**TP Génie Logiciel, 3IF, 2018/2019**

**Étape 4**

*Historique de modifications par rapport à l’étape 3 :*

* *Fin de la reprise des spécifications entamée dans l’étape 3*
* *Fin et relecture du manuel utilisateur*
* *reprise des diagrammes*
* *ajout de tests de performance*
* *Relecture finale de tout le livrable*

1. Spécification des besoins et plan de tests de validation
   * Spécifications fonctionnelles de l’application :

* Lecture des fichiers

|  |  |
| --- | --- |
| Fonctionnalité F1 | Chargement des fichiers de données  void chargerFichiers(String urlCapt, String urlAttr, String urlMesures) |
| Description | Cette fonctionnalité est lancée à l’ouverture de l’application. Elle permet de stocker en mémoire dans l’application les données des fichiers .csv présents sur un serveur central. Ces fichiers contiennent la liste des capteurs, la liste de attributs (ou types de mesures) et les mesures. |
| Données nécessaires | Chemin d’accès de chaque fichier .csv (fichier Capteurs, attributs et mesures). |
| Pré conditions | Les fichiers doivent être conformes au format spécifié dans le sujet du TP (c’est-à-dire chaînes de caractères distinctes séparées par un « ; » par exemple). |
| Post conditions | Le contenu des fichiers est mis en mémoire, il est possible d’y accéder et d’utiliser ces données. |
| Données en sortie | / |
| Tests | Cas d’exception :   * Test d’ouverture d’un fichier vide : exception avec un message d’erreur indiquant que le fichier ne contient rien * Test d’ouverture d’un fichier inexistant ou sans droit de lecture : exception avec un message d’erreur indiquant si le fichier est introuvable ou si une erreur survient lors de son ouverture/lecture   Autre : Test avec des fichiers conformes |

Désormais on considère que les fichiers de données ont été correctement chargés afin de permettre le bon fonctionnement de l’application (on ne considère pas de cas d’erreur où il manque un/les fichiers dans la suite des fonctionnalités). Cette précondition (fichiers mis en mémoire correctement) concerne toutes les fonctionnalités ci-après.

* Gestion des capteurs

|  |  |
| --- | --- |
| Fonctionnalité F2 | Consulter la liste des capteurs  String consulterCapteurs() |
| Description | Cette fonctionnalité affiche la liste des tous les capteurs |
| Données nécessaires | Aucune |
| Pré conditions | Des capteurs sont renseignés, il n’y a pas aucun capteur |
| Post conditions | La liste des capteurs s’affiche, le fichier n’est pas modifié |
| Données en sortie | Tableau de chaînes de caractères avec l’ID de chaque capteur, ses coordonnées (latitude et longitude) et sa description |
| Tests | Cas d’exception :   * s’il n’y a aucun attribut dans le fichier d’origine, s’il est vide * si le fichier est inaccessible en lecture   Autre : avec un fichier conforme et affichage de la liste |

|  |  |
| --- | --- |
| Fonctionnalité F3 | Ajouter un capteur  bool ajouterCapteur(Capteur c) |
| Description | Cette fonctionnalité permet d’ajouter un capteur dans le circuit, il est ajouté aux structures de données correspondantes et est rendu fonctionnel (on peut récupérer ses données). |
| Données nécessaires | SensorID, Latitude, Longitude et Description du capteur ajouté |
| Pré conditions | Le SensorID ne doit pas être déjà attribué, les coordonnées doivent être cohérentes. |
| Post conditions | Le capteur est présent dans les structures de données en mémoire de l’application (mais le fichier .csv de base n’est pas modifié). |
| Données en sortie | Boolean (*true* si l’ajout a fonctionné, *false* sinon.) |
| Tests | Cas d’exception :   * Indiquer un ID déjà existant * Vérifier que les latitude et longitude du capteur sont possibles * Il est possible de consulter les données du capteur, il est présent en mémoire et fonctionnel |

|  |  |
| --- | --- |
| Fonctionnalité F4 | Supprimer un capteur  bool supprimerCapteur(String sensorId) |
| Description | Cette fonctionnalité permet de supprimer un capteur du circuit, il est supprimé des structures de données correspondantes et n’est plus fonctionnel (on peut plus récupérer ses données ou le retrouver dans la liste des capteurs). |
| Données nécessaires | SensorID |
| Pré conditions | Le capteur doit exister, c’est-à-dire que le SensorID doit correspondre à un capteur existant. |
| Post conditions | Le capteur n’est plus présent dans les structures de données en mémoire de l’application. |
| Données en sortie | Boolean : *true* si ça a fonctionné, *false* sinon. |
| Tests | Cas d’exception : test de la suppression d’un capteur inexistant  Autre :   * Vérifier que le capteur est bien supprimé (impossible de le surveiller, de le trouver dans la liste des capteurs par exemple) * Tester qu’on arrive bien à récupérer les données dans le fichier data même si le capteur est supprimé |

|  |  |
| --- | --- |
| Fonctionnalité F4 | Surveiller un capteur  bool surveillerCapteur(String sensorId) |
| Description | Cette fonctionnalité permet de vérifier la fiabilité d’un capteur : données viables ou non (c’est-à-dire comprises dans des intervalles prédéfinis), dernière date d’actualisation (un capteur ne reste pas 2 jours sans envoyer d’informations). |
| Données nécessaires | SensorID |
| Pré conditions | Le capteur doit exister, c’est-à-dire que le SensorID doit correspondre à un capteur existant. |
| Post conditions | Aucune opération modifiant le capteur n’a été effectuée (le capteur est considéré *const* pour cette fonctionnalité). |
| Données en sortie | Boolean : *true* si le capteur est valide, *false* sinon. |
| Tests | Cas d’exception :   * Tester un capteur inexistant * Tester un capteur non fonctionnel (un capteur qui envoie des données aberrantes ou qui n’envoie plus de données depuis 2 jours)   Autre : Tester que tout fonctionne avec un capteur fonctionnel |

