**2022년도 졸업프로젝트**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **작성자** | 김도현 | **작성일자** | 2022.05.23 |

* **금일 진행 상황**

|  |  |
| --- | --- |
| **진행 상황** | **비고** |
| **질문 및 진행방식에 대한 논의**  **진행 중**  **완료**   1. **비밀번호 찾기, 수정 추가**   **Controller**  @PostMapping(path = "/find\_Password") @ApiOperation(value = "비밀번호 찾기") public BaseResult findPassword(@ApiParam @RequestBody UserFindPasswordReq userFindPasswordReq){  try {  return responseService.singleResult(  userService.setTemplatePassword(userFindPasswordReq));  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  } }   @PostMapping(path = "/updatePassword") @ApiOperation(value = "임시 비밀번호를 가진 사용자의 비밀번호 변경") public BaseResult updateUserPassword(@ApiParam @RequestBody UserPasswordUpdateReq userPasswordUpdateReq){  try {  userService.updateNewPassword(userPasswordUpdateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  } }  **dto/UserFindPasswordReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class UserFindPasswordReq {   @ApiModelProperty(required = true)  private String loginId;  @ApiModelProperty(required = true)  private String question;  @ApiModelProperty(required = true)  private String ansWord;  }  **dto/UserPasswordUpdateReq**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class UserPasswordUpdateReq {   @ApiModelProperty(required = true)  private String loginId;  @ApiModelProperty(required = true)  private String oldPassword;  @ApiModelProperty(required = true)  private String newPassword1;  @ApiModelProperty(required = true)  private String newPassword2;  }  **dto/UserSessionDto**  public class UserSessionDto implements Serializable {  private String userId;  private String password;   public UserSessionDto(User user){  this.userId = user.getLoginId();  this.password = user.getPassword();  } }  **질문,답 엔티티추가**  @Column private String question;  @Column(nullable = false) private String ansWord;  **Repository**  @Modifying(clearAutomatically = true) @Query("update User u set u.password = ?1 where u.userId = ?2") void updateUserPassword(@Param("newPassword")String newPassword, @Param("userId")Long userId);  @Modifying(clearAutomatically = true) @Query("update User u set u.password = ?1, u.usingTempPassword = true where u.userId = ?2") void updateUserTemplatePassword(@Param("templatePassword")String templatePassword, @Param("userId")Long userId);  **사용자 질문검증, 임시비밀번호사용중인지 확인, 임시비밀번호 일치확인, password 규칙 확인**  @Transactional public String setTemplatePassword(UserFindPasswordReq userFindPasswordReq){  User user = userRepository  .findByLoginId(userFindPasswordReq.getLoginId())  .orElseThrow(() -> new UserException(ExceptionMessages.*ERROR\_USER\_NOT\_FOUND*));   // 사용자 질문에 대한 검증  if (!user.getQuestion().equals(userFindPasswordReq.getQuestion())){  throw new UserException(ExceptionMessages.*ERROR\_USER\_QUESTION\_NOT\_MATCH*);  }  // 사용자 질문에 대한 답을 검증  if(!user.getAnsWord().equals(userFindPasswordReq.getAnsWord())){  throw new UserException(ExceptionMessages.*ERROR\_USER\_ANSWORD\_NOT\_MATCH*);  }  try {  String templatePassword = randomNumberGen();  userRepository.updateUserTemplatePassword(  passwordEncoder.encode(templatePassword),  user.getUserId()  );  return templatePassword;  }catch (Exception e){  e.printStackTrace();  throw new UserException("임시비밀번호로 변경이 실패했습니다.");  } }  @Transactional public void updateNewPassword(UserPasswordUpdateReq userPasswordUpdateReq){  User user = userRepository  .findByLoginId(userPasswordUpdateReq.getLoginId())  .orElseThrow(() -> new UserException(ExceptionMessages.*ERROR\_USER\_NOT\_FOUND*));   // 사용자가 임시 비밀번호를 사용중인지 확인  if(!user.isUsingTempPassword()){  throw new UserException("사용자는 임시비밀번호를 사용중이 아닙니다.");  }   // 사용자의 임시 비밀번호가 맞는지 확인.  if(!passwordEncoder.matches(userPasswordUpdateReq.getOldPassword(),user.getPassword())){  throw new UserException(ExceptionMessages.*ERROR\_USER\_PASSWORD*);  }  // 입력한 password가 규칙에 맞는지 확인.  checkUserPassword(userPasswordUpdateReq.getNewPassword1(), userPasswordUpdateReq.getNewPassword2());  try {  userRepository.updateUserPassword(  passwordEncoder.encode(userPasswordUpdateReq.getNewPassword1()),  user.getUserId());  }catch (Exception e){  e.printStackTrace();  throw new UserException("비밀번호 변경 실패");  } }  **임시 비밀번호 생성**  private String randomNumberGen(){  int lefLimit = 48; // '0'  int rightLimit = 122; // 'z'  int targetStringLength = 15; // 임의로 생성할 Password의 길이  Random random = new Random();   return random.ints(lefLimit,rightLimit +1)  .filter(i -> (i <= 57 || i >= 65) && (i <= 90 || i >= 97))  .limit(targetStringLength)  .