**2022년도 졸업프로젝트**

**팀이름: 황제펭귄**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **작성자** | 이유민 | **작성일자** | 2022.05.31 |

* **금일 진행 상황**

|  |  |
| --- | --- |
| **진행 상황** | **비고** |
| **진행 중**  **완료**  **Card 기능 추가 및 수정**   1. **AddCard 추가**   **DateConfig 수정**  **-카드 만료일을 위한 ExpirteDate 추가**  @Configuration public class DateConfig {  public String getDateTime(){  return LocalDateTime.now(ZoneId.of("Asia/Seoul")).format(DateTimeFormatter.ofPattern("yyyy-MM-dd-HH:mm:ss"));  }    public static String getExpireDate() {  DateFormat dtf = new SimpleDateFormat("yyyy-MM-dd");  final Calendar cal = Calendar.getInstance();  cal.add(Calendar.YEAR, +10);  return dtf.format(cal.getTime());  } }  **CardException**  public class CardException extends RuntimeException{   public CardException(ExceptionMessages exceptionMessages) {  super(exceptionMessages.getMessage());  }   public CardException(String message){  super(message);  } }  **ExceptionMessages(추가)**  *ERROR\_CARD\_EXIST*("해당 카드번호를 가진 카드가 이미 있습니다."), *ERROR\_CARD\_NOT\_EXIST*("해당 카드번호를 가진 카드가 없습니다"),  **Card(Entity)**  @Builder @Getter @Entity @NoArgsConstructor @AllArgsConstructor public class Card {   @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long cardNumber;   @ManyToOne(fetch = FetchType.*LAZY*)  @JoinColumn(name="memberId")  private Member memberId;   @ManyToOne(fetch = FetchType.*LAZY*)  @JoinColumn(name="accountNumber")  private Account accountNumber;   @Column(nullable = false)  private String cardType;   @Column  private String expireDate;   public CardRequestRes toDto(){  return CardRequestRes.*builder*()  .cardNumber(cardNumber)  .accountNumber(accountNumber.getAccountNumber())  .cardType(cardType)  .expireDate(expireDate)  .build();  } }  **CardController**  @Api(tags="카드 생성") @RequiredArgsConstructor @RestController @RequestMapping("/users/card") public class CardController {  private final CardService cardService;  private final ResponseService responseService;   @PostMapping("")  @ApiOperation(value="카드 생성", notes = "사용자의 아이디와 계좌번호, 카드종류를 받아 카드 생성")  public BaseResult createCard(@RequestBody CardCreateReq cardCreateReq) {  try {  cardService.createCard(cardCreateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }  @GetMapping(path = "/cardlist")  @ApiOperation(value = "카드 조회", notes = "아이디에 해당 하는 모든 카드를 받아옵니다.")  public BaseResult cardList(@ApiParam @RequestParam String loginId) {  try {  return responseService.listResult(cardService.cardList(loginId));  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }  **CardCreateReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CardCreateReq {   @ApiModelProperty(required = true)  private String loginId;   @ApiModelProperty(required = true)  private Long accountNumber;   @ApiModelProperty(required = true)  private String cardType;   public Card toEntity(Member member, Account account) {  return Card.*builder*()  .memberId(member)  .accountNumber(account)  .cardType(cardType)  .expireDate(new DateConfig().*getExpireDate*())  .build();  }  }  }  **CardRequestRes**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CardRequestRes {   private Long cardNumber;  private Long accountNumber;  private String cardType;  private String expireDate; }  **CardRepository**  public interface CardRepository extends JpaRepository<Card, Long> {  List<Card> findAllByMemberId(Member memberId); }  **CardService**  @Service @RequiredArgsConstructor public class CardService {   private final CardRepository cardRepository;  private final MemberRepository memberRepository;  private final AccountRepository accountRepository;   @Transactional  public void createCard(CardCreateReq cardCreateReq) {   String loginId = cardCreateReq.getLoginId();  Long accountNumber = cardCreateReq.getAccountNumber();   try {  cardRepository.save(cardCreateReq.toEntity(  memberRepository.findByLoginId(loginId).get(),  accountRepository.findAccountByAccountNumber(accountNumber).get())  );  } catch (Exception e) {  e.printStackTrace();  throw new CardException("카드 생성에 실패했습니다.");  }  }   @Transactional  public List<CardRequestRes> cardList(String loginId) {  // 정확한 사용자를 넘겨줬는지 확인  if (memberRepository.findByLoginId(loginId).isEmpty()) {  throw new CardException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*);  }  return cardRepository  .findAllByMemberId(memberRepository.findByLoginId(loginId).get())  .stream()  .map(Card::toDto)  .collect(Collectors.