|  |  |
| --- | --- |
| Fonctionnalité F5 | Rechercher capteur  Capteur rechercherCapteur(int latitude, int longitude) |
| Description | Cette fonctionnalité permet de trouver l’ID du capteur le plus proche des coordonnées renseignées. |
| Données nécessaires | Latitude, longitude |
| Pré conditions | Les coordonnées doivent être possibles. |
| Post conditions | Aucune opération modifiant le capteur n’a été effectuée (le capteur est considéré *const* pour cette fonctionnalité). |
| Données en sortie | Capteur correspondant à la recherche |
| Tests | Cas d’exception : Tester sûr des coordonnées erronées  Autre :   * Tests avec 2 capteurs à même distance * Test avec un seul capteur proche |

|  |  |
| --- | --- |
| Fonctionnalité F6 | Capteurs similaires  list<Capteur> capteursSimilaires(String sensorId, int inter) |
| Description | Cette fonctionnalité permet de trouver les capteurs ayant un comportement similaire à celui du capteur donc l’ID est passé en paramètre. Un capteur est similaire à un autre si pour tous les attributs qu’ils ont en commun (un seul attribut peut suffire) toutes leurs valeurs enregistrées sont identiques, à l’intervalle de confiance donné près. |
| Données nécessaires | SensorID, intervalle de confiance |
| Pré conditions | L’intervalle de confiance renseigné est un entier |
| Post conditions | Aucune opération modifiant le(s) capteur(s) n’a été effectuée (le capteur est considéré *const* pour cette fonctionnalité). |
| Données en sortie | list de capteurs, contenant les capteurs similaires s’il y en a, rien sinon |
| Tests | Cas d’exception :   * sensorID inconnu * aucun capteur similaire trouvé   Autre :   * plusieurs capteurs similaires trouvés (beaucoup, avec un intervalle de confiance élevé) * peu de capteurs similaires trouvé (intervalle faible) * un seul capteur similaire renvoyé |

* Données capteurs

|  |  |
| --- | --- |
| Fonctionnalité F7 | Consulter liste des attributs  String consulterAttributs() |
| Description | Cette fonctionnalité permet de consulter le contenu du document avec la liste des tous les attributs |
| Données nécessaires | Aucune |
| Pré conditions | Des attributs sont renseignés dans le fichier, celui-ci n’est pas vide. |
| Post conditions | La liste des attributs s’affiche, le fichier n’est pas modifié. |
| Données en sortie | Tableau de chaînes de caractères avec l’ID de chaque attribut, son unité et sa description |
| Tests | Cas d’exception :   * s’il n’y a aucun attribut dans le fichier d’origine, s’il est vide * si le fichier est inaccessible en lecture   Autre : avec un fichier conforme et affichage de la liste |

|  |  |
| --- | --- |
| Fonctionnalité F8 | Consulter les données  String consulterDonnées() |
| Description | Cette fonctionnalité permet de consulter le contenu du document avec l’enregistrement des données |
| Données nécessaires | Aucune |
| Pré conditions | Des données sont renseignées dans le fichier, celui-ci n’est pas vide. |
| Post conditions | La liste des données s’affiche, le fichier n’est pas modifié. |
| Données en sortie | Tableau de chaînes de caractères contenant la date de mesure de chaque donnée, l’ID du capteur l’ayant réalisée, l’ID de l’attribut et sa valeur mesurée. |
| Tests | Cas d’exception :   * s’il n’y a aucun attribut dans le fichier d’origine, s’il est vide * si le fichier est inaccessible en lecture   Autre : avec un fichier conforme et affichage de la liste |

|  |  |
| --- | --- |
| Fonctionnalité F9 | Recueillir données similaires  list<Mesure> donnéesSimilaires(Mesure m, Date jour, int inter) |
| Description | Cette fonctionnalité permet de retourner toutes les données similaires à celle d’une mesure (valeur d’un attribut) pour un intervalle de confiance et une date donnés par l’utilisateur.  Algorithme : On parcourt toutes les mesures en mémoire en recherchant l’attribut. Si la valeur de l’attribut correspond à celle de la mesure passée en paramètre + ou – l’intervalle de confiance : on ajoute la Mesure à la liste de Mesures qui sera renvoyée. |
| Données nécessaires | Mesure, date, intervalle de confiance |
| Pré conditions | L’attribut existe pour le capteur donné à la date renseignée et possède des valeurs |
| Post conditions | Affichage des mesures similaires renvoyées, aucune modification n’est faite sûr des éléments. |
| Données en sortie | Liste des mesures similaires correspondant à la recherche. |
| Tests | Cas d’exception :   * Si sensorID inconnu * Si attributID n’existe pas pour le sensorID * Si pas de valeur renseignée   Autre :   * Si une seule mesure est renvoyée (intervalle de confiance faible) * Si beaucoup de valeurs sont renvoyées (intervalle de confiance élevé) |

|  |  |
| --- | --- |
| Fonctionnalité F9 | Consulter mesure(s)  list<Mesure> getMesureCapteur(int \* bitTab, String \* arg), l’algorithme correspondant à cette implémentation est décrit dans une partie ci-après avec les autres descriptions d’algorithmes. |
| Description | Cette fonctionnalité permet de trouver les caractéristiques de la qualité de l’air. Elle renvoie les mesures qui correspondent à une recherche dans la map des données, avec les paramètres suivants précisés ou non :   * + sensorID d’un capteur (seules les mesures du capteur sont prises en compte)   + intervalles de coordonnées, latitude et longitude de départ et latitude et longitude de fin (seules les mesures du ou des capteurs le plus proche ou compris dans l’intervalle sont prises en compte)   + intervalle de dates : précisé par une date de début et un e date de fin (seules les mesures comprises entre le jour de début et de fin donnés sont prises en compte). Les dates sont indiquées par 3 entiers : l’année (XXXX), le mois (1-12) et le jour (1-31)   + attributID d’un Attribut (seules mesures de l’attribut sont prises en compte)   Par défaut, si l’on ne précise aucun paramètre, toutes les mesures de tous les attributs pour tous les capteurs pour la date du jour actuel sont renvoyées. |
| Données nécessaires | Rien ou SensorID et/ou intervalle de coordonnées et/ou intervalle de dates et/ou intervalle d’horaires et/ou attributID |
| Pré conditions | S’ils sont renseignés : les longitudes et latitudes passées en paramètres ainsi que les dates doivent être cohérentes, le sensorID et l’attributID doivent exister, les horaires doivent être des entiers compris dans 0 et 23 |
| Post conditions | Affichage des mesures correspondantes à la recherche. Aucune modification n’est effectuée. |
| Données en sortie | Mesures répondant aux critères décrits ci-dessus |
| Tests | Cas d’exception :   * coordonnées impossibles. * sensorID inconnu * attributID inconnu * date inexistante * horaire renseigné non entier * lorsqu’aucune mesure ne correspond à la recherche   Autre :   * avec intervalle de coordonnées * avec intervalle de dates * avec intervalle d’horaires * avec sensorID * avec attributID * en combinant les différents éléments ci-dessus, pour des valeurs différentes (extrêmes et intermédiaires). On vérifie que les mesures retournées sont bien celles attendues. |

