REVUE DE CODE SIAO

Diffusion de la présentation :

☐ Libre ☐ Interne ☐ Restreinte ☑ Confidentielle

Propriétaire de la présentation :

TITRE DE LA PRÉSENTATION

Suivi du Document

- Liste de diffusion :

Entité	Destinataire	Entité	Destinataire
SIAO			

- Historique :

Version	Date	Rédigé par	Vérifié par	Validé par
v1.0	19/05/2021	Anas JAADA	Anas JAADA	
	Motif et nature de la modification :		Création du document	
V2.0	jj/mm/aaaa			
	Motif et nature de la modification :			
	jj/mm/aaaa			
	Motif et nature de la modification :			

SOMMAIRE

CHAPITRE 1 : REVUE DE CODE

1

Audit de code

1.1 Audit de l'existant

1.2 Préconisations et Recommandations

1.1

Audit de l’existant

AUDIT DE L'EXISTANT

Légende

Echelle d'évaluation		
• Critique	• Vigilance	• Conforme
Une intervention prioritaire est nécessaire	Une mise à niveau est conseillée	Respecte la norme et satisfait le besoin

AUDIT DE L'EXISTANT

Axes d'évaluation

Thème	Axes d'évaluation
Qualité des développements et maintenabilité	• Couplages et dépendances du code • Commentaires • Nommage et lisibilité • Testabilité • Robustesse • Gestion des traces (logs) • Gestion des exceptions • Gestion des éditions

AUDIT DE L'EXISTANT

Audit Workspace

- L'audit de code a été réalisé via les outils et les références suivantes :
  - SonarQube 8.4.2
  - Intelij ide Ultimate Edition
  - Tattletale

AUDIT DE L'EXISTANT

Classes

- Les classes sont bien structurées suivant les standards java :
  - Ordre des définitions, des variables et des méthodes respecté
- Le scope des variables et des méthodes est maitrisé

  - Responsabilité des classes

```
public ResponseEntity<String> getAvailableLogin(String nom, String prenom, List<String> roles, List<Long> idsTerritoires) {
    LOG.debug("Acces à getAvailableLogin - UtilisateurService");
    String base = "siao_";
    String pNom = Normalizer.normalize( prenom.substring(0, 1).toLowerCase() + nom.toLowerCase(), Normalizer.Form.NFD ).replaceAll("\\p{InCombiningDiacriticalMarks}+\\p{P}\\p{S}\"", "").replaceAll("(\\W|_)*","");
    String login = this.buildLogin(base, pNom, roles, idsTerritoires);

    return new ResponseEntity<>("{\"login\":\"" + login + "\"}", HttpStatus.OK);
}
```

```
public String buildURL(String username, String password) {
    return UriComponentsBuilder.fromHttpUrl(this.baseUrl).path("/#/login").queryParam("login", username).queryParam("token", password).build().toString();
}
```

AUDIT DE L'EXISTANT

Classes

- Injection de dépendances via autowired ou constructeur ➔ harmonisation de la méthode

```java
@RestController
@RequestMapping("/api/demandeHebergements")
public class DemandeHebergementController {

    /**
     * Logger.
     */
    private static final Log LOG = LogFactory.getLog(DemandeHebergementC

    private static final int FORBIDDEN_ACCESS = 403;

    private DemandeHebergementRepository repository;
    private FicheRepository ficheRepository;
    private ReponseRepository reponseRepository;
    private DemandeHebergementService service;
    private GroupePlaceService groupePlaceService;
    private HistoriqueGroupePlaceService historiqueGroupePlaceService;
    private ReponseService reponseService;
    private UtilisateurService utilisateurService;
    private EditionCertificatService editionCertificatService;
    private TerritoireService territoireService;
    private DemandesExportService exportService;
    private ClotureDemandeHebergementService clotureDemandeService;
    private FicheService ficheService;
    private DemandeService demandeService;
    private DemandeInsertionService demandeInsertionService;
```

```java
@RestController
@RequestMapping("/api/demandeInsertion")
public class DemandeInsertionController {

    @Autowired
    private DemandeInsertionService service;

    @Autowired
    private DemandeInsertionRepository repository;

    @Autowired
    private UtilisateurService utilisateurService;

    @Autowired
    private FicheService ficheService;

    @Autowired
    private StructureService structureService;

    @Autowired
    private GroupesOrientationService groupesOrientationService;

    @Autowired
    private DemandeInsertionExportService exportService;

    @Autowired
    private DemandeService demandeService;

    @Autowired
    private DemandeHebergementService demandeHebergementService;

    @Autowired
    private EditionCertificatPECService editionCertificatPECService;

    @Autowired
    private GroupePlaceService groupePlaceService;
```

AUDIT DE L'EXISTANT

Classes (Controllers Abstraits)

