Tempora 4

Affichage de l’historique (stage)

Port ssh : 22....
Port front : 80....
Port api : 90....

Début du projet le 07/04/2025

Résumé exécutif

Projet

Tempora 4 - Refonte et Modernisation d'une Application d'Agenda Partagé

Tempora 4 est un projet ambitieux visant à refondre l'application d'agenda partagé de la société Tempora. Cette application est utilisée par plus de 1000 utilisateurs dans des domaines variés tels que médical, juridique et libéral. Créée en 1998, l'application en est à sa troisième version mise à disposition des clients en 2016. L'application est aujourd'hui confrontée à des défis techniques et fonctionnels dus à l'obsolescence de certaines technologies, rendant son évolution complexe et coûteuse en temps.

Objectifs du projet

1. **Amélioration technique :**
    - Moderniser l'architecture et le code de l'application (API et frontend).
    - Mettre à jour les technologies utilisées pour améliorer la sécurité, la performance et la maintenabilité.
2. **Refonte fonctionnelle :**
    - Simplifier et optimiser les fonctionnalités existantes (agenda, messagerie, consignes, répertoire).
    - Ajouter de nouvelles fonctionnalités répondant aux besoins actuels des utilisateurs.
3. **Expérience utilisateur :**
    - Moderniser l'interface utilisateur pour une meilleure ergonomie.
    - Intégrer des droits d'accès personnalisés pour une gestion fine des autorisations.
4. **Sécurité et conformité :**
    - Implémenter le protocole HTTPS pour sécuriser les échanges de données.
    - Assurer un chiffrement robuste des mots de passe et des informations sensibles.

Enjeux stratégique

- Répondre aux attentes croissantes des utilisateurs en termes de fluidité et de fonctionnalités.
- Maintenir une position concurrentielle sur le marché des solutions d'agenda partagé.
- Réduire les coûts de maintenance et faciliter l'intégration de mises à jour futures.

Approche et Méthodologie

Le développement sera réalisé selon une méthodologie **agile**, garantissant flexibilité et réactivité face aux évolutions. Une version bêta sera d'abord mise à disposition pour recueillir les retours des utilisateurs, avant le déploiement final.

Technologies clés

- **API Backend :** Scala 3.4.2, Play Framework 3.0.4.
- **Frontend :** Angular, Clarity Design.
- **Base de données :** PostgreSQL (base existante utilisée).

Livrables principaux

- Une application complète, stable et performante.
- Une interface utilisateur intuitive et ergonomique.
- Des modules fonctionnels prioritaires : agenda, messagerie, consignes, répertoire.

Plan de mise en oeuvre

- **Phase 1 :** Développement de la version simplifié (beta) et tests interne (octobre 2023 - mars 2025)
- **Phase 2 :** Mise en ligne de la version bêta (juin 2025)
- **Phase 3 :** Développement de la version complète et test interne (juin 2025 - date indéfinie)
- **Phase 4 :** Livraison finale et déploiement (courrant 2026)

Conclusion

Tempora 4 est une étape décisive dans l'évolution de l'application. En modernisant son socle technique et en enrichissant ses fonctionnalités, la société Tempora vise à offrir une solution pérenne et performante, adaptée aux besoins actuels et futurs de ses clients.

Introduction

Contexte

La société Tempora est un éditeur d'application qui propose à plusieurs milliers de professionnels du domaine médical, paramédical, libérale et autre, une solution applicative d'agenda partagé. Cette application existe depuis 1998 et la version actuellement en production est utilisée depuis 2016. Aujourd'hui, Tempora dénombre plus de 1000 dossiers actifs.

L'équipe de Tempora reçoit régulièrement des demandes de mise à jour de la part de ses clients, cependant le code et les technologies utilisés pour la version actuelle deviennent obsolètes. Cela peut rendre certaines mises à jour difficiles à développer. De plus, certains morceaux de code et fonctionnalités sont redondants ou inutiles, ce qui ne simplifie pas la gestion des projets et la maintenance du code.

C'est dans l'objectif de rendre les mises à jour et la gestion du projet plus simples que le développement de Tempora 4 a débuté. L'application permettant à ses utilisateurs d'effectuer diverses actions telles que la prise de rendez-vous, la création de messages et bien d'autres encore. L'erreur étant humaine, les actions des utilisateurs doivent pouvoir être tracées facilement. C'est pour cela que Tempora propose à ses utilisateurs une historique complète des actions effectuées sur les différents agendas du système. C'est cette partie qui sera réalisée par les stagiaires.