|  |  |
| --- | --- |
| Fonctionnalité F10 | Moyenne des valeurs d’un attribut d’un capteur  int moyenneValAttribut(Attribut att, String sensorId, Date début, Date fin = NULL) |
| Description | Cette fonctionnalité permet de calculer la moyenne des valeurs d’un attribut d’un capteur sûr une durée en jours donnée.  La durée est soit précisée par une date, auquel cas la moyenne est effectuée sur les mesures de ce jour-là, ou 2 dates : la moyenne est effectuée sur les mesures des jours compris dans l’intervalle. |
| Données nécessaires | SensorID, attributID et une ou deux dates |
| Pré conditions | La ou les dates doivent exister. Si 2 dates sont renseignées, la première doit être antérieure à la seconde.  SensorID et attributID doivent exister. |
| Post conditions | Affichage de la moyenne renvoyée. Aucune modification n’est effectuée. |
| Données en sortie | Entier correspondant à la valeur moyenne calculée |
| Tests | Cas d’exceptions :   * dates invaldies * sensorID inconnu * attributID inconnu * aucune mesure correspondante correspondante * avec une seule mesure correspondante   Autre :   * avec une date * avec un intervalle de dates * avec beaucoup de mesures correspondantes * avec peu de mesures correspondantes |

|  |  |
| --- | --- |
| Fonctionnalité F11 | Générer graph représentant les valeurs d’une liste de mesures  bool genererGraph(list <Mesure>) |
| Description | Cette fonctionnalité permet de générer un graph (graphique en nuage de points excel) représentant les valeurs d’une liste de mesures. Le graph est généré sous forme d’un fichier .xlsx dans le dossier où se trouve le fichier main. |
| Données nécessaires | liste de mesures |
| Pré conditions | la liste de mesures ne doit pas être vide et comporter des valeurs |
| Post conditions | Génération du fichier correspondant. |
| Données en sortie | Booléen valant *true* si l’opération a bien été effectuée, *false* en cas d’erreur. |
| Tests | Cas d’exceptions :   * dates invaldies * sensorID inconnu * attributID inconnu * aucune mesure correspondante correspondante * avec une seule mesure correspondante   Autre :   * avec une date * avec un intervalle de dates * avec beaucoup de mesures correspondantes * avec peu de mesures correspondantes |

* Alertes, seuil et decisions

|  |  |
| --- | --- |
| Fonctionnalité F12 | Calculer les données prévisionnelles  calculerDonneePrevisionelle (map<struct tm, map<string,double>>listeMesurebyDate, string lAttribut) |
| Description | Cette fonctionnalité permet d’estimer / de faire une prévision de la mesure au temps t+5 pour l’attribut d’un capteur |
| Données nécessaires | La structure de données ayant comme nœud principal le capteur, attributID |
| Pré conditions | Au moins 5 mesures doivent correspondre à l’attributID du capteur. Ces mesures permettent de calculer la mesure a t+5 grâce à un algorithme décrit dans la partie « Description d’algorithmes » ci-après. |
| Post conditions | Affichage d’un message d’erreur et demande de prendre une decision. Aucune modification n’est effectuée. |
| Données en sortie | Booléen |
| Tests | Cas d’exception :   * sensorID inconnu * attributID inconnu * pas suffisamment de mesures correspondantes (<5) pour l’algorithme   Autre :   * Tester que les données calculées soient cohérentes. * avec des mesures à valeurs croissantes * avec des mesures à valeurs décroissantes * avec des mesures à valeurs à peu près constantes |

|  |  |
| --- | --- |
| Fonctionnalité F13 | Donnée au-delà d’un seuil  bool valeurAuDelaSeuil(string attribut, double val) |
| Description | Cette fonctionnalité permet de déterminer si une valeur est au delà du seuil prédéfini |
| Données nécessaires | Id de l’attribut, et la valeur. |
| Pré conditions | AttributID doit exister. Le seuil est defini par l’indice ATMO. |
| Post conditions | Generer un message d’erreur si le bool est « true » |
| Données en sortie | Booléen true si valeur est au-delà du seuil sinon false (ou si l’attribut n’est pas reconnu). |
| Tests | Cas d’exception :   * attributID inconnu |

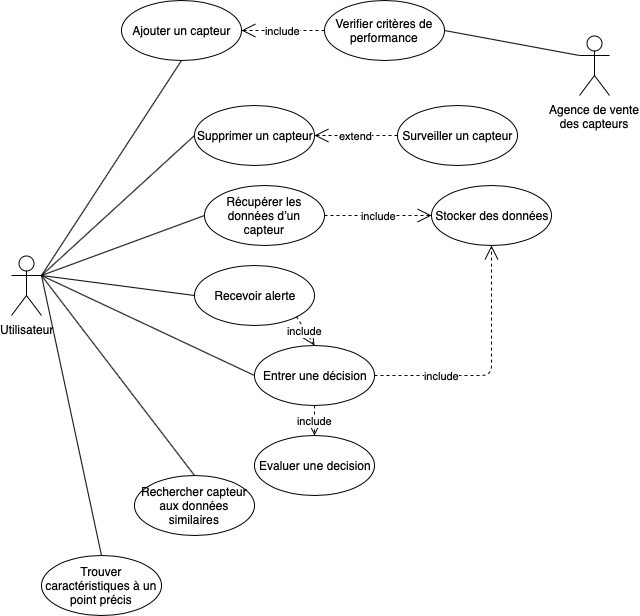
|  |  |
| --- | --- |
| Fonctionnalité F15 | Entrer une décision  void entrerDecision(Decision laDecision,double valeur) |
| Description | Cette fonctionnalité permet après une alerte passer à l’utilisateur de rentrer la décision qu’il a pris grâce a des « oui / non ». |
| Données nécessaires | Décision que l’utilisateur a pris. |
| Pré conditions | L’utilisateur doit avoir reçu une alerte. |
| Post conditions | RAS |
| Données en sortie | RAS |
| Tests | * Tester si on rentre 2 fois la même décision. |