- Utilisation de classes Controllers Abstraites afin de données accès à la couche Service et méthodes de conversions

```java
/**
 * Classe abstraite de controller d'{@link Utilisateur} qui met à disposition le service {@link UtilisateurService}
 * et une méthode de conversion d'un {@link Utilisateur} vers un {@link UtilisateurDTO}
 *
 * @author slerouvillois
 * @version 0$ 23/05/2019$
 * @date 23/05/2019
 */
public abstract class AUtilisateurController {

    protected UtilisateurService utilisateurService;
```

```java
@RestController
@RequestMapping("/api/utilisateurs")
public class UtilisateurController extends AUtilisateurController {

    /**
     * LOGGER
     */
    private static final Log LOG = LogFactory.getLog(UtilisateurController.class);

    private UtilisateurRepository repository;
```

AUDIT DE L'EXISTANT

Classes (Controllers Abstraits)

- Utilisation de classes Controller Abstraites devrait être généralisée à l'ensemble des Controllers

```
@RestController
@RequestMapping("/api/entiteGestionnaire")
public class EntiteGestionnaireController {

    @Autowired
    private UtilisateurService utilisateurService;

    @Autowired
    private EntiteGestionnaireService entiteGestionnaireService;

    @Autowired
    private TypeVoieRepository typeVoieRepository;
```

```
/**
 * Created by Guillaume CRESPEL on 03/05/2016.
 */
@RestController
@RequestMapping("/api/appels")
public class AppelController {

    private static final Log LOG = LogFactory.getLog(AppelController.class);

    @Autowired
    private AppelRepository repository;

    @Autowired
    private UtilisateurService utilisateurService;

    @Autowired
    private ProfilService profilService;
```

```
@RestController
@RequestMapping("/api/dispositifAccompagnement")
public class DispositifAccompagnementController {

    private DispositifAccompagnementService dispositifAccompagnementService;
    private UtilisateurService utilisateurService;
```

AUDIT DE L'EXISTANT

Classes (Controllers Abstraits)

- L'accès au repository devrait s'effectuer à travers le service, déjà disponible via la classe Controller abstrait (et non directement depuis le controller) : application des patterns

```java
@RestController
@RequestMapping("/api/utilisateurs")
public class UtilisateurController extends AUtilisateurController {

    /**
     * LOGGER
     */
    private static final Log LOG = LogFactory.getLog(UtilisateurController.class);

    private UtilisateurRepository repository;
```

AUDIT DE L'EXISTANT

Classes

Existence de classe Abstraite « **AService** » qui doit être étendue par les services et qui offre accès à des méthodes génériques (accès aux données…)

- Ceci devrait être généralisé sur tous les services

```java
/**
 * Generic abstract class for services using {@link CrudRepository}
 *
 * @author slerouvillois
 * @version 1$ 29/10/2019$
 */
public abstract class AService<T, R extends CrudRepository<T, Long>> {

    protected R repository;

    public AService(R repository) {
        this.repository = repository;
    }
```

```java
@Service
public class EditionService {
    private static final int FILENAME_LENGTH = 8;

    @Autowired
    private EditionRepository repository;
```

```java
@Service
public class DemandeHebergementService extends AService<DemandeHebergement, DemandeHebergementRepository> {

    private TerritoireService territoireService;
```

AUDIT DE L'EXISTANT

Méthodes

- Détection des méthodes avec multiples points de sortie (Multiple Return)
  - Plus de points de sortie augmentent la complexité, et réduit la maintenabilité des programmes(refactoring, debug, tests)

```
public String updatePasswordWithOldPassword(Stri
    if (StringUtils.isBlank(login) || StringUtil
        return USER_NOT_FOUND;
    }

    // encode the passwords already for the next
    String encodedMotDePasse = this.passwordEncc
    String encodedExistingMotDePasse = this.pass

    // validate the password
    String validationErrorCode = this.validatePa
    if (validationErrorCode != null) {
        return validationErrorCode;
    }

    // find the user by login & password
    List<Utilisateur> users = this.repository.fi
    Utilisateur user = (users == null) || (users

    // incorrect login or password
    if (user == null) {
        return USER_NOT_FOUND;
    }

    // save the new password
    user.setMotDePasseToken(null);
    this.updatePassword(user, encodedMotDePasse)

    return "OK";
}
```

AUDIT DE L'EXISTANT

Méthodes (Transactional)

Bonne gestion des transactions en général

```
@Transactional
public DemandeInsertion accepterOrientation(Long demandeId, Long idGroupesOrientation, LocalDateTime dateEntreePrevueParsed) {
    DemandeInsertion demandeInsertion = get(demandeId);
    if (demandeInsertion == null) {
        throw new javax.validation.ValidationException("La demande ayant pour id: " + demandeId + " n'existe pas.");
    } else if (demandeInsertion.getGroupesOrientation() == null) {
```

```
public void delete(Long id) {
    Fiche fiche = get(id);
    delete(fiche);
}

/**
 * Supprime la fiche, les liens avec la personne
 * Si la fiche n'est pas une fiche de type {@link Typ
 * mais {@link TypeFiche#Personne}, la {@link Personn
 * Si la personne supprimée est dans un groupe qui se
 * on supprime le groupe également.
 * */
@Transactional
public void delete(Fiche fiche) {
    switch (fiche.getTypefiche()) {
```