Objectifs

Le projet aura plusieurs objectifs dont certains seront secondaires. Les objectifs principaux sont :

- Afficher l'historique pour une période souhaitée
- Filtrer l'historique par :
  - Période
  - Sujet
  - Action
  - Mots clés

Les objectifs secondaires sont :

- Ajouter le filtre par :
  - Opérateur
  - Agenda
- Afficher toutes l'historique de l'organisation

- Ajouter des catégories, proposée par les stagiaires

Tempora, qu'est ce que c'est ?

L'application

L'application Tempora est une application d'agenda partagé entre le secrétariat et les professionnels propriétaires d'un agenda. Cette application en est actuellement à sa troisième version.

Tempora 1, a été développée en 1998 par Mr Joël Connault. Ce développement à eu lieu pour répondre au besoin de la société Allo Supletel, une permanence téléphonique à Brest, aujourd'hui toujours cliente de la société Tempora. Cette version de l'application était à la fois disponible en tant qu'application native et web. Pour la version native le langage utilisé était le Delphi. Deux SGBD[1] étaient utilisés : Paradox pour la version locale et MySQL pour la version web. Cependant, Tempora 1 était très contraignante car elle demandait l'installation d'un serveur chez les clients ainsi qu'une synchronisation des données faite manuellement par l'utilisateur. La mise à jour de l'application native pouvait poser certains problèmes car elle devait être réalisée individuellement sur chaque poste utilisateur.

Tempora 2, a été réalisé en 2010. L'avantage de cette version est qu'elle demandait l'utilisation d'un serveur unique, réglant ainsi le souci de mise à jour. Elle demandait l'utilisation d'un applet Java sur chaque poste où l'application serait utilisée, ce qui nécessitait l'utilisation d'un navigateur web. Contrairement à la version précédente, cette version n'utilisait plus qu'un seul SGBD. Elle a aussi apporté une nouveauté par rapport à Tempora 1, la synchronisation des données en temps réel.

C'est suite à un problème de support des applets Java par les navigateurs que Tempora 3 a été réalisé. Cette version a été mise à disposition des clients en 2016. Elle a nécessité la refonte complète de l'application, avec changement de structure, de langage et de SGBD. Le développement initial a été réalisé par une société de sous-traitance. L'application étant une application web avec utilisation d'une API REST, les langages utilisés sont les suivant :

- HTML 5, CSS et JavaScript pour l'interface utilisateur
- Scala pour la partie api.

[1] Serveur de Gestion de Base de Données

Le SGBD choisi est Postgresql. Tempora 3, est la version actuellement proposée aux clients.

En 2019, une version dérivée de Tempora 3 a été mise à disposition des clients. C'est une version simplifiée et mobile. Elle à été réalisée en utilisant la technologie hybride en passant par ionic. C'est-à-dire qu'elle a été développée comme une application web en utilisant HTML 5 et compiler comme une application native disponible sur les téléphone et tablette Android et IOS.