|  |  |
| --- | --- |
| Fonctionnalité F16 | Evaluer décision  void evaluerDecision(string sensorId, double valeurActuel) |
| Description | Cette fonctionnalité permet de vérifier si une décision qui a été prise était bien ou non et ainsi d’attribuer une note à la décision. Cette fonction est appelée de manière automatique . |
| Données nécessaires | Une décision et des données enregistrées après la décision. |
| Pré conditions | Il faut que le capteur qui envoie les données soit toujours viable |
| Post conditions | RAS |
| Données en sortie | RAS |
| Tests | * Tester si on veut noter une décision inexistante * Tester si on a rentré une décision puis que les données enregistrées sont bonne qu’elle soit bien notée. * Tester si on a rentré une décision puis que les données enregistrées sont mauvaise qu’elle soit mal notée. |

|  |  |
| --- | --- |
| Fonctionnalité F17 | Proposer décision  Decision proposerDecision() |
| Description | Cette fonctionnalité permet de proposer une décision à prendre si l’utilisateur le souhaite lors de l’envoie d’alertes. |
| Données nécessaires | Une base de donnée avec toutes les décision prises dans le passé ainsi que les notes qu’elles ont reçues. |
| Pré conditions | Il faut que l’utilisateur ai reçu une alerte, qu’il y ait déjà au moins une décision prise et qu’elle soit mieux noté que 2. |
| Post conditions | RAS |
| Données en sortie | Envoie d’une proposition de décision à l’utilisateur |
| Tests | * Tester que l’on n’envoie pas des décision mal notée. * Tester si on a pas de bonne proposition à envoyer. |

* + Spécifications non fonctionnelles de l’application :
* Coûts : des capteurs, de l'entretien, de la maintenance …
* Fiabilité : utilisation des indices Atmo pour évaluer les données reçues
* Intégrabilité : facile d’ajouter des capteurs
* Sécurité : les données récupérées ne sont accessibles qu’aux utilisateurs de l'application
* Robustesse : en présence de données invalides par un capteur supposé défaillant
* Performance : actualisation des données fréquentes
* Évolutivité : l’application peut évoluer en ajoutant des fonctionnalités et des attributs aux données des capteurs

Diagramme de Use Case :



1. Manuel utilisateur

**Manuel Utilisateur application Capt’air**

Version 3, 06/06/2019

**Généralités**

              L’application Capt’air permet de contrôler le bon fonctionnement et d’utiliser les données de capteurs surveillant la qualité de l’air installés sur un territoire. **Il est important de savoir que nous supposons que les données sont entrées en temps réel** (nous permettons la lecture d’un fichier afin d’avoir des données réalisés avant le temps actuel) .Cette application est destinée à des employés gouvernementaux et s’exécute sur terminal. Une fois exécutée, un menu textuel s’affiche sur le terminal et propose 3 catégories d’actions : 1) Gestion des capteurs, 2) Gestion des mesures et 3) Paramètres d’alertes. L’utilisateur peut renseigner le type d’action qu’il souhaite effectuer en entrant 1, 2 ou 3 sur son clavier (4 étant réservé pour quitter la plateforme). S’affiche alors les différentes fonctionnalités proposées : elles sont présentées dans les parties suivantes.

A chaque fois l’utilisateur indique la fonctionnalité qu’il veut utiliser en rentrant le numéro correspondant et renseigne les différents éléments demandés également par une entrée clavier sur le terminal.

Après l’exécution de la fonction, le menu textuel est de nouveau affiché.

A tout moment l’utilisateur peut revenir de l’onglet 1, 2 ou 3 au menu de départ en entrant le numéro correspondant au « Retour au Menu ».

Un lexique est présent à la fin de ce guide pour spécifier des termes ambigus.

**Gestion des capteurs**

Dans cet onglet, 6 fonctions sont proposées :

* « Consulter liste des capteurs » affiche la liste de tous les capteurs ainsi que leurs coordonnées et description.
* « Ajouter », qui permet d’ajouter un capteur au circuit afin de le rendre fonctionnel et de pouvoir récupérer ses données. Après avoir sélectionné cette fonction, les caractéristiques (SensorID, latitude, longitude et description) du capteur sont demandées afin de pouvoir l’ajouter au circuit. Puis un message de confirmation ou d’erreur indique si l’ajout a été effectué avec succès.
* « Supprimer » permet de supprimer un capteur du circuit. Après avoir sélectionné cette fonction, il suffit de renseigner le SensorID du capteur à supprimer puis de confirmer sa suppression pour le retirer du circuit. Puis un message de confirmation ou d’erreur indique si l’ajout a été effectué avec succès.
* « Surveiller » permet de vérifier la fiabilité d’un capteur. Après avoir sélectionné cette fonction, il faut renseigner le SensorID du capteur dont on veut vérifier le fonctionnement pour que celui-ci soit testé. La viabilité des données du capteur, du capteur lui-même et sa dernière date d’actualisation (qui doit être inférieures à 2 jours avant la date d’exécution de cette demande) sont vérifiés. Un message indique si le capteur est toujours fonctionnel et sinon pourquoi (absence de données, ou pas d’actualisation récente etc.).
* « Rechercher capteur » : on renseigne des coordonnées c’est à dire une valeur de latitude et une valeur de longitude. Le SensorID du capteur le plus proche de ces coordonnées est affiché.
* « Capteurs similaires » : on renseigne un SensorID et un intervalle de confiance. Les capteurs (ID, coordonnées et description) ayant un comportement similaire au capteur renseigné à l’intervalle près sont affichés.