AUDIT DE L'EXISTANT

Méthodes (Transactional)

```
@Caching(evict = {
        @CacheEvict(cacheNames = "utilisateurs", key="#utilisateur.id"),
        @CacheEvict(cacheNames = "utilisateurs", key="#utilisateur.login")
})
public  Utilisateur save(Utilisateur utilisateur) {
    return this.repository.save(utilisateur);
}

public void deleteInfosUtilisateurMaraudeByIdUtilisateur(Long idUtilisateur) {
    Utilisateur utilisateur = repository.findOne(idUtilisateur);
    UtilisateurMaraude maraude = utilisateur.getMaraude();
    if (maraude != null) {
        utilisateurMaraudeRepository.delete(maraude.getId());
    } else {
        throw new ValidationException("L'utilisateur n'a pas d'informations maraude.");
    }
}
```

AUDIT DE L'EXISTANT

Méthodes (Complexité) : extractions

```
public BoolQueryBuilder constructSearchQuery(List<DemandeInsertionSearchParams> paramsBlocs, Utilisateur utilisateur) {
    BoolQueryBuilder query = QueryBuilders.boolQuery();

    // Vérification si un des blocs a un filtre sur le transfert vers un autre SIAO
    // Si c'est le cas, on doit filtrer par rapport aux territoires de l'utilisateur dans les autres blocs
    // Dans celui où le filtre est sélectionné, on ne doit pas faire de filtre sur le territoire
    Boolean transfertSIAOFiltre = !paramsBlocs.stream().filter(paramBloc -> paramBloc.getDemandeTransfereSIAO() != null && paramBloc.

    paramsBlocs.forEach(paramsBloc -> {
        BoolQueryBuilder queryBloc = constructBlocQuery(paramsBloc, utilisateur, transfertSIAOFiltre);
        query.should(queryBloc);
        query.minimumShouldMatch(1);

    });
```

```
private void constructPrecoBlocQuery(DemandeInsertionSearchParams paramsBloc, BoolQueryBuilder queryBloc) {

    addFilter(queryBloc, NB_PRECO_PA, paramsBloc.getNbPrecoPA());
    addFilter(queryBloc, PRECO_PA_DISPOSITIF, paramsBloc.getPrecoPaDispositifsIds());
    addFilter(queryBloc, PRECO_PA_TYPE_ETAB_1, paramsBloc.getPrecoPaTypeEtab1Ids());
    addFilter(queryBloc, PRECO_PA_TYPE_ETAB_2, paramsBloc.getPrecoPaTypeEtab2Ids());
    addFilter(queryBloc, PRECO_PA_TYPE_ETAB_3, paramsBloc.getPrecoPaTypeEtab3Ids());
    addFilter(queryBloc, PRECO_PA_STRUCT, paramsBloc.getPrecoPaStructIds());
    addFilter(queryBloc, PRECO_PA_CONF_PHYSIQUE, paramsBloc.getPrecoPaConfPhysiqueIds());
    addFilter(queryBloc, PRECO_PA_ZONE_GEO, paramsBloc.getPrecoPaZoneGeoIds());
    addFilter(queryBloc, PRECO_PA_TYPE_PLACE, paramsBloc.getPrecoPaTypePlaceIds());
    addFilter(queryBloc, PRECO_PA_CATEG_PLACE, paramsBloc.getPrecoPaCategPlaceIds());
    addFilter(queryBloc, PRECO_PA_RESTAURATION, paramsBloc.getPrecoPaRestaurationIds());
    addFilter(queryBloc, PRECO_PA_ACC, paramsBloc.getPrecoPaInfoPlaceIds());
    addFilter(queryBloc, PRECO_PA_INFO_PLACE, paramsBloc.getPrecoPaInfoPlaceIds());
    addFilter(queryBloc, PRECO_PA_PUBLIC_ACC, paramsBloc.getPrecoPaPublicAccIds());

    addFilter(queryBloc, NB_PRECO_SIAO, paramsBloc.getNbPrecoSIAO());
    addFilter(queryBloc, PRECO_SIAO_DISPOSITIF, paramsBloc.getPrecoSiaoDispositifsIds());
    addFilter(queryBloc, PRECO_SIAO_TYPE_ETAB_1, paramsBloc.getPrecoSiaoTypeEtab1Ids());