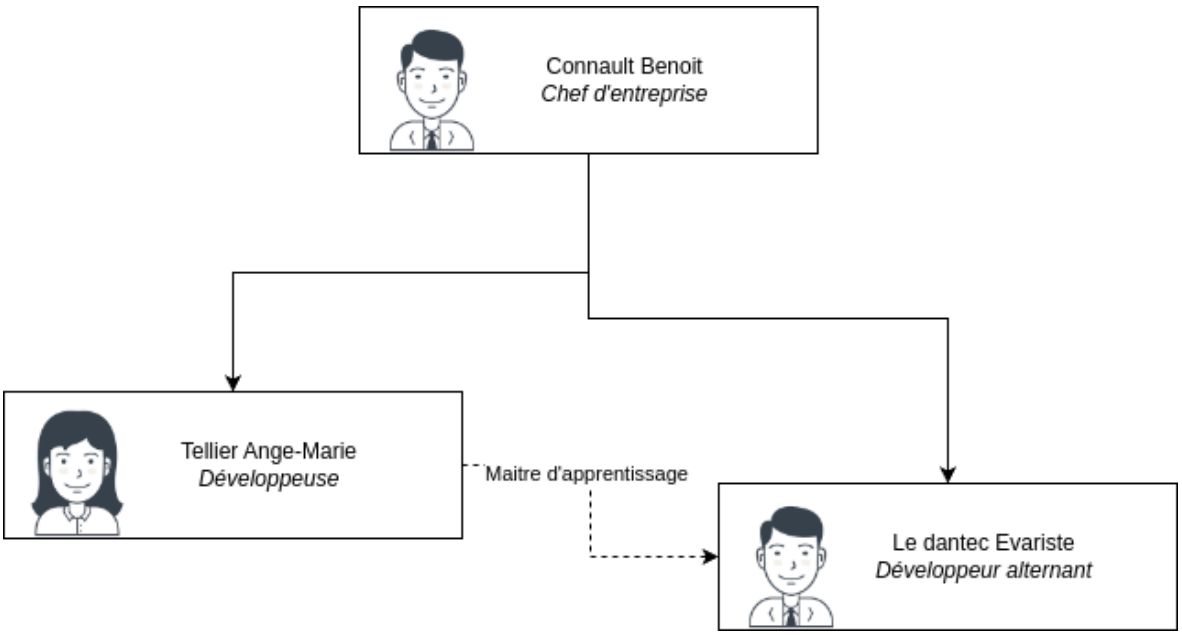
Depuis mai 2023, la société Tempora a débuté le développement de Tempora 4.

L'entreprise et son équipe

La société Tempora à été fondée en 2009 afin de pouvoir proposer Tempora à des clients externes à Allo Supletel. L'équipe de Tempora est actuellement composée de 2 personne à temps plein :

- Connault Benoit en sa qualité de chef d'entreprise et de développeur. Pour son parcours scolaire, il a réalisé un Bac STI, et une première année en BTS informatique, qui a malheureusement été écourté suite à un problème financier de l'école. Après cela, il a suivi quelques périodes de formation non diplômante. Il a commencé un contrat professionnel avec Allo Supletel quelques années avant la création de la société Tempora. Il a commencé par faire de la maintenance et du dépannage sur les deux versions de l'application. Il a aussi réalisé quelques opérations d'amélioration de Tempora 2. Lorsque le sous-traitant qui devait réaliser le développement initial a pris du retard, Benoit a participé à ce dernier. Depuis, il réalise des améliorations continues. Entre septembre et octobre 2020, il a racheté l'entreprise de Tempora après le départ du précédent chef d'entreprise.
- Tellier Ange-Marie en sa qualité de développeuse. Son parcours scolaire est assez atypique, elle a réalisé un Bac Pro Service à la personnes dans un lycée agricole. Puis elle a poursuivie ses études dans l'informatique en réalisant un BTS SIO et une formations de deux ans en Bac +4, avec le CESI de Brest. C'est pendant cette formation qu'elle a signé un contrat de professionnalisation en alternance avec Tempora, en 2016. Et a continué son travail au sein de l'entreprise avec un CDI depuis 2018.

Depuis septembre 2023, une nouvelle personne, Le Dantec Evariste, a rejoint l'équipe de Tempora en tant que développeur en alternance.



Organigramme de Tempora depuis septembre 2023

Une synthèse des dates concernant l'application et l'entreprise est disponible en Annexe 1.

Étude de l'existant - Tempora 3

La version 3 de Tempora, comme la version 4, se divise en deux projets: le frontend et l'api. Comme mentionné précédemment, elle a été mise en production et à disposition des clients en 2016.

L'application a été conçue pour répondre aux besoins de divers professionnels, que ce soit des professionnels de santé, des avocats, des notaires ou autres. Elle est, de manière générale, à destination des permanences téléphoniques (secrétariat externe). Elle est composée de 4 modules principaux : la prise de rendez-vous (l'agenda), la messagerie, les consignes et le répertoire. Les rendez-vous pris sur l'agenda sont composés de motifs, ce qui, entre autres choses, leur donne une durée et une couleur. Mais avant d'entrer dans le vif du sujet, voici une description et une étude de l'existant.

L'api

L'api de la version 3 tout comme celle en cours de développement pour la version 4, sont des api REST, c'est-à-dire, de manière simplifiée, qu'elle peut être utilisée par d'autres applications, s'ils ont les routes d'accès et des droits pour l'utilisation. Actuellement toutes les applications de Tempora (le frontend, la prise de rendez-vous en ligne, les applications mobiles) en production utilisent cette API.

