Contenus

[1 Partie 1 3](#_Toc6382367)

[1.1 Enoncé 3](#_Toc6382368)

[1.2 Planning 3](#_Toc6382369)

[1.3 Journal de bord 3](#_Toc6382370)

[2 Partie 2 3](#_Toc6382371)

[2.1 Résumé Rapport TPI 3](#_Toc6382372)

[2.2 Documentation TPI (manuel technique) 3](#_Toc6382373)

[2.2.1 Specifications 4](#_Toc6382374)

[2.2.2 Conception 4](#_Toc6382375)

[2.2.2.1 Environnement de développement 4](#_Toc6382376)

[2.2.2.2 Modèle Conceptuel de donnée 5](#_Toc6382377)

[2.2.2.3 Modèle logique de donnée 5](#_Toc6382378)

[2.2.2.4 Conception détaillée 6](#_Toc6382379)

[2.2.3 Plan de test 7](#_Toc6382380)

[2.3 Manuel Utilisateur 9](#_Toc6382381)

[2.4 Code Source 9](#_Toc6382382)

# Partie 1

## Enoncé

Le sujet du TPI est de développer un serveur API qui permet de manipuler une base de données. Cette base de données est remplie par les contrôleurs SC4200C, leurs numéros de série, les sondes connectées à celui-ci ainsi que leur type de connexion.

Les commandes gérées par le serveur d’API sont :

En lecture

* Lire les instruments
  + Retourne la liste des instruments présents dans la base de données
* Lires les instruments, leurs sondes connectés, leur type de connectivité
  + Retourne la liste des instruments et pour chaque instrument les sondes connectées ainsi que leur type de connectivité
* Lire le nombre de sondes connectés par famille
  + Retourne le nombre de sonde connectées par famille

En Ecriture

* Ajouter un instrument avec ses sondes connectés et son type de connectivité
  + Ajouter à la base ce nouvelle instrument
* Effacer un instrument
  + Effacer de la base l’instrument (avec ses sondes et sa connectivité)

Sécurité

* La gestion de l’accès au serveur est assurée par un système de login/mot de passe selon le standard JSON WebToken. Le module nodejs « jsonwebtoken » est utilisé et la libraire d’encryptions est « bcryptjs ». Le jeton a une durée de validité de 1 heure.

## Planning

## Journal de bord

# Partie 2

## Résumé Rapport TPI

## Documentation TPI (manuel technique)

Le modèle de gestion de projet utilisé est le cycle en V.

### Specifications

La base de données aura pour but de stocker l’ensemble des contrôleurs SC4200C (numéros de séries), leurs sondes connectées, et leur type de connectivité.

Grace à l’API on pourra :

* Récupérer l’ensemble des contrôleurs
* Récupérer l’ensemble des contrôleurs, des sondes, ainsi que leurs types de connectivités
* Récupérer le nombre de sondes connectés par famille
* Ajouter un instrument avec ses sondes connectés et sa connectivité
* Effacer un instrument de la base de données
* Se connecter

### Conception

Cet API sera développé en Javascript ainsi que du SQL pour les interactions avec la base de données.

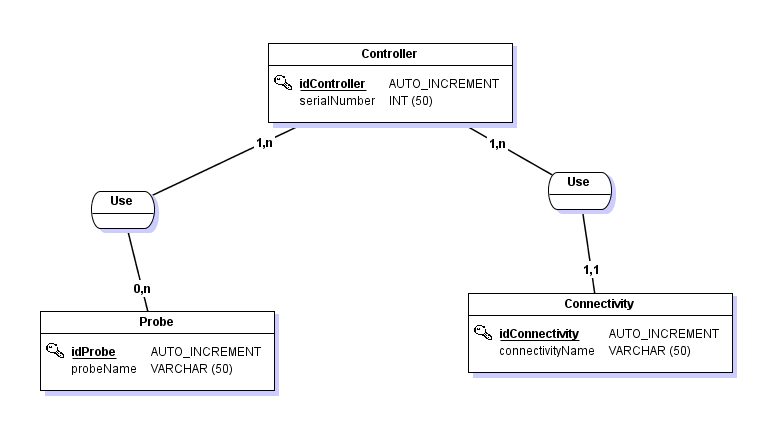
#### Environnement de développement

Le logiciel utilisé pour développer est Visual studio Code. L’archivage du code est fait avec git à l’adresse suivante « [http://stash.hach.ewqg.com/users/lsaccone/repos/tpi-luca/browse](http://stash.hach.ewqg.com/users/lsaccone/repos/tpi-luca/browse%20)  ».