*toList*());  }   1. **CardList 추가**   **CardListException**  public CardListException(ExceptionMessages exceptionMessages) {  super(exceptionMessages.getMessage());  }   public CardListException(String message){  super(message);  } }  **ExceptionMassages**  *ERROR\_CARDLIST\_EXIST*("해당 제목을 가진 카드목록이 이미 있습니다."), *ERROR\_CARDLIST\_NOT\_EXIST*("해당 아이디를 가진 카드목록이 없습니다"),  **CardList(Entity)**  @Builder @Getter @Entity @NoArgsConstructor @AllArgsConstructor public class CardList {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;   @Column(unique = true ,nullable = false)  private String title;   @Column(nullable = false)  private String content;   public CardListInquiryRes toDto(){  return CardListInquiryRes.*builder*()  .id(id)  .title(title)  .content(content)  .build();  } }  **CardListController**  @Api(tags="카드목록 생성, 모든 카드목록 가저오기, 특정 카드목록 가져오기, 카드목록 업데이트, 카드목록 삭제") @RequiredArgsConstructor @RestController @RequestMapping("/users/cardlist") public class CardListController {   private final CardListService cardListService;  private final ResponseService responseService;   @PostMapping("")  @ApiOperation(value="카드목록 생성")  public BaseResult createCardList(@RequestBody CardListCreateReq cardListCreateReq) {  try {  cardListService.createCardList(cardListCreateReq);  return responseService.successResult();  } catch (Exception e) {  return responseService.failResult(  e.getMessage()  );  }  }   @GetMapping("")  @ApiOperation(value="카드목록 모두 가져오기")  public BaseResult listAllCardLists() {  try {  return responseService.listResult(cardListService.getAllCardLists());  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @GetMapping("/{id}")  @ApiOperation(value="특정 카드목록 가져오기")  public BaseResult getCardListDetail(@PathVariable Long id) {  try {  return responseService.singleResult(cardListService.getSingleCardList(id).toDto());  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @PutMapping("/{id}")  @ApiOperation(value="카드목록 업데이트")  public BaseResult updateCardList(@RequestBody CardListUpdateReq cardListUpdateReq) {  try {  cardListService.updateCardList(cardListUpdateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @DeleteMapping("/{id}")  @ApiOperation(value="카드목록 삭제")  public BaseResult deleteCardList(@PathVariable Long id) {  try {  cardListService.deleteCardList(id);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  } }  **CardListCreateReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CardListCreateReq {   @ApiModelProperty(required = true)  private String title;  @ApiModelProperty(required = true)  private String content;   public CardList toEntity() {  return CardList.*builder*()  .title(title)  .content(content)  .build();  } }  **CardListInquiryRes**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CardListInquiryRes {   private Long id;  private String title;  private String content;  }  **CardListInquiryReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CardListUpdateReq {   @ApiModelProperty(required = true)  private Long id;  @ApiModelProperty(required = true)  private String title;  @ApiModelProperty(required = true)  private String content; }  **CardListRepository**  public interface CardListRepository extends JpaRepository<CardList, Long> {  Optional<CardList> findByTitle(String title);   @Modifying(clearAutomatically = true)  @Query("update CardList n set n.title = ?2, n.content = ?3 where n.id = ?1")  void updateCardList(@Param("id")Long id, @Param("title") String title, @Param("content") String content); }  **CardListService**  @Service @RequiredArgsConstructor public class CardListService {   private final CardListRepository cardListRepository;   @Transactional  public void createCardList(CardListCreateReq cardListCreateReq) {  if(cardListRepository.findByTitle(cardListCreateReq.getTitle()).isPresent()){  throw new NewsException(ExceptionMessages.