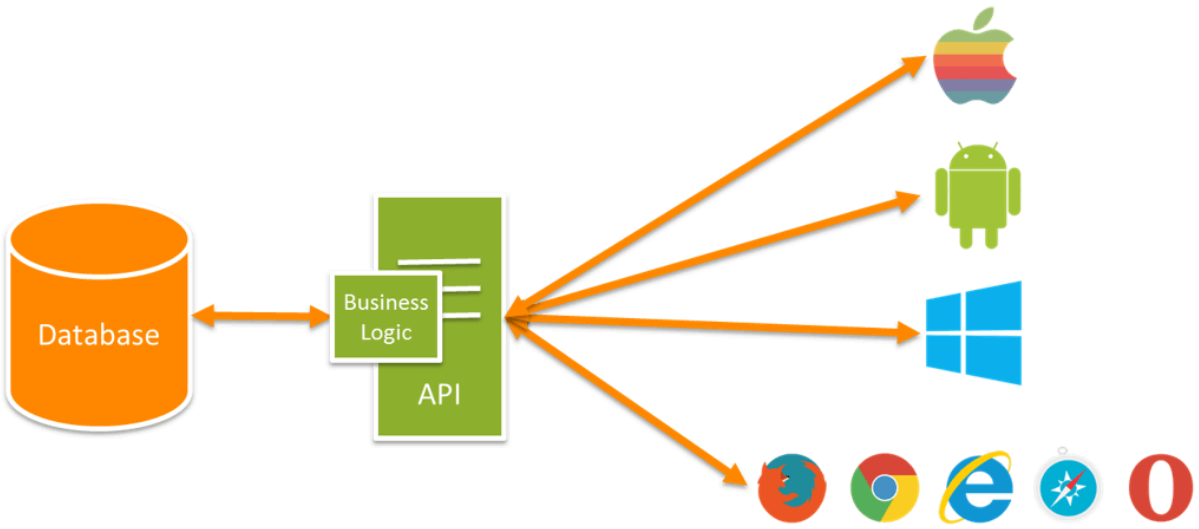


Schéma d'une API REST

Or depuis quelques temps, le projet tempora-backend (l'API actuel) rencontre quelques soucis pour l'évolution de l'application. C'est pour cela que le projet de l'API v4 a été mis en place. L'un des soucis principal rencontré par la V3, est que certaines technologies sont obsolètes et n'ont jamais été mises à jour.

Les technologies utilisées

L'API, comme mentionné précédemment, a été développée en scala. Le scala est un langage à la fois objet et fonctionnel, dérivé du Java. Pour la version 3 de Tempora, la version de scala utilisée est la 2.11.1, qui est une version obsolète puisqu'actuellement scala en est à sa 3ème version. Pour ce projet, les développeurs utilisent le framework play qui lui aussi est dans une version obsolète, la 2.3.10. Ce retard de version pose plusieurs inconvénient : les développeurs ne peuvent pas utiliser d'autres versions de java que la 8, alors que depuis mars 2024 on en est à la version 20. Autre souci, l'évolution et l'intégration de certaines librairies nécessaires à l'évolution de Tempora n'est pas possible.

L'obsolescence du code

L'API ayant été développée en couche, certains morceaux de code et certaines fonctionnalités sont obsolètes ou carrément inutiles, voir mise en doublons dans certains fichiers. Cette obsolescence du code de l'api peut elle aussi avoir un inconvénient. Par exemple, l'inclusion de l'utilisation des droits utilisateurs personnalisés, présents dans les objectifs secondaires.

L'architecture du projet

L'architecture globale de tempora-backend est faites en MVC[2] (voir dans la partie concernant le projet) avec quelques petites subtilités qui seront expliqués ensuite.

Le projet possède plusieurs dossiers tels que app, conf, lib, etc… Le dossier conf, comme son nom l'indique, contient la configuration du projet et notamment les fichiers de routes qui permettent de rediriger les requêtes faites par le frontend. Le dossier app contient la totalité des fichiers nécessaires au fonctionnement du projet, c'est-à-dire les contrôleurs, les modèles (dbdata), les fichiers d'api, et les vues.

- **Les fichiers d'api**
  Ils servent d'interface entre Tempora et d'autres applications externes comme google calendar, par exemple.