**Gestion des mesures**

Dans cet onglet, 8 fonctions sont proposées :

* « Consulter liste des attributs » affiche l’ensemble des attributs renseignés (attributeID, unité, description).
* « Consulter les données » affiche l’ensemble des données (date, SensorID, AttributeID, valeur) présentes dans le fichier des data.
* « Recueillir données similaires » : après avoir sélectionné cette fonction, il faut renseigner un SensorID et un AttributID et une date (ce qui permet d’avoir une mesure précise) ainsi qu’un intervalle de confiance. Les mesures similaires à la valeur de l’attribut du capteur à la date précisée pour l’intervalle de différence renseigné seront affichées.
* « Consulter des données » : après avoir sélectionné cette fonction, différents cas sont possibles, on peut choisir de renseigner ou non les paramètres suivants :
  + sensorID d’un capteur (seules les mesures du capteur seront prises en compte)
  + intervalles de coordonnées, latitude et longitude de départ et latitude et longitude de fin (seules les mesures du ou des capteurs le plus proche ou compris dans l’intervalle seront prises en compte)
  + intervalle de dates : précisé par un jour de début et un jour de fin (seules les mesures comprises entre le jour de début et de fin donnés seront prises en compte)
  + intervalle horaire : précisé par une heure de début et une heure de fin (seules les mesures comprises entre les 2 heures seront prises en compte)
  + attributID d’un Attribut (seules mesures de l’attribut seront prises en compte)

Une fois la fonction lancée, les mesures correspondant à notre recherche sont affichées.

* « Calculer moyenne de valeurs » : après avoir sélectionné cette fonction, il faut renseigner un sensorID, un attributID ainsi qu’une ou deux dates (dont la première est antérieure à la deuxième s’il y en 2). La moyenne des valeurs de l’attribut dans l’intervalle de dates et pour le capteur donné est calculée puis affichée.
* « Calculer les données prévisionnelles » : après avoir sélectionné cette fonction,  il faut renseigner un sensorID et un attributID. Des données prévisionnelles pour cet attribut et ce capteur sont calculées sur les 7 prochains jours et affichés.
* « Données au-delà d’un seuil » : après avoir sélectionné cette fonction, il faut renseigner un attributID, une valeur de seuil et sélectionner si l’on souhaite des valeurs au-dessus ou en-dessous ce seuil. Les données correspondantes, au-delà du seuil pour l’attribut donné, sont affichées.
* « Générer un graph » : après avoir sélectionné cette fonction, on effectue le même processus que pour la fonctionnalité « Consulter des données ». Cette fois, ce n’est pas la liste des mesures correspondantes qui est affichée mais un fichier .xlsx contenant un graphique représentant en nuage de points leurs valeurs est généré.

**Alertes, seuil et décisions**

Ce paquetage permet à l’application de générer en temps réel des alertes à l’utilisateur l’indiquant quand un capteur repère qu’une composante de l’air est trop élevé. Le programme peut aussi prédire pour un capteur précis et une composante donnée l’évolution de cette composante à un temps t+5 (ainsi le prévenant de l’éventualité où les valeurs prédites dépasseraient les seuils prédéfinis). Si le programme déduit qu’une valeur est au-dessus d’un seuil (en temps réel ou dans cinq temps), il demandera donc à l’utilisateur de prendre une décision afin d’améliorer la qualité de l’air (et pourra même lui en proposer une ancienne si elle a déjà été réalisée). Finalement, une note sera attribué à la décision prise afin de pouvoir mieux proposer à l’utilisateur la prochaine fois.

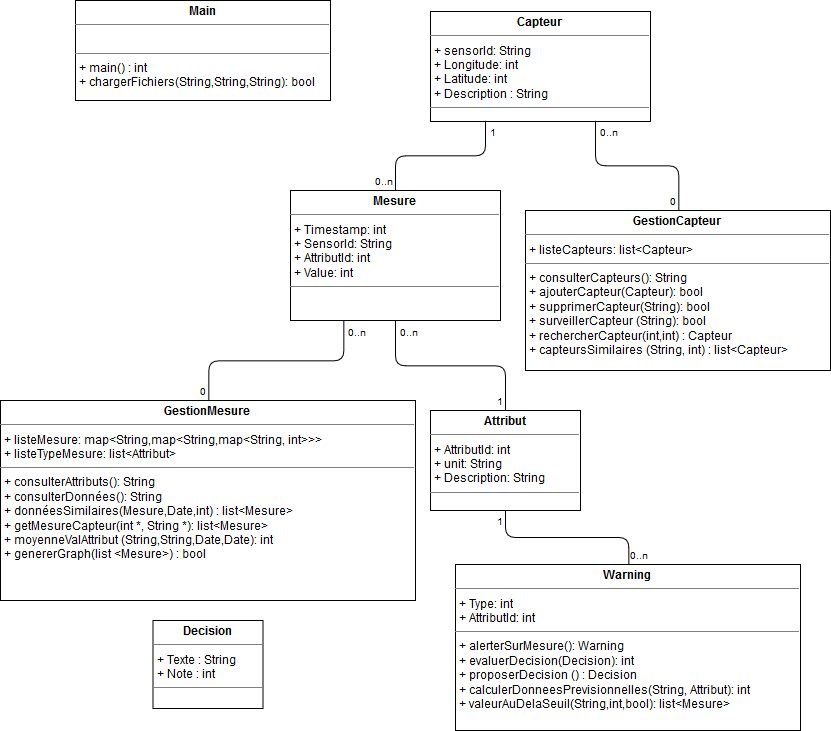
Les seuils définis pour le programme sont les indices ATMO de niveaux mauvais pour chacune des composantes.

Il est important de noter que toute ses fonctionnalités sont opaques a l’utilisateur, il ne peut pas faire appel à l’une d’elle ni les modifier.

