```java
@Component
@Getter
@PropertySource(value="classpath:/db.properties")
public class DataBaseProperties {
  @NonNull
  @Value("${db.username}")
  private String username;
  @NonNull
  @Value("${db.password}")
  private String password;
  @NonNull
  @Value("${db.driverClassName}")
  private String driverClassName;
  @NonNull
  @Value("${db.url}")
  private String url;
  @NonNull
  @Value("${db.initialSize}")
  private Integer initialSize;
  @NonNull
  @Value("${db.maxTotal}")
  private Integer maxTotal;
  @NonNull
  @Value("${db.maxIdle}")
  private Integer maxIdle;
  @NonNull
  @Value("${db.minIdle}")
  private Integer minIdle;
  @NonNull
  @Value("${db.maxWaitMillis}")
  private Integer maxWaitMillis;
  @NonNull
  @Value("${db.validationQuery}")
  private String validationQuery;
  @NonNull
  @Value("${db.testOnBorrow}")
  private String testOnBorrow;

  public boolean isTestOnBorrow() {
    return testOnBorrow.equals("true");
  }
}
```

```java
@EnableJpaRepositories(basePackageClasses = RepositoryBase.class)
@Configuration
```

```
public class JpaConfig {
@Bean
  LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource
dataSource){
  LocalContainerEntityManagerFactoryBean emf = new
LocalContainerEntityManagerFactoryBean();
  emf.setDataSource(dataSource);

emf.setPackagesToScan("com.nhnacademy.springboard.spirngmvcboard.entity");
  emf.setJpaVendorAdapter(jpaVendorAdapter());
  emf.setJpaProperties(jpaProperties());
  return emf;
}
private JpaVendorAdapter jpaVendorAdapter(){
  HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
  hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
  return hibernateJpaVendorAdapter;
}
private Properties jpaProperties(){
  Properties jpaProperties= new Properties();
  jpaProperties.setProperty("hibernate.show_sql", "true");
  jpaProperties.setProperty("hibernate.format_sql", "true");
  jpaProperties.setProperty("hibernate.use_sql_comments", "true");
  jpaProperties.setProperty("hibernate.globally_quoted_identifiers",
"true");
  jpaProperties.setProperty("hibernate.temp.use_jdbc_metadata_defaults",
"false");

  return jpaProperties;
}
  @Bean
  public PlatformTransactionManager
transactionManager(EntityManagerFactory entityManagerFactory){
    JpaTransactionManager jpaTransactionManager = new
JpaTransactionManager();
    jpaTransactionManager.setEntityManagerFactory(entityManagerFactory);
  return jpaTransactionManager;
  }
}
```

- service
- com/nhnacademy/springboard/spirngmvcboard/service/BoardService.java
- com/nhnacademy/springboard/spirngmvcboard/service/PostService.java
- com/nhnacademy/springboard/spirngmvcboard/service/UserService.java

```
public interface UserService {
User createUser(User user);
User getUser(String email);
```

```
User modifyUser(User user);
}
```

```
public interface PostService {
  Post createPost(Post post);
  Post getPost(String postId);
  Post modifyPost(Post post);
  List<Post> findByUserId(String userId);
  List<Post> findByBoardId(String boardId);
}
```

```
public interface BoardService {
Board createBoard(Board board);
Board getBoard(String boardId);
Board modifyBoard(Board board);
List<Board> findAll();

  }
```

