()  .map(Transaction::toDto)  .collect(Collectors.*toList*());  }   @Transactional  public void saveData(Transaction transaction){  try{  transactionRepository.save(transaction);  }catch (Exception e) {  throw new TransactionException(ExceptionMessages.*ERROR\_UNDEFINED*);  }  } }  **진행 예정** |  |
| * **특이사항 / 협업 사항** | |