**Lexique**

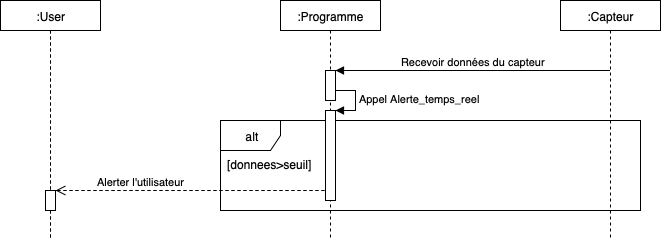
* + Latitude et longitude : entiers compris entre -90° et +90°

1. Diagramme de classes

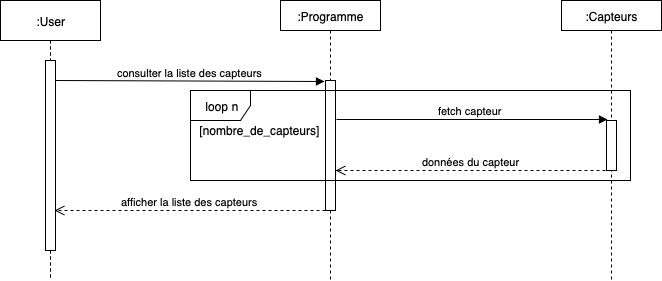


**Diagramme de classes de l’application Capt’Air**

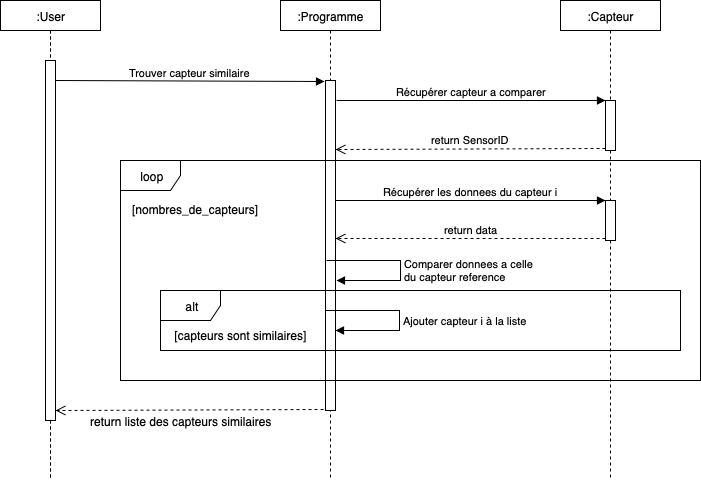
1. Diagrammes de séquence



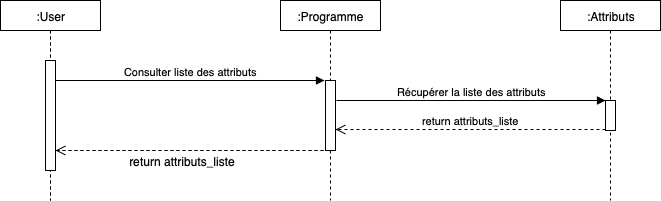
Alerte en temps réel



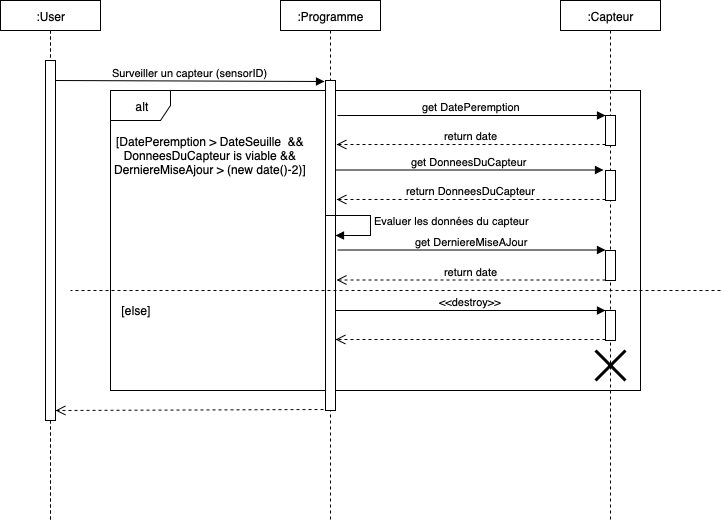
Consulter la liste des capteurs



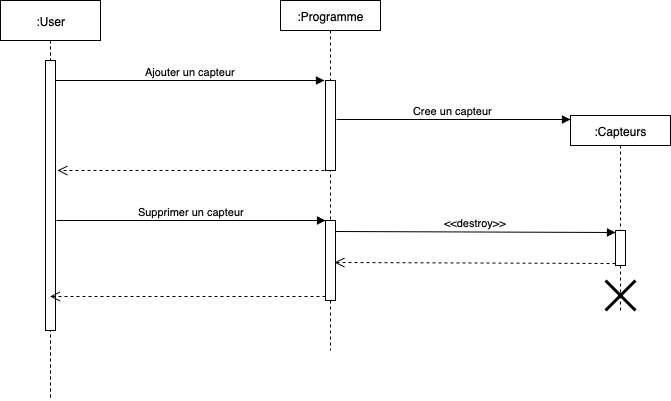
Capteurs similaires



Consulter attributs

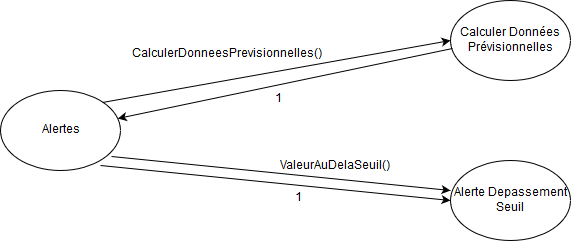


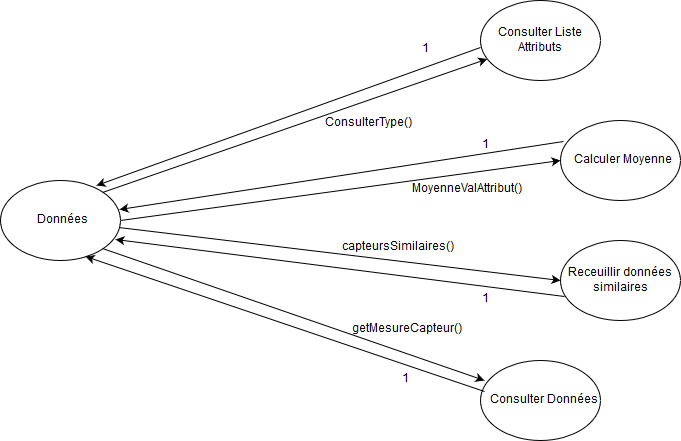
Surveiller un capteur

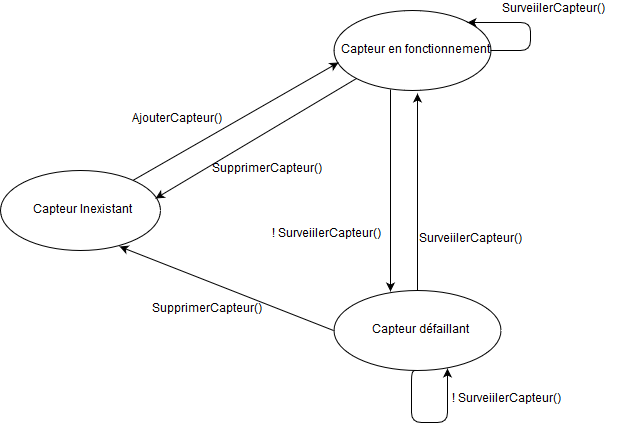


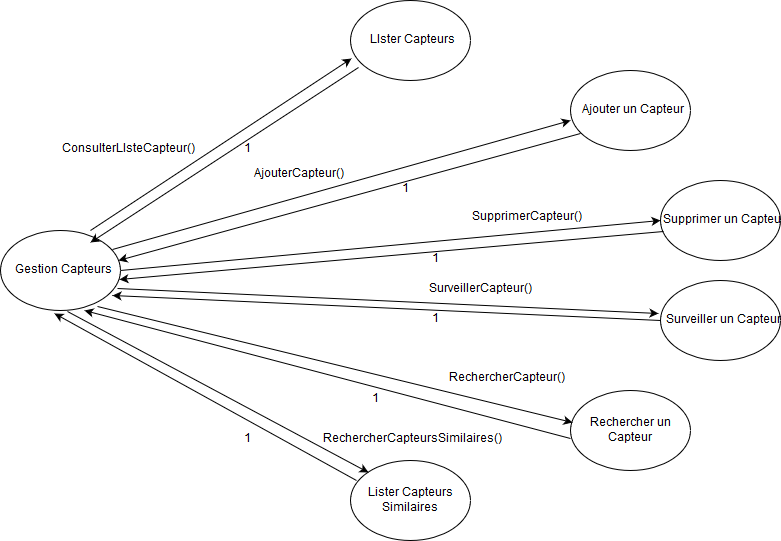
Ajouter et supprimer un capteur

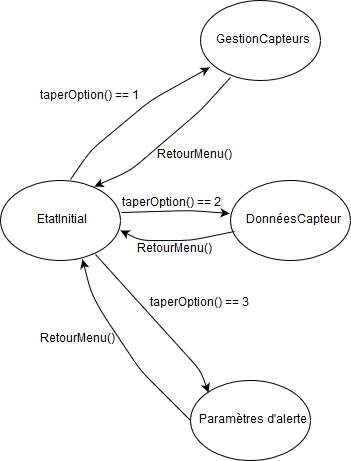
1. Diagrammes d’états transitions











1. Décomposition en modules

Pour une organisation du code optimal, nous divisions notre application en différents modules :

* + Packetage *Lecture*, qui contient les algorithmes de lecture de fichiers.
  + Packetage *Données*, qui contient les classes avec les structures de données mises en mémoire dans notre programme.
  + Packetage *Algorithmes*, avec toutes les fonctionnalités et algorithmes internes à l’application (autre que ceux de lecture).

1. Description des algorithmes importants

**moyenneValAttributs(String,String,Date,Date)**

Entrée : l’ID d’un des attributs des capteurs

Sortie : L’entier indiquant la moyenne des valeurs relevées en rapport avec cet attribut depuis le début des mesures

Si ( attributID n’est pas un des 4 conformes)

    Return -1

Sinon

    Parcourir la map globale pour tous les capteurs, puis pour l’attribut sélectionné, faire la moyenne de toutes les valeurs dont la clé est cet AttributeID