    addFilter(queryBloc, PRECO_SIAO_TYPE_ETAB_2, paramsBloc.getPrecoSiaoTypeEtab2Ids());
    addFilter(queryBloc, PRECO_SIAO_TYPE_ETAB_3, paramsBloc.getPrecoSiaoTypeEtab3Ids());
    addFilter(queryBloc, PRECO_SIAO_TYPE_PLACE, paramsBloc.getPrecoSiaoTypePlaceIds());
    addFilter(queryBloc, PRECO_SIAO_STRUCT, paramsBloc.getPrecoSiaoStructIds());
    addFilter(queryBloc, PRECO_SIAO_CONF_PHYSIQUE, paramsBloc.getPrecoSiaoConfPhysiqueIds());
    addFilter(queryBloc, PRECO_SIAO_ZONE_GEO, paramsBloc.getPrecoSiaoZoneGeoIds());
    addFilter(queryBloc, PRECO_SIAO_TYPE_PLACE, paramsBloc.getPrecoSiaoTypePlaceIds());
    addFilter(queryBloc, PRECO_SIAO_CATEG_PLACE, paramsBloc.getPrecoSiaoCategPlaceIds());
    addFilter(queryBloc, PRECO_SIAO_RESTAURATION, paramsBloc.getPrecoSiaoRestaurationIds());
    addFilter(queryBloc, PRECO_SIAO_ACC, paramsBloc.getPrecoSiaoAccIds());
    addFilter(queryBloc, PRECO_SIAO_INFO_PLACE, paramsBloc.getPrecoSiaoInfoPlaceIds());
    addFilter(queryBloc, PRECO_SIAO_PUBLIC_ACC, paramsBloc.getPrecoSiaoPublicAccIds());
}

private void constructDatesBlocQuery(DemandeInsertionSearchParams paramsBloc, BoolQueryBuilder queryBloc) {
    addFilterRange(queryBloc, PERS_DATE_CREATIONS, paramsBloc.getDateCreationPersonneMin(), paramsBloc.getDateCreationPersonneMax());
    addFilterRange(queryBloc, DATE_TRANSMISSION_SIAO, paramsBloc.getDateTransmissionMin(), paramsBloc.getDateTransmissionMax());
    addFilterRange(queryBloc, DATE_TRANSMISSION_INITIALE_SIAO, paramsBloc.getDateTransmissionInitialeMin(), paramsBloc.getDateTransmissionInitialeMax());
    addFilterRange(queryBloc, DATE_ENTREE, paramsBloc.getDateEntreeMin(), paramsBloc.getDateEntreeMax());
    addFilterRange(queryBloc, DATE_SORTIE, paramsBloc.getDateSortieMin(), paramsBloc.getDateSortieMax());
    addFilterRange(queryBloc, DATE_ENTREE_PREVUE, paramsBloc.getDateEntreePrevueMin(), paramsBloc.getDateEntreePrevueMax());
    addFilterRange(queryBloc, DATE_SORTIE_PREVUE, paramsBloc.getDateSortiePrevueMin(), paramsBloc.getDateSortiePrevueMax());
    addFilterRange(queryBloc, DATE_MODIFICATION, paramsBloc.getDateModificationMin(), paramsBloc.getDateModificationMax());
    addFilterRange(queryBloc, DATE_ANNULATION, paramsBloc.getDateAnnulationMin(), paramsBloc.getDateAnnulationMax());
    addFilterRange(queryBloc, DATE_REPONSE, paramsBloc.getDateReponseMin(), paramsBloc.getDateReponseMax());
    addFilterRange(queryBloc, DEM_LA_DATE_INSCRIPTION, paramsBloc.getDateInscriptionLAMin(), paramsBloc.getDateInscriptionLAMax());
    addFilterRange(queryBloc, DEM_LA_DATE_DESINSCRIPTION, paramsBloc.getDateDesinscriptionLAMin(), paramsBloc.getDateDesinscriptionLAMax());
    addFilterRange(queryBloc, DEM_DISP_DATE_INSCRIPTION, paramsBloc.getDateInscriptionDispMin(), paramsBloc.getDateInscriptionDispMax());
    addFilterRange(queryBloc, DEM_DISP_DATE_DESINSCRIPTION, paramsBloc.getDateDesinscriptionDispMin(), paramsBloc.getDateDesinscriptionDispMax());
    addFilterRange(queryBloc, DEM_COMMISSION_DATE, paramsBloc.getDateCommissionMin(), paramsBloc.getDateCommissionMax());
    addFilterRange(queryBloc, DATE_REFUS, paramsBloc.getDateRefusMin(), paramsBloc.getDateRefusMax());
}
```