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)  .toString(); }   1. **트랜젝션 패키지 생성**   **Controller**  public class TransactionController { }  **entity**  @Builder @Getter @Entity @NoArgsConstructor @AllArgsConstructor public class Transaction {   @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long transactionId;   @OneToOne  @JoinColumn(name = "Account")  private Account send;    @OneToOne  @JoinColumn(name = "Account")  private Account to;   @Column(nullable = false)  private Long balance; }  **Repository**  public interface TransactionRepository extends JpaRepository<Transaction, Long> { }  **service**  public class TransactionService { }   1. **상담센터 패키지 생성**   **Exception**  public class CounselException extends RuntimeException{   public CounselException(ExceptionMessages exceptionMessages) {  super(exceptionMessages.getMessage());  }   public CounselException(String message){  super(message);  } }  **Controller**  @Api(tags="상담글 생성, 모든 상담글 가져오기 ,특정 상담글 가져오기, 상담글 업데이트, 상담글 삭제") @RequiredArgsConstructor @RestController @RequestMapping("/cont") public class CounselController {  private final CounselService counselService;  private final ResponseService responseService;   @PostMapping("/counsel")  @ApiOperation(value="상담글 생성")  public BaseResult createCounsel(@RequestBody CounselCreateReq counselCreateReq) {  try {  counselService.createCounsel(counselCreateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @GetMapping("/counsel")  @ApiOperation(value="상담글 모두 가져오기")  public BaseResult listAllCounsel() {  try {  return responseService  .listResult(counselService.listAllCounsel());  }catch (Exception e) {  return responseService.failResult(  e.getMessage()  );  }  }   @GetMapping("/counsel/{id}")  @ApiOperation(value="특정 상담글 가져오기")  public BaseResult getCounselDetail(@PathVariable Long id) {  try {  return responseService.singleResult(  counselService.getSingleCounsel(id).toDto()  );  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @PutMapping("/counsel/{id}")  @ApiOperation(value="상담글 업데이트")  public BaseResult updateCounsel(@RequestBody CounselUpdateReq counselUpdateReq) {  try {  counselService.updateCounsel(counselUpdateReq);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  }   @DeleteMapping("/counsel/{id}")  @ApiOperation(value="상담글 삭제")  public BaseResult deleteCounsel(@PathVariable Long id) {  try {  counselService.deleteCounsel(id);  return responseService.successResult();  }catch (Exception e){  return responseService.failResult(  e.getMessage()  );  }  } }  **Dto**  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CounselCreateReq {  @ApiModelProperty(required = true)  private String title;  @ApiModelProperty(required = true)  private String content;   public Counsel toEntity() {  return Counsel.*builder*()  .title(title)  .content(content)  .writeDate(new DateConfig().getDateTime())  .build();  } }  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CounselRequestRes {  private String title;  private String content;  private String writeDate; }  @Data @Builder @RequiredArgsConstructor @AllArgsConstructor public class CounselUpdateReq {   @ApiModelProperty(required = true)  private Long id;   @ApiModelProperty(required = true)  private String title;   @ApiModelProperty(required = true)  private String content; }  **Entity**  @Builder @Getter @Entity @NoArgsConstructor @AllArgsConstructor public class Counsel { // @ManyToOne(fetch = FetchType.EAGER) // @JoinColumn(name="loginId", referencedColumnName = "loginId") // private User loginId;   @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;   @Column(columnDefinition = "text", unique = true ,nullable = false)  private String title;   @Column(columnDefinition = "text", nullable = false)  private String content;   @Column  private String writeDate;   public CounselRequestRes toDto(){  return CounselRequestRes.*builder*()  .title(title)  .content(content)  .writeDate(writeDate)  .build();  } }  **Repository**  public interface CounselRepository extends JpaRepository<Counsel, Long> {  Optional<Counsel> findByTitle(String title);   @Modifying  @Query("update Counsel n set n.title = ?2, n.content = ?3 where n.id = ?1")  void updateCounsel(@Param("id")Long id, @Param("title") String title, @Param("content") String content); }  **Service**  @Service @RequiredArgsConstructor public class CounselService {  private final CounselRepository counselRepository;   @Transactional  public void createCounsel(CounselCreateReq counselCreateReq) {  if (counselRepository.findByTitle(counselCreateReq.getTitle()).isPresent()) {  throw new EventException(ExceptionMessages.*ERROR\_EVENT\_EXIST*);  }  try {  counselRepository.save(counselCreateReq.toEntity());  } catch (Exception e) {  e.printStackTrace();  throw new EventException("이벤트 생성에 실패했습니다.");  } // return counselRepository.save(CounselCreateReq.toEntity());  }   @Transactional  public List<CounselRequestRes> listAllCounsel() {  return counselRepository.findAll()  .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 EventException(ExceptionMessages.*ERROR\_EVENT\_NOT\_EXIST*);  }  try {  counselRepository.updateCounsel(counselUpdateReq.getId(),counselUpdateReq.getTitle(),counselUpdateReq.getContent());  }catch (Exception e){  e.printStackTrace();  throw new CounselException("상담글 업데이트에 실패했습니다.");  }  }   @Transactional  public void deleteCounsel(Long id) {  if(!counselRepository.existsById(id)){  throw new CounselException(ExceptionMessages.*ERROR\_COUNSEL\_NOT\_EXIST*);  }  try {  counselRepository.deleteById(id);  }catch (Exception e){  e.printStackTrace();  throw new CounselException("상담글 삭제에 실패했습니다.");  }  } }  **진행 예정** | re |
| * **특이사항 / 협업 사항** | |