**calculerDonneePrevisionelle (map<struct tm, map<string,double>>listeMesurebyDate, string lAttribut)**

Entrée : la structure de donnée triée par date pour le capteur donnée et l’id de l’attribut

Sortie : un Booléen indiquant si la donnée du capteur pour l’attribut donnée dépassera ou non le seuil prédéfini.

Récupérer les 5 dernières valeurs avec un reverse\_iterator, puis ensuite récupérer les différences successives entres ces valeurs. Grace aux valeurs recuperer, nous supposons qu’elles suivent une loi de fonctionne affine :ax+b = y. Nous trouvons le coefficient directeur et l ‘ordonne à l’origine afin de prédire y au temps t+5 ;

*Proposition de l’algorithme :*

double coef = (values[0]-values[4])/5;

double ordo\_ori = values[0]-(coef\*5);

double valeur\_futur= coef\*10+ordo\_ori;

Ensuite nous faisons appel à la fonction valeurAuDelaSeuil(lAttribut, valeur\_futur);

**capteursSimilaires(String,int)**

Entrée : un String représentant l’ID du capteur à tester

Sortie : liste de Capteurs similaires à ce capteur

Comparer les 5 dernières valeurs avec tous les capteurs pour les attributs, faire la racine carré de la somme des écarts au carré, puis regarder si elle dépasse un seuil (à fixer) pour chaque attribut. Si non, rajouter le capteur testé dans la liste, sinon ne pas le rajouter.

list<Mesure> getMesureCapteur(int \* boolTab, String \* arg)

Le tableau d’entiers boolTab ne contient que des 0 ou des 1. Il contient 1 si l’utilisateur choisi de préciser le paramètre, 0 sinon.

boolTab[0] correspond au sensorId

boolTab[1] correspond à l’attributeId

boolTab[2] correspond à un intervalle de dates

boolTab[3] correspond à une point de coordonnées

boolTab[4] correspond à un intervalle de coordonnées

Le tableau de String arg contient les différents arguments nécessaires aux paramètres.