AUDIT DE L'EXISTANT

Duplication de code

Les Taux de duplication remontées par Sonar pour l'applications :

	SIAO
Taux Duplication	6,7%
Lignes dupliquées	16460

- SIAO a un taux moyen de 6,7% avec 16460 lignes dupliquées (6% recommandé comme taux max)
- Privilégier l'utilisation des classes Helpers/Util (DateUtil, ExportUtil…) déjà existantes

AUDIT DE L'EXISTANT

Commentaires

Tableau récapitulatif des taux de javadoc pour chaque module

siao-115-batch	1.9%
siao-115-core	8.6%
siao-115-extraction	0.9%
siao-115-indexer	4.3%
siao-115-web/src	4.8%

AUDIT DE L'EXISTANT

Commentaires

- Présence de Commentaires javadoc non mis à jour (paramètres, description…)
  - Changer les paramètres des méthodes sans mettre à jour la javadoc a un impact sur la compréhension de la méthode et de son objectif, et peut porter confusion aux développeurs.
- Présence de la notion « je m'exprime » (express yourself) dans les commentaires dans certaines parties :
  - Il est recommandé de rendre le code lisible au lieu d'ajouter un commentaire explicatif, car le lecteur cherchera à comprendre doublement.

AUDIT DE L'EXISTANT

Commentaires

- Présence de commentaires avec TODO :

```
@RequestMapping(value = "/uploadLogo", method = RequestMethod.POST)
@Secured({Role.Secured.ROLE_ADMIN_FONCTIONNEL_115, Role.Secured.ROLE_ADMIN_TERRITORIAL})
public ResponseEntity<FileSaveSuccessResult> handleFileUpload(@RequestParam("file") MultipartFile file) {
    // validation
    if ((file == null) || file.isEmpty() || (file.getOriginalFilename() == null) || (file.getOriginalFilename().length() == 0) ||

        // MultipartException exception = (MultipartException) this.request.getAttribute("MultipartException");
        // TODO: I AM HERE: this bit doesn't work, the connection always gets aborted
        return new ResponseEntity<>(HttpStatus.OK);
    }
```

```
 * <p>
 * TODO : Faire des tests unitaires.
 *
 * @param pListeDemandes La liste des demande dont on doit récupérer les données du PHRH.
 */
public void fetchPhrhData(final Fiche pFiche, final List<DemandeHebergement> pListeDemandes) {
    if (BooleanUtils.isTrue(phrhService.isPhrhActif())) {
        List<DemandeHebergement> newDemandes = new ArrayList<>();
        List<DemandeHebergement> demandesToDelete = new ArrayList<>();

        pListeDemandes.stream().forEach((DemandeHebergement demande) -> {
            fetchMotifApec(demande);
            /*
             * La méthode "fetchChangementHotelData" doit être appelée à la fin du stream
             * "forEach" car elle est susceptible de supprimer la demande 115 initiale, et
             * donc les traitements réalisés après ne seraient pas conservés.
             *
             * Comme la méthode clone la demande avant de l'enregistrer pour suppression,
             * il faut réaliser tous les traitements avant.
             * TODO : Trouver un moyen d'éviter ce comportement
             */
            fetchChangementHotelData(pFiche, demande, newDemandes, demandesToDelete);
        });
```

AUDIT DE L'EXISTANT

Commentaires

- Présence de commentaires avec TODO :

```
// TODO Solution temporaire pour l'ajout de personne sur demande en présence, à remplacer par Liste de personne pour la v1.2
/**
 * Liste des personnes par orientation
 */
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "por_dgp_id")
@JsonView({View.DemandeInsertion.class, View.GroupesOrientation.class})
private Set<OrientationPersonnes> orientationPersonnes;
```

```
/**
 * Permet de construire une chaîne de caractères servant aux procédures stockées
 * remontant les nombres d'alertes du cycle en cours.<br/>
 * La chaîne construite correspond à la liste des ID des structures séparés par
 * des virgules, sans brackets. TODO voir si il y a une autre méthode.
 *
 * @param structureFilter La liste des ID des structures.
 * @return La chaîne de caractères prête à être employée par la procédure
 * stockée de PostgreSQL.
 */
private String buildStrStructureIds(final List<Long> structureFilter) {
    // Pour pouvoir utiliser la procédure stockée avec les ids des structures
    // il faut transformer le List<Long> en string sans les brackets.
    // Puis dans la procédurer il faut séparer les ids avec les virgules
    String result = "";
    if (structureFilter != null) {
        Long[] structureIds = new Long[structureFilter.size()];
        structureIds = structureFilter.toArray(structureIds);
        result = Arrays.toString(structureIds).replace("[", "").replace("]", "");
```

AUDIT DE L'EXISTANT

Nommage

- Respect des nommages du standard java :
  - PascalCase pour les classes, camelCase pour les variables, SNAKE_CASE pour les constantes
- Utilisation du français et anglais :
  - Certaines méthodes ont comme nommage update*() et miseAJour*() dans la même classe.

AUDIT DE L'EXISTANT

Gestion des traces - Logs

- Utilisation de l'outil log4j pour la gestion des logs
- Détection d'un service pour tracer quelques évènements métiers (table audit néanmoins vide)
- Présence de niveaux de log inadéquats avec le contexte:
  - Au niveau des logs, il est impérativement recommandé de bien logguer les informations suivant le bon contexte en cas d'erreur, utiliser le contexte erreur au lieu de info, pour bien cibler les messages liés aux problèmes.

```
String siteLink = UriComponentsBuilder.fromHttpUrl(this.baseUrl).build().toString();
try {
    this.buildEmailForgotPasswordEmailUnknownForAdmins(recipients, nom, prenom, telephone, email, login, siteLink, roles);
} catch (Exception ex) {
    LOG.info(ex.getMessage(), ex);
}

return false;
```

AUDIT DE L'EXISTANT

Gestion des traces - Logs

- Présence de traitements de bout en bout sans gestion de logs :
  - Sans gestion de logs, il est difficile de suivre et de tracer l'appel et l'exécution des traitements

```java
@RestController
@RequestMapping("/api/aides")
public class AideController {

    private AideService service;

    private AideRepository repository;

    private UtilisateurService utilisateurService;
```

```java
@RestController
@RequestMapping("/api/listeAttente")
public class ListeAttenteController {

    @Autowired
    private SiaoService siaoService;

    @Autowired
    private ListeAttenteService listeAttenteService;
```