- **Modèles**
  Ils font l'interface entre la base de données et les contrôleurs. Ils exécutent les requêtes SQL en fonction des demandes faites par les contrôleurs. Puis ils traduisent le résultat obtenu en objet à fournir à ce dernier.
  Ce sont eux qui contiennent la structure d'une classe objet et les fonctions de conversion pour ces dernières.

- **Contrôleurs**
  Ce sont les fichiers au centre des requêtes envoyées par le frontend. Il les reçoivent par le biais des fichiers de routes et font appel au dbdata pour les exécuter dans la base de données.
  Ils traitent ensuite le résultat obtenu et renvoient une réponse au frontend.

- **Vues**
  Ce sont des fichiers HTML, permettant d'envoyer des informations par mails à la suite d' une requête de l'api. Ou bien d'avoir un aperçu avant l'impression.

Pour en savoir plus sur la communication entre les différents fichiers du projet voir l'Annexe 2.

Le frontend

Les technologies utilisées

Le projet tempora-frontend, est un projet web classique qui utilise :

- HTML pour l'IHM[3].

[2] Model View Controller
[3] Interface Homme Machine

- CSS, en passant par du less pour les feuilles de style. L'un des avantages de less, utilisés dans le projet, est qu'il permet l'imbrication de blocs. Par exemple, il est possible de mettre un style général à une div et d'ajouter un bloc pour dire que si l'une d'entre elle à la classe "A", un ou plusieurs éléments de son style doivent changer.
- Javascript en utilisant jQuery pour le côté fonctionnel de l'application. Le projet utilise aussi de nombreuses librairies et utilitaires toutes basées sur jQuery qui permettent de faire des raccourcis et ainsi éviter de faire de la répétition de code.

La communication avec le backend

Tempora étant une application dynamique avec une API REST, elle doit envoyer des requêtes à cette dernière afin d'envoyer ou de récupérer des données. Et ce parfois sans avoir à recharger la page. Pour cela, les développeurs utilisent Ajax.

Ajax n'est ni une technologie ni un langage de programmation, il s'agit d'un concept qui reposent sur plusieurs technologies tel que json, XML et javascript. Comme mentionné précédemment, il consiste à envoyer des requêtes au serveur sans avoir besoin d'exécuter un rechargement de la page. C'est d'ailleurs pour cela que la plupart des développeurs vont préférer utiliser javascript pour réaliser des requêtes ajax.

Pour simplifier le codage des requêtes ajax, Tempora 3 utilise jQuery et a mis en place une fonctionnalités qui demandent des paramètres tels que la route et le type de requête entre autres choses.

Pour en savoir plus sur Ajax https://www.brioude-internet.fr/8454/qu-est-ce-que-l-ajax/

L'obsolescence du code

Le code du projet tempora-frontend est non seulement obsolète dans certains cas, mais comme pour l'API, elle a été développée en couche et son code est parfois spaghetti (mélangé). Bien que des mises à jour restent faisables, elles peuvent s'avérer parfois difficiles à mettre en place. Certains morceaux de codes ne sont même plus utilisés. Des modules et librairies utilisés dans ce projet ne sont plus mis à jour et sont donc utilisés avec des versions obsolètes; bien qu'ils restent fonctionnels. Par exemple, moment js qui est une librairie pour la manipulation des dates et heures.

L'architecture du projet

Comme pour l'API, le frontend utilise globalement une architecture MVC, avec quelques particularités et fichiers supplémentaires. Le développement sur ce projet concerne

principalement trois types de fichiers qui sont : les contrôleurs, les managers et les vues. Voici leurs descriptions :

- **Contrôleurs**

  Ils sont le cœur du logiciel et permettent de gérer les aspect de Tempora.

  C'est par le biais des contrôleurs que les demandes de données sont faites. C'est aussi eux qui les traitent et les fournissent au vue pour être affichée. Ce sont les contrôleur qui déclenche l'affichage d'un fichier HTML lorsque cela est nécessaire.

  Ils reçoivent et traitent les demandes utilisateurs au travers d'événements fournis par un autre fichier de l'application.

  Un contrôleur "connaît" et peut appeler n'importe quel manager et vue js. Généralement un contrôleur est dédié à un seul fichier HTML. Il peut arriver que dans certains cas, comme les formulaires de création et d'édition d'un élément, un fichier HTML soit relié à deux contrôleur, un pour chacune des deux action.