Il est de taille maximal 7 puisqu’il peut y avoir au plus 8 paramètres renseignées (1 attributeId, 2 dates, 2 coordonnées ou un sensorId ou 4 coordonnées)

1. Planning

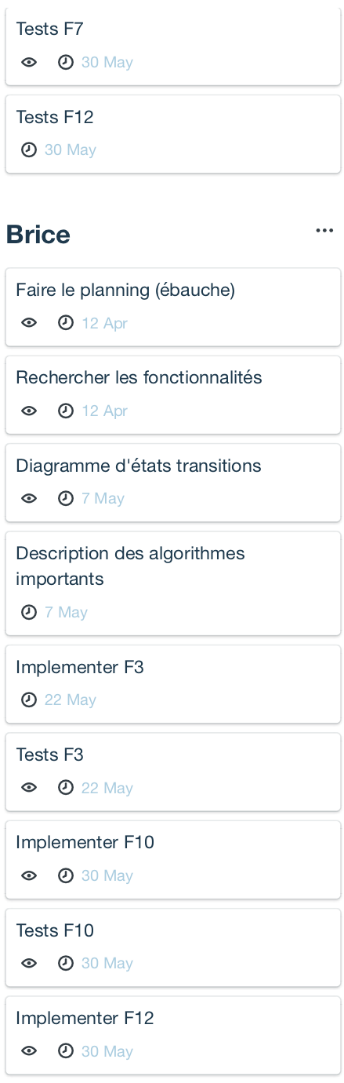
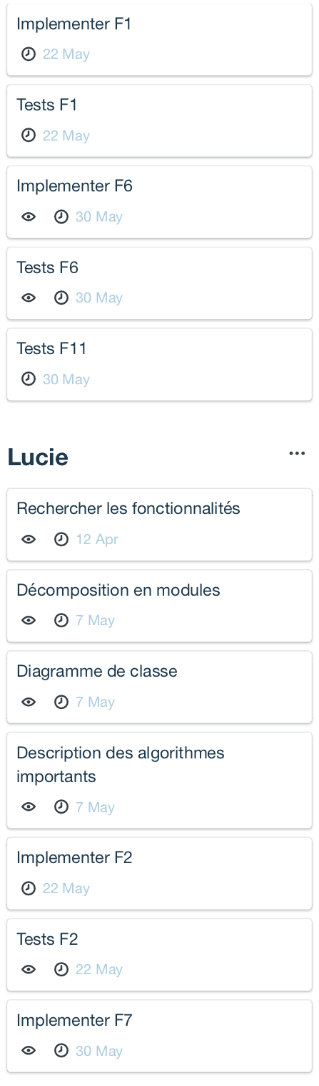
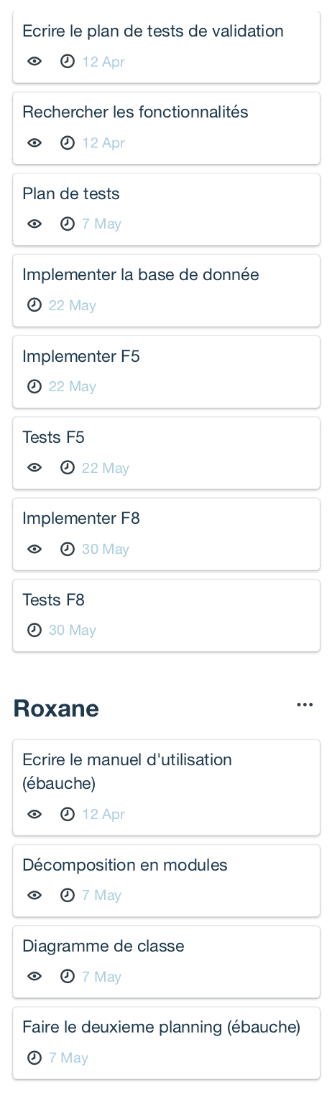
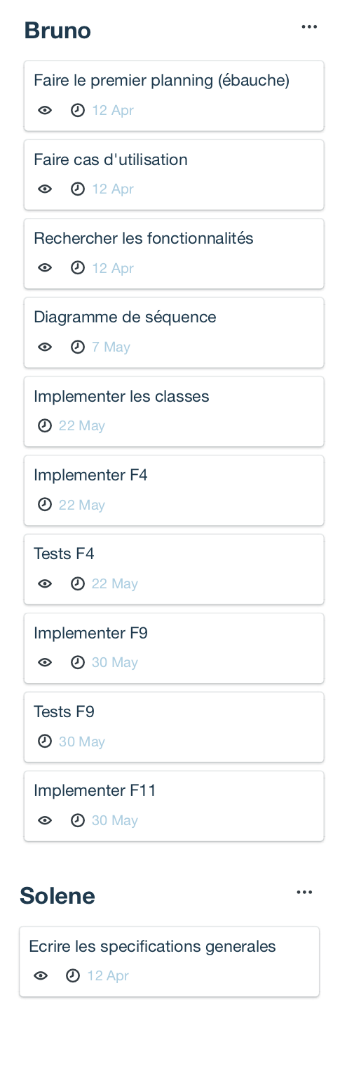
Afin de réaliser le planning provisoire, nous avons choisis de travailler avec un logiciel spécialisé dans la gestion de projet : *Trello*.

Nous avons créé une liste par membre du groupe, où nous avons ajouté des tâches (qui sont commentées et ont toute une date limite). De plus nous avons créé des listes de tâches à faire (actuellement vide) afin de pouvoir rajouter au fur et à mesure des tâches qui ne sont pas encore prévues (réparer un bug par exemple) et les attribuer au membre le plus “libre” ou le plus “motivé”.

Chef de projet : Roxane

Responsable test : Lucie

Responsable communication : Bruno



1. Tests de performances

Pour notre programme, nous voulions avoir toute les données en mémoire afin d’avoir un accès plus rapide lors de leurs utilisations ou consultations.

Afin de valider cet objectif nous avons voulu avoir un accès en mémoire extrêmement rapide qui justifierais le temps d’initialisation long (qui est dû au temps de chargement).

Le test a été mené de la façons suivante :

* + Nous avons fait augmenté le nombre de valeurs progressivement (par un facteur de 5 chaque fois)
  + Ensuite sur chaque période, nous avons pris un capteur aléatoire et avons chercher ses mesures. Une mesure sur le temps de la recherche est faite.
  + Nous avons effectué l’opération précédente N/2 fois (avec N le nombre de valeurs) et avons fait une moyenne sur le temps de recherche fait.

Notons que l’accès en mémoire est quasi instantané et a l’air constant pour un nombre de valeurs supérieur à 100 (il est possible que pour un nombre plus petit le nombre de valeurs prises était trop petit pour arriver à une conclusion cohérente).

Nous avons aussi voulu déterminer le temps de chargement (et donc d’initialisation du programme).

On peut déduire grâce à notre test que pour lire un demi-million de données notre programme met autour de 5 minutes, un temps qui peut être considérer long mais qui permet plus tard d’avoir un accès au valeurs instantané (de l’ordre de 2-3 µs).