AUDIT DE L'EXISTANT

Gestion des exceptions

- SIAO dispose d'une gestion des exceptions personnalisées :

```java
@ResponseStatus(value=HttpStatus.UNAVAILABLE_FOR_LEGAL_REASONS)
public class UnavailableForLegalReasonsException extends RuntimeException {

    /**
     * serial Version UID.
     */
    private static final long serialVersionUID = -3212944158305286489L;

    public UnavailableForLegalReasonsException() {}

    public UnavailableForLegalReasonsException(String message) {
        super(message);
    }
```

- Cette pratique n'est pas généralisée

AUDIT DE L'EXISTANT

Gestion des exceptions

- Présence de traitements sans utilisation des fichiers de propriétés
  - Des exceptions avec des messages écrits en dur dans le code sans utilisation du mécanismes clé/valeur (fichiers de messages)

```
public List<DemandeInsertion> retirerPersonneEnMasse(final String dateSortie, final
                                                      final List<DemandeInsertion> d
    List<DemandeInsertion> demandeInsertionsSaved = new ArrayList<>();
    for (DemandeInsertion demandeInsertion : demandeInsertions) {
        try {
            demandeInsertionsSaved.add(this.sortiePersonnes(demandeInsertion.getId(
        } catch (Exception e) {
            LOG.error(MAIL_ERROR_LOG, e);
        }
    }
    return demandeInsertionsSaved;
```

- Détection de gestion d'exceptions sans remontée d'erreurs:
  - En cas d'erreur, il est recommandé de les remonter pour être traitées et présentées aux utilisateurs.

```
static boolean set(Object object, String fieldName, Object fieldValue) {
    Class<?> clazz = object.getClass();
    while (clazz != null) {
        try {
            Field field = clazz.getDeclaredField(fieldName);
            field.setAccessible(true);
            field.set(object, fieldValue);
            return true;
        } catch (NoSuchFieldException e) {
            clazz = clazz.getSuperclass();
            LoggerFactory.getLogger(clazz).error("Le champ suivant n'existe pas : " + fieldName, e);
        } catch (Exception e) {
            throw new IllegalStateException(e);
        }
    }
```

AUDIT DE L'EXISTANT

Gestion des exceptions

- Détection de gestion d'exceptions avec la classe Exception et non des exceptions spécifiques
  - Bien cibler les exceptions spécifiques permet de savoir quels types d'erreurs peuvent se produire et de mieux les gérer.
- La non-propagation de certaines exceptions avec des messages explicites

```
            this.mailService.buildEmailAndSend(Collections.singletonList(entry.getKey()), objet, corpsMails, null, null, null);
        } catch (Exception e) {
            exceptions.add(e);
            LOG.error(e.getMessage());
        }
    }
    if (!exceptions.isEmpty()) {
        throw new Exception("Une erreur est survenu lors de l'envoi des mails. Vérifiez les logs à la date " + LocalDateTime.now().toString());
    }
}
```

```
public InputStream genererCertificatPEC(Utilisateur utilisateur, Fiche fiche, DemandeInsertion demande) throws IO

    XWPFDocument documentGlobal = new XWPFDocument();
    ByteArrayOutputStream out = new ByteArrayOutputStream();

    addLogoSIAOImageToWordDocument(demande.getSiao(), documentGlobal);
    GroupesOrientation orientationPresence = demande.getGroupesOrientation().stream()
            .filter(ori -> ori.getStatut() == StatutAffectation.PRESENCE).collect(Collectors.toList()).get(0);
    createWordFilePerson(utilisateur, documentGlobal, fiche, demande, orientationPresence);
    if (utilisateur.getCurrentRole().equals(Role.ROLE_SIAO_INSERTION)) {
        try {
            addSignatureSIAOImageToWordDocument(demande.getSiao(), documentGlobal);
        } catch (Exception e) {};
    }
    documentGlobal.write(out);
```

AUDIT DE L'EXISTANT

Gestion des exceptions

```
private void enrichDataSyploInsetion(Map<String, Object> doc, Map<Integer, Map<String, Object>> infosPersonnes,
                                     List<Integer> personnesIds) {
    for (ColonnesExtractionESInsertion col : ColonnesExtractionESInsertion.getColonnesSyplo()) {
        List syplo = (List) doc.get(col.getColonne().toLowerCase());
        try {
            if (CollectionUtils.isNotEmpty(syplo)) {
                IntStream.range(0, personnesIds.size()).forEach(index -> infosPersonnes.get(personnesIds.get(index)).p
                doc.put(col.getColonne(), syplo.get(0));
            }
        }
        catch (Exception e) {
            LOG.error("{} - {} - {} - {} . Exception : ", syplo, personnesIds, infosPersonnes, doc.get(DEM_ID), e);
        }
    }
}
```

```
try {
    //check des triggers
    List<String> tablesWithoutTriggers = repository.checkSyncedTables();
    if (!tablesWithoutTriggers.isEmpty()) {
        LOG.warn("Les tables suivantes n'ont pas de trigger de synchro : {}", tablesWithoutTriggers);
    }

    //restart des taches en cours suite à arret/crash
    int restarted = repository.restartPendingTasks();
    if (restarted > 0) {
        LOG.warn("{} taches au statut en cours sont redémarrées", restarted);
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