*ERROR\_CARDLIST\_EXIST*);  }  try{  cardListRepository.save(cardListCreateReq.toEntity());  }catch (Exception e){  e.printStackTrace();  throw new CardListException("카드목록 생성에 실패했습니다.");  }  }   @Transactional  public List<CardListInquiryRes> getAllCardLists() {  return cardListRepository.findAll()  .stream()  .map(CardList::toDto)  .collect(Collectors.*toList*());  }   @Transactional  public CardList getSingleCardList(Long id) {  if(!cardListRepository.existsById(id)){  throw new CardException(ExceptionMessages.*ERROR\_CARDLIST\_NOT\_EXIST*);  }  return cardListRepository  .findById(id)  .orElseThrow(() -> new CardListException(ExceptionMessages.*ERROR\_UNDEFINED*));  }   @Transactional  public void updateCardList(CardListUpdateReq cardListUpdateReq) {  if(!cardListRepository.existsById(cardListUpdateReq.getId())){  throw new CardListException(ExceptionMessages.*ERROR\_CARDLIST\_NOT\_EXIST*);  }  try {  cardListRepository.updateCardList(cardListUpdateReq.getId(),cardListUpdateReq.getTitle(),cardListUpdateReq.getContent());  }catch (Exception e){  e.printStackTrace();  throw new CardListException("카드목록 업데이트에 실패했습니다.");  }  }   @Transactional  public void deleteCardList(Long id) {  if(!cardListRepository.existsById(id)){  throw new CardListException(ExceptionMessages.*ERROR\_CARDLIST\_NOT\_EXIST*);  }  try{  cardListRepository.deleteById(id);  }catch (Exception e){  e.printStackTrace();  throw new CardListException("카드목록 삭제에 실패했습니다.");  }  } }  **Create TIme 수정(getDateTime→getDate)**  **EventCreateReq**  Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class EventCreateReq {   @ApiModelProperty(required = true)  private String title;   @ApiModelProperty(required = true)  private String content;   @ApiModelProperty(required = true)  private String startDate;   @ApiModelProperty(required = true)  private String endDate;   public Event toEntity() {  return Event.*builder*()  .title(title)  .content(content)  .createDate(new DateConfig().getDate())  .startDate(startDate)  .endDate(endDate)  .build();  } }  **NewCreateReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class NewsCreateReq {   @ApiModelProperty(required = true)  private String title;  @ApiModelProperty(required = true)  private String content;   public News toEntity() {  return News.*builder*()  .title(title)  .content(content)  .createdDate(new DateConfig().getDate())  .build();  } }  **SecurityNoticeCreateReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class SecurityNoticeCreateReq {   @ApiModelProperty(required = true)  private String title;  @ApiModelProperty(required = true)  private String content;   public SecurityNotices toEntity() {  return SecurityNotices.*builder*()  .title(title)  .content(content)  .createdDate(new DateConfig().getDate())  .build();  } }  **회의 논의사항 수정**   1. **빈 값으로 회원가입이 진행되는 문제**   **ExceoptionMassages(추가)**  *ERROR\_MEMBER\_CREATE\_FORM\_HAS\_NULL*("회원가입의 항목중 비어있는 항목이 있습니다"),  **MemberRegisterReq(추가)**  public boolean checkNull(){  if(loginId.length() == 0 |  password1.length() == 0 |  password2.length() == 0 |  name.length() == 0 |  email.length() == 0 |  age == 0 |  sex.length() == 0|  phoneNumber.length() == 0 |  question.length() == 0 |  ansWord.length() == 0 ){  return true;  }  else {  return false;  } }  public boolean checkLength(){  if (sex.length() >= 2 | phoneNumber.length() >= 14){  return true;  }  else{  return false;  } }  **MemberService(추가)**  f(memberRegisterRequest.checkNull()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_CREATE\_FORM\_HAS\_NULL*); }  if(memberRegisterRequest.checkLength()){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_LENGTH\_LIMIT*); }   1. **거래내역 조회 기간 조회**   **TransferReq(추가:SenderMessage,ReceiverMessage)**  @ApiModelProperty(required = true) private String toSenderMessage;  @ApiModelProperty(required = true) private String toReceiverMessage;  **AccountService(추가:SenderMessage,ReceiverMessage)**  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  transactionRepository.save(Transaction.