- service.impl
- com/nhnacademy/springboard/spirngmvcboard/service/impl/BoardServiceImpl.java
- com/nhnacademy/springboard/spirngmvcboard/service/impl/PostServiceImpl.java
- com/nhnacademy/springboard/spirngmvcboard/service/impl/UserServiceImpl.java

```
@Service
@RequiredArgsConstructor
public class BoardServiceImpl implements BoardService {
  private final BoardRepository boardRepository;

  @Override
  public Board createBoard(Board board) {
    return boardRepository.save(board);
  }

  @Override
  public Board getBoard(String  boardId) {
    Optional<Board> boardOptional = boardRepository.findById(boardId);
    return boardOptional.orElse(null);
  }

  @Override
  public Board modifyBoard(Board board) {
    Optional<Board> boardOptional =
```

```
boardRepository.findById(String.valueOf(board.getBoardId()));
    if (boardOptional.isPresent()) {
      Board existingBoard = boardOptional.get();
      existingBoard.setBoardName(board.getBoardName());
      existingBoard.setDescription(board.getDescription());
      return boardRepository.save(existingBoard);
    } else {
      return null;
    }
  }

  @Override
  public List<Board> findAll() {
    return boardRepository.findAll();
  }
}
```

```
@Service
@RequiredArgsConstructor
@Transactional
public class PostServiceImpl implements PostService {
  private final PostRepository postRepository;

  @Override
  public Post createPost(Post post) {
    return postRepository.save(post);
  }

  @Override
  public Post getPost(String postId) {
    Optional<Post> postOptional = postRepository.findById(postId);
    return postOptional.orElse(null);
  }

  @Override
  public Post modifyPost(Post post) {
    Optional<Post> postOptional =
postRepository.findById(post.getPk().toString());
    if (postOptional.isPresent()) {
      Post existingPost = postOptional.get();
      existingPost.setTitle(post.getTitle());
      existingPost.setContent(post.getContent());
      return postRepository.save(existingPost);
    } else {
      return null;
    }
  }


  @Override
```

```
  public List<Post> findByPkUserId(String userId) {
    List<Post> allPosts = postRepository.findAll();
    return allPosts.stream()
        .filter(post -> post.getPk().getAuthorId().equals(userId))
        .collect(Collectors.toList()); }

  @Override
  public List<Post> findByPkBoardId(String boardId) {
    List<Post> allPosts = postRepository.findAll();
    return allPosts.stream()
        .filter(post -> post.getPk().getBoardId().equals(boardId))
        .collect(Collectors.toList());  }
}
```

```
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

  private final UserRepository userRepository;

  @Override
  public User createUser(User user) {
    return userRepository.save(user);
  }

  @Override
  public User getUserById(String id) {
    return userRepository.findById(id).orElse(null);
  }

  @Override
  public User getUserByEmail(String email) {
    List<User> allUsers = userRepository.findAll();
    Optional<User> userByEmail = allUsers.stream()
        .filter(user -> user.getEmail().equals(email))
        .findFirst();
    return userByEmail.orElse(null);
  }

  @Override
  public User modifyUser(User user) {
    Optional<User> existingUser =
userRepository.findById(String.valueOf(user.getUserId()));

    if (existingUser.isPresent()) {
      User modifiedUser = existingUser.get();
      modifiedUser.setEmail(user.getEmail());
      modifiedUser.setUsername(user.getUsername());
      modifiedUser.setPassword(user.getPassword());
```

```
        // 추가적으로 수정하고자 하는 필드들이 있다면 여기에서 수정하세요.

        return userRepository.save(modifiedUser);
      } else {
        return null;
      }
    }

    @Override
    public List<User> findAll() {
      return userRepository.findAll();
    }
}
```

```
@Getter
@Setter
public class FindPasswordRequest {
  private String email;
private String passwword;
}
```

```
@Getter
@Setter
public class LoginRequest {
  @NotBlank(message="이메일를 입력하세요")
  private String email;
  @NotBlank(message="비밀번호를 입력하세요")
  private String pwd;

  @Getter
  @Setter
  public static class FindIdRequest {
    private String email;

    // 생성자, getter, setter
  }
}
```

```
@Slf4j
@Service
@RequiredArgsConstructor
public class LoginService {
  private final UserMapper userMapper;
```

```
  public boolean isValidate(LoginRequest loginRequest) {
    User user = userMapper.findByUseremail(loginRequest.getEmail());
    boolean result = user != null &&
user.getPassword().equals(loginRequest.getPwd());
    log.info("isValidate : {}", result);
    return result;
  }
}
```

- com/nhnacademy/springboard/spirngmvcboard/thymeleaf
- com/nhnacademy/springboard/spirngmvcboard/thymeleaf/CustomTagDialet.java
- com/nhnacademy/springboard/spirngmvcboard/thymeleaf/TagUtils.java

```
//todo#15 CustomTagDialet 생성 , 커스텀 함수를 사용하기위해서 IDialect 확장한
IExpressionObjectDialect 구현.
public class CustomTagDialet extends AbstractDialect implements
IExpressionObjectDialect {

    public CustomTagDialet() {
        super("nhnacademy");
    }

    @Override
    public IExpressionObjectFactory getExpressionObjectFactory() {
        return new SpringStandardExpressionObjectFactory(){
            @Override
            public Set<String> getAllExpressionObjectNames() {
                return Collections.singleton("nhnacademy");
            }

            @Override
            public Object buildObject(IExpressionContext context, String
expressionObjectName) {
                 super.buildObject(context, expressionObjectName);
                 //todo#15 미리 만들어 뒀떤 TagUtils 객체;
                 return new TagUtils();
            }

            @Override
            public boolean isCacheable(String expressionObjectName) {
                return true;
            }
        };
    }
}
```

```
public class TagUtils {
    public String gender(Gender gender){
        if(gender.name().equals("M")){
            return "남성";
        } else if (gender.name().equals("F")) {
            return "여성";
        }else {
            return "";
        }
    }
}
```

```
public interface BoardRepository  extends JpaRepository<Board, String> {


}
@Repository
  public interface PostRepository  extends JpaRepository<Post, String> {

  List<Post> findByPkUserId(String userId);

  List<Post> findByPkBoardId(String boardId);
}
public interface RepositoryBase {

}
public interface UserRepository extends JpaRepository<User, String> {

  }
```