```
String mappingDef = elasticMapping.get(e.getKey()).toString();
if (mappingDef.contains("date") && value.toString().length() == 10) {
    //c'est une date sans heure.
    try {
        value = LocalDate.parse(value.toString()).atStartOfDay();
    } catch (Exception ex) {
        LOG.error("", ex);
    }
```

AUDIT DE L'EXISTANT

Gestion des exceptions

```java
/**
 * Fusionne la liste de doublons passée en paramètre
 *
 * @param recapDoublonDTO
 * @throws Exception
 */
@RequestMapping(value = "/fusionnerDoublons", method = RequestMethod.POST)
@JsonView(View.Structure.class)
@Secured({Role.Secured.ROLE_ADMIN_FONCTIONNEL_115, Role.Secured.ROLE_ADMIN_TERRITORIAL})
public void fusionnerDoublons(@RequestBody RecapDoublonStructureDTO recapDoublonDTO) throws Exception {
    try {
        fusionStructuresService.fusionnerDoublons(recapDoublonDTO.getStructureMaitre(), recapDoublonDTO.getStructuresAFusionner());
    } catch (Exception e) {
        LOG.error("Erreur lors de la fusion structures");
        throw e;
    }
}
```

AUDIT DE L'EXISTANT

Méthode de production des éditions

- Absence d'utilisation des templates (envoie de mails), nécessitant une relivraison en cas de modification

```
private final static String SIGNATURE_RGPD = "<em>« Les informations que vous allez saisir ou consulter sont des données à caractère personnel " +
        "protégées par le règlement général de la protection des données (RGPD) du 25 mai 2018 et la loi informatique et libertés (LIL). " +
        "Les usagers disposent d'un droit d'accès et de rectification, de portabilité et voire d'effacement des données. Pour exercer ce " +
        "droit, les usagers doivent exprimer leurs souhaits auprès d'un travailleur social, du SIAO ou auprès de la DGCS (directement en " +
        "envoyant un mail à dgcs-siao@social.gouv.fr). »</em>";

public final static String RGPD = "<em>« Les personnels des structures habilitées dans le SI SIAO sont soumis au secret professionnel " +
        "et au principe de confidentialité. Le régime d'habilitation par l'administrateur du SIAO doit être respecté. La transmission " +
        "des habilitations est prohibée. L'emploi de mots de passe est indispensable et doit respecter les normes imposées par la CNIL. " +
        "Ils doivent être changés de manière régulière. Pour des raisons de sécurité, les impressions sont déconseillées à l'exception " +
        "d'une transmission au bénéficiaire ou à des fins d'analyse des demandes. Il est fortement recommandé de détruites après usage " +
        "l'ensemble des impressions contenant des données nominatives. Le transfert de ces données à des tiers non autorisés est " +
        "strictement interdit et engage votre responsabilité. »</em>";
```

```
String objet = "SI SIAO - Réactivation de votre compte.";
StringBuilder corpsMail = new StringBuilder();
corpsMail.append("Bonjour " + utilisateur.getPrenom() + " " + utilisateur.getNom() + ",")
        .append(retourChariot)
        .append(retourChariot)
        .append("Vous avez demandé la réinitialisation du mot de passe pour le compte " + utilisateur.getLogin() + ".")
        .append(retourChariot)
        .append("En cliquant sur le lient suivant, vous allez être redirigé vers une page afin de saisir un nouveau mot de
        .append(retourChariot)
        .append("L'application va vous demander de créer votre mot de passe à la première connexion.")
        .append(retourChariot)
        .append(retourChariot)
        .append("Bien cordialement,")
        .append(retourChariot)
        .append(retourChariot)
        .append("L'équipe SI SIAO");