*builder*()  .sendAccount(transferReq.getMyAccountNumber())  .toSenderMessage(transferReq.getToSenderMessage())  .receiveAccount(transferReq.getSendAccountNumber())  .toReceiverMessage(transferReq.getToReceiverMessage())  .balance(-transferReq.getBalance())  .transactionDate(new DateConfig().getDateTime())  .build());  **TransactionInquiryRes**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class TransactionInquiryRes {   private Long senderAccount;  private String toSenderMessage;  private Long receiverAccount;  private String toReceiverMessage;  private Long sendMoney;  private String transactionDate; }  **Transaction(Entity)(추가)**  @Builder @Getter @Entity @NoArgsConstructor @AllArgsConstructor public class Transaction {   @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long transactionId;   @Column  private Long sendAccount;   @Column  private Long receiveAccount;   @Column(nullable = false)  private Long balance;   @Column  private String toSenderMessage;   @Column  private String toReceiverMessage;   @Column  private String transactionDate;   public TransactionInquiryRes toDto() {  if (balance > 0) {  return TransactionInquiryRes.*builder*()  .senderAccount(receiveAccount)  .toSenderMessage(toSenderMessage)  .receiverAccount(sendAccount)  .toReceiverMessage(toReceiverMessage)  .sendMoney(balance)  .transactionDate(transactionDate)  .build();  } else {  return TransactionInquiryRes.*builder*()  .senderAccount(sendAccount)  .toSenderMessage(toSenderMessage)  .receiverAccount(receiveAccount)  .toReceiverMessage(toReceiverMessage)  .sendMoney(balance)  .transactionDate(transactionDate)  .build();  }  } }   1. **계좌에 자기가 자기에게 이체하는 서큘링 막는 기능 추가**   **ExceptionMassage(추가)**  *ERROR\_ACCOUNT\_CURRING*("이체를 하는 사람과 받는 사람이 동일합니다."),  **AccountService(추가:서큘링 막기)**  if(transferReq.getMyAccountNumber() == transferReq.getSendAccountNumber()){  throw new AccountException(ExceptionMessages.*ERROR\_ACCOUNT\_CURRING*); }   1. **본인 패스워드 변경 기능 추가**   **MemberUtilController(수정,추가)**  @Api(tags = "비밀번호 찾기, 임시비밀번호 수정 ,비밀번호 수정, 사용자 정보 가져오기")  @PostMapping(path = "/updateTempPassword") @ApiOperation(value = "임시 비밀번호를 가진 사용자의 비밀번호 변경",notes = "임시 비밀번호를 제공받은 유저의 새로운 비밀번호로 비밀번호를 변경합니다.") public BaseResult updateTemplateUserPassword(@ApiParam @RequestBody MemberPasswordUpdateReq memberPasswordUpdateReq){  try {  memberUtilService.updateTemplatePassword(memberPasswordUpdateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  } }  **MemberUtilService**  @Transactional public void updateTemplatePassword(MemberPasswordUpdateReq memberPasswordUpdateReq){  Member member = memberRepository  .findByLoginId(memberPasswordUpdateReq.getLoginId())  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*));   // 사용자가 임시 비밀번호를 사용중인지 확인  if(!member.isUsingTempPassword()){  throw new MemberException("사용자는 임시비밀번호를 사용중이 아닙니다.");  }   // 사용자의 임시 비밀번호가 맞는지 확인.  if(!passwordEncoder.matches(memberPasswordUpdateReq.getOldPassword(), member.getPassword())){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_PASSWORD*);  }  // 입력한 password가 규칙에 맞는지 확인.  checkMemberPassword(memberPasswordUpdateReq.getNewPassword1(), memberPasswordUpdateReq.getNewPassword2());  try {  memberRepository.updateUserPassword(  passwordEncoder.encode(memberPasswordUpdateReq.getNewPassword1()),  member.getMemberId());  memberRepository.updateUserModifyDate(  new DateConfig().getDateTime(),  member.getMemberId()  );  }catch (Exception e){  e.printStackTrace();  throw new MemberException("비밀번호 변경 실패");  } }  @Transactional public void updateNewPassword(MemberPasswordUpdateReq memberPasswordUpdateReq){  Member member = memberRepository  .findByLoginId(memberPasswordUpdateReq.getLoginId())  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*));   // 사용자의 현재 비밀번호가 맞는지 확인.  if(!passwordEncoder.matches(memberPasswordUpdateReq.getOldPassword(), member.getPassword())){  throw new MemberException(ExceptionMessages.