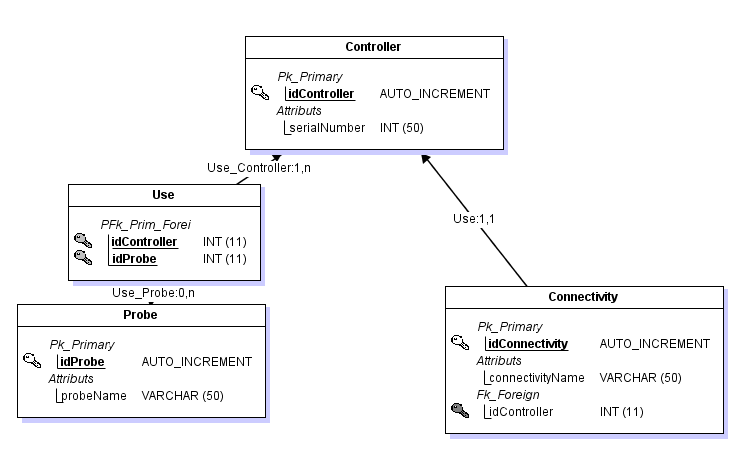
Les paquets NPM utilisés sont :

* Babel : outil aidant les développeurs à écrire du code dans la dernière version de Javascript
* Cross-env : utilise les variables d’environnements à travers la plateforme
* Eslint et ces plugins : outil d’analyse statique du code source
* Jsdoc : outil générant la documentation du code source à partir des commentaires
* Minami : Theme pour la documentation du code source
* Swagger-jsdoc : outil générant la documentation de l’API
* Ibm-openapi-validator : outil validant la documentation de l’API
* Nodemon : outil relançant automatiquement l’application quand le code source est modifié. Cela facilite le développement
* Koa.js : Middleware utilisant les fonctions asynchrones ECMAScript 2017
* JSON-web-token : outil permettant l’échange sécurisé de jeton entre plusieurs parties
* Bcrypt.js : librairie hashant et comparant des mots de passes dans node.js
* Mysql : node.js module pour écrire les requêtes sql

#### Modèle Conceptuel de donnée



#### Modèle logique de donnée



Controller (idController, serialNumber, #idConnectivity);

Connectivity (idConnectivity, connectivityName);

Probe (idProbe, probeName);

Use (#idController, #idProbe);

#### Conception détaillée

Une connexion sera faite pour pouvoir faire des requêtes sur les routes cette connexion sera valide pendant une heure. Sur les routes, l’utilisateur fait des requêtes HTTP pour récupérer les données. Ces requêtes créent une interaction avec le serveur SQL qui renverra les données.

### Plan de test

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Lecture de la liste d’instruments | * L’utilisateur fait une requête GET avec le logiciel Postman sur la liste des instruments disponible * Il reçoit un fichier JSON lui montrant la liste des instruments présent   Example:  {  "Controllers existing in the database": [  {  "serialNumber": "HL001\_17386\_000001729709"  },  {  "serialNumber": "HL001\_17386\_000001729708"  }  ]  } |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Lecture de la liste d’instruments, de deux sondes et d’une connexion | * L’utilisateur fait une requête GET avec le logiciel Postman sur la liste des instruments, des sondes et de leurs connectivites * Il reçoit un fichier JSON lui montrant la liste des instruments présent, leurs sondes ainsi que leurs connectivités   Example:  {  "Controllers":  {  "id": "1",  "serialNumber": "HL001\_17386\_000001729709",  "1 probe ": "Amtax sc",  "2 probe": "Phosphax sc",  "Connectivity": "LAN"  }  } |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Lecture du nombre de sondes du même type | * L’utilisateur fait une requête GET avec le logiciel Postman sur le nombre de sondes Amtax présentes dans la base de données * Il reçoit un fichier JSON lui montrant le nombre de sonde Amtax   Example:  {  "Numbers of Amtax sc":  {  "numbers": "12"  }  } |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Ajout d’un instrument | * L’utilisateur fait une requête POST avec le logiciel Postman, dans le Body il écrit un fichier JSON comme ci-dessous :   {  "serialNumber": "HL001\_17386\_000001729709",  "1 probe": "LDO sc",  "2 probe": "Phospax sc",  "Connectivity”  }   * Il reçoit un fichier JSON lui montrant son nouvel ajout à la base de données   Example:  {  "serialNumber": "HL001\_17386\_000001729709",  "1 probe": "LDO 2 sc",  "2 probe": "Phospax sc",  "Connectivity”  } |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Effacement d’un instrument | * L’utilisateur fait une requête DELETE avec le logiciel Postman avec dans l’url rechercher il met l’id qu’il veut supprimer à la fin, clique sur SEND * Il a supprimé l’instrument qu’il a voulu enlever |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Création d’un utilisateur (sign up) | * L’utilisateur fait une requête POST avec le logiciel Postman, dans le Body il écrit un fichier JSON comme ci-dessous :   {  "email": "luca.saccone@hach.com",  "password": "lulu"  }   * Clique sur Send pour envoyer la requête * Son profil est créé |  |

|  |  |  |
| --- | --- | --- |
| Sujet à tester | Test à faire | Test validé |
| Connexion d’un utilisateur (sign in) | * L’utilisateur fait une requête POST avec le logiciel Postman, dans le Body il écrit un fichier JSON comme ci-dessous :   {  "email": "luca.saccone@hach.com",  "password": "lulu"  }   * Clique sur Send pour envoyer la requête * L’utilisateur est connecté |  |

## Manuel Utilisateur

## Code Source