this.mailService.buildEmailAndSend(destinataires, objet, corpsMail, MailService.RGPD, null, null);
```

AUDIT DE L'EXISTANT

Java 8

Java 8 a fournit de nouvelles fonctionnalités rendant le code plus souple et plus performant.

Le code de SIAO ne contient pas suffisamment de traitements avec utilisation de ces fonctionnalités, notamment sur la partie streams API, lambda et la gestion des dates

1.2

RECOMMANDATIONS

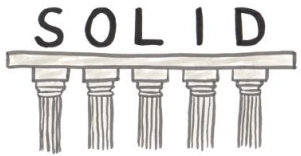
RECOMMANDATIONS

Classes

«La première règle des classes est qu'elles doivent être petites. La deuxième règle des classes est qu'elles doivent être plus petites que cela. ... Avec les fonctions, nous avons mesuré la taille en comptant les lignes physiques. Avec les classes, nous utilisons une mesure différente. Nous comptons les responsabilités. [Chapitre 10, page 136] ” Code Clean

Suivant le principe SRP,

- Il est recommandé de limiter les responsabilités des classes, ce principe implique que le code au sein d’une classe ne doit avoir qu’une seule responsabilité. Cela permet d’organiser le code et obtenir un code plus maintenable,
- Il est recommandé d’utiliser les ateliers CRC(Classes, Responsabilité, CSollaborateurs)
- Il est recommandé de vérifier l’utilisation des déclarations et imports



RECOMMANDATIONS

Méthodes

- Il est recommandé d'avoir une moyenne de 30 lignes par méthode (Rule of 30)
- Il est recommandé d'avoir un taux de complexité cyclomatique inférieur à 10
- Il est recommandé de bien nommer les méthodes (français ou anglais)
- Il est recommandé d'avoir un nombre de paramètres < 4 et éviter les méthodes dites polyadic

RECOMMANDATIONS

Duplication

- Au niveau des IDE, il existe des options comme Extract Method qui aident à extraire les traitements dupliqués et de les utiliser dans des endroits cibles pour réduire les duplications.

- Planifier un temps pour factoriser le code et de regrouper les duplications, cela permet de réduire la complexité du code et de minimiser les risques de bugs et de régressions

- Utiliser les classes Helper, sous forme de classe d'assistance qui englobent des traitements utilisés par les autres classes.

RECOMMANDATIONS

Commentaire

- Il est recommandé d'ajouter systématiquement du javadoc pour bien décrire les responsabilités de chaque méthode/classe.
- Il est recommandé de mettre à jour la javadoc, en cas de changement des noms ou du nombre de paramètres.
- Eviter les commentaires (Express yourself) et refactoriser le code pour plus de lisibilité

RECOMMANDATIONS

Nommages

- Il est recommandé de choisir une langue (Français ou Anglais) dans les nommages de classes et méthodes.

RECOMMANDATIONS

Gestion des traces-Logs

Les logs jouent un rôle important dans la maintenance et le suivi des traitements effectués dans l'application

- Il est recommandé de tracer les informations suivant leur bon contexte :
  - Fatal: Toute erreur qui force un arrêt du service ou de l'application pour empêcher la perte de données
  - Error : Toute erreur fatale à l'opération, mais pas au service ou à l'application
  - warn : Peut potentiellement causer des problèmes d'application
  - Info : Informations généralement utiles à consigner (démarrage / arrêt du service, configuration…)
  - Debug: Désigne les événements d'information détaillés qui sont les plus utiles pour déboguer une application.
  - Trace: Des événements d'information plus détaillés que le DEBUG

- Eviter les system.out.println et les printStackTrace et utiliser les logs
- Il est recommandé d'ajouter la gestion des traces (logs) au niveau de toutes les classes pour tracer et suivre les traitements

RECOMMANDATIONS

Gestion des exceptions

Il est recommandé de :

- Créer et utiliser des classes d'exceptions personnalisées pour simplifier la gestion des exception et simplifier les traitements
- Eviter la gestion des exceptions avec NullPointerException
- Utiliser des exceptions spécifiques au lieu de Exception
- Remonter l'exception à l'utilisateur et de ne pas se contenter de la logguer
- Garder un chainage des exceptions afin de connaitre les causes des erreurs
- Utiliser le mécanisme (clé,valeur) du fichier de propriétés qui contient la liste des messages d'erreurs

RECOMMANDATIONS

Tests

Les tests unitaires ont pour but de tester le bon fonctionnement d'une partie précise d'un programme.

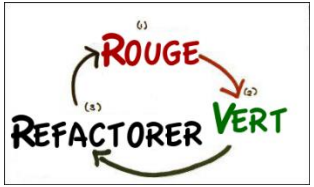
- Il est recommandé d'appliquer les règles F.I.R.S.T pour avoir de bons tests unitaires :
  - Fast : Les tests doivent être rapides, afin de pouvoir les lancer a chaque occasion
  - Isolated : Les tests ne doivent pas dépendre des uns des autres.
  - Repeatable : Les tests doivent être répétables dans n'importe quel environnement.[prod,qal,local]
  - Self-Validating : Les tests doivent avoir un résultat Booléen, [réussir ou échouer] pas autre moyen de validation
  - Timely :Les tests sont écrits au bon moment, juste avant le code de production qui les fait passer

L'approche TDD :

1-Vous devez écrire un test qui échoue avant d'écrire tout code de production

2-Vous ne devez pas écrire plus d'un test suffisant pour échouer, ou qui échouera à la compilation

3-Vous ne devez pas écrire plus de code que nécessaire pour faire passer le test en cours



RECOMMANDATIONS

Méthode des éditions

Il est recommandé d'utiliser des templates afin de mieux gérer les éditions et envoies de mail, et éviter ainsi la modification directement dans le code

MERCI DE VOTRE ATTENTION

WE EMPOWER
YOUR DIGITAL WORLD

www.open.global