*ERROR\_MEMBER\_PASSWORD*);  }  // 입력한 password가 규칙에 맞는지 확인.  checkMemberPassword(memberPasswordUpdateReq.getNewPassword1(), memberPasswordUpdateReq.getNewPassword2());  try {  memberRepository.updateUserPassword(  passwordEncoder.encode(memberPasswordUpdateReq.getNewPassword1()),  member.getMemberId());  memberRepository.updateUserModifyDate(  new DateConfig().getDateTime(),  member.getMemberId()  );  }catch (Exception e){  e.printStackTrace();  throw new MemberException("비밀번호 변경 실패");  } }   1. **새소식에 게시물마다 번호 넣어주기**   **News(Entity)(수정)**  public NewsInquiryRes toDto(){  return NewsInquiryRes.*builder*()  .id(id)  .title(title)  .content(content)  .createdDate(createdDate)  .build(); }  **NewsInquiryRes(수정)**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class NewsInquiryRes {   private Long id;  private String title;  private String content;  private String createdDate;  }   1. **새소식에 등록일 8자리로 잘라서 날짜표시**   **DateConfig(추가:getDate)**  @Configuration public class DateConfig {  public String getDateTime(){  return LocalDateTime.*now*(ZoneId.*of*("Asia/Seoul")).format(DateTimeFormatter.*ofPattern*("yyyy-MM-dd-HH:mm:ss"));  }   public String getDate(){  return LocalDateTime.*now*(ZoneId.*of*("Asia/Seoul")).format(DateTimeFormatter.*ofPattern*("yyyy-MM-dd"));  }    public static String getExpireDate() {  DateFormat dtf = new SimpleDateFormat("yyyy-MM-dd");  final Calendar cal = Calendar.*getInstance*();  cal.add(Calendar.*YEAR*, +10);  return dtf.format(cal.getTime());  } }   1. **상담센터 변경:자기글만 보이도록 표시**   **CounselController(수정)**  @GetMapping("/counsels") @ApiOperation(value="개인의 상담글 모두 가져오기", notes = "상담글을 모두 가져옵니다.") public BaseResult listAllCounsel(@RequestParam String loginId) {  try {  return responseService  .listResult(counselService.listAllCounsel(loginId));  }catch (Exception e) {  return responseService.failResult(  e.getMessage()  );  } }  **CounselRepository(수정)**  public interface CounselRepository extends JpaRepository<Counsel, Long> {   @Query("select c from Counsel c where c.memberId = ?1")  List<Counsel> findAllByMemberId(@Param("memberId") Member memberId);   @Modifying  @Query("update Counsel c set c.title = ?2, c.content = ?3, c.modifyDate = ?4 where c.id = ?1")  void updateCounsel(@Param("id")Long id, @Param("title") String title, @Param("content") String content, @Param("modifyDate") String modifyDate); }  **CounselService(수정)**  @Transactional public List<CounselInquiryRes> listAllCounsel(String loginId) {  Member member = memberRepository.findByLoginId(loginId)  .orElseThrow(() -> new MemberException(ExceptionMessages.*ERROR\_MEMBER\_NOT\_FOUND*));   return counselRepository.findAllByMemberId(member)  .stream()  .map(Counsel::toDto)  .collect(Collectors.*toList*()); }   1. **회원가입 Length 관련 추가**   **ExceptionMessages**  *ERROR\_MEMBER\_LENGTH\_LIMIT*("회원가입 항목 성별 또는 전화번호의 길이가 옳지 않습니다."),  **Member(Entity)**  @Column(length = 1, nullable = false) private String sex;  @Column(length = 14, unique = true, nullable = false) private String phoneNumber;  **MemberInquiryRes**  private String sex;   1. **거래시간 표시**   **TransactionController(수정)**  @Api(tags = "거래 내역 조회")  **Transaction(Entity)(추가)**  public TransactionInquiryRes toDto() {  if (balance > 0) {  return TransactionInquiryRes.*builder*()  .senderAccount(receiveAccount)  .toSenderMessage(toSenderMessage)  .receiverAccount(sendAccount)  .toReceiverMessage(toReceiverMessage)  .sendMoney(balance)  .transactionDate(transactionDate)  .build();  } else {  return TransactionInquiryRes.*builder*()  .senderAccount(sendAccount)  .toSenderMessage(toSenderMessage)  .receiverAccount(receiveAccount)  .toReceiverMessage(toReceiverMessage)  .sendMoney(balance)  .transactionDate(transactionDate)  .build();  }  **진행 예정** |  |
| * **특이사항 / 협업 사항** | |