- **Managers**

  Ils font la passerelle entre les contrôleurs et l'API. manager est instancié et appelé par ce premier. Le contrôleur souscrit à un événement déclenché par le manager pour attendre la réponse à sa requête.

  Ce dernier transmet la demande au serveur par le biais d'une requête Ajax. Quand il reçoit la réponse, il déclenche l'événement de succès ou d'erreur auxquels est inscrit le contrôleur afin qu'il puisse traiter la réponse.

  L'adresse du serveur sur lequel les requêtes API sont exécutées est enregistrée dans un fichier afin de ne pas avoir à parcourir toutes les fonctions pour la changer si besoin est. Le fichier en question se nomme api.js. Pour les tests l'adresse du serveur est bien entendu localhost, pour cela api.js fournie deux variables : "serveur" pour la production et "local" pour les test et le dev.

  Un manager ne peut faire référence qu'à lui-même, il ne connaît pas les contrôleurs, vues etc.. qui font appel à lui.

- **Vues**

  Une vue dans Tempora 3, correspond à un fichier js qui gère l'affichage dynamique d'une page. Elle reçoit les données de la part du contrôleur et les affiche en modifiant le contenu du html.

  Elle met également en place des événement pour réagir aux commandes de l'utilisateur, afin qu'elles soient traitées et, si besoin, transmises au contrôleur en générant un autre événement, qui sera déclenché dans un fichier secondaire et pourra transmettre la demande.

  Généralement une vue est spécifique à un seul contrôleur et donc un seul html, mais il peut arriver que dans certains cas, une vue correspondent à plusieurs

contrôleurs, comme mentionné dans la description des contrôleurs. Par défaut, une vue ne connaît qu'elle-même et les fichiers html qui auront son sélecteur.

Pour comprendre vraiment la correspondance entre les différents fichiers du frontend et la communication avec l'api, voir l'Annexe 3.

Le projet - Tempora 4

Il a précédemment été établi que la version 3 de Tempora utilisait des versions obsolètes des différentes technologies qui la composent, tant pour l'api que pour le frontend. Après avoir étudié la possibilité de mettre à jour les versions utilisées, l'équipe de développement s'est aperçue qu'il est bien plus simple de créer une toute nouvelle version de l'application.

Pour l'API, le changement de version serait trop compliqué et demanderait un trop grand nombre de modification du code et de changement de certaines librairies et plugins utilisés. Cela pourrait vite devenir ingérable. C'est pour ça que le projet tempora-api a été initialisé.

Pour le frontend, l'obsolescence du code et la complexité de son arborescence demanderait aussi beaucoup de modifications. De plus, l'équipe de Tempora avait déjà étudié la possibilité de refaire l'interface avec une nouvelle technologie.

C'est dans le but de pouvoir refaire une application au propre et de pouvoir plus facilement ajouter des mises à jour que Tempora à choisie de créer sa version 4 de l'IHM.

Le module historique

Tempora 4 se décline en plusieurs modules qui sont tous plus ou moins liés entre eux, cependant le présent cahier des charges ne concerne qu'un seul d'entre eux : l'historique.
Les utilisateurs peuvent effectuer diverses actions d'écriture sur les dossiers (agenda), ils peuvent donc aussi commettre des erreurs. Afin de pouvoir tracer les différentes actions émises sur un dossier, l'historique les répertorie toutes. Une action enregistrée dans l'historique contient plusieurs informations qui permettent de l'identifier : un type/catégorie (ex : rendez-vous), une opération (ex : ajout), la personne ayant effectué l'opération (l'opérateur) et une date-heure d'émission. Bien entendu en base de données les informations sont plus nombreuses (cf. Annexe 7 et Annexe 8).
Les développeurs devront donc travailler sur une interface et des fonctionnalités afin de permettre aux utilisateurs de consulter l'historique d'un agenda, voir même de l'organisation.

Les développeurs pourront aussi proposer de nouvelles catégories (types d'action) à stocker dans l'historique.

Exigences du projet

Exigences techniques et fonctionnelles

L'api

La nouvelle version de l'api sera elle aussi réalisée en scala et utilisera des versions plus récentes des technologies suivantes :

- sbt
- Scala 3.4.2, qui est la dernière version déployée en mai 2024.
- Play framework 3.0.4, qui est la dernière version déployée en juin 2024.
- Java 17, qui est la version minimum recommandé pour les technologies précédemment citées.

***Attention** : ces versions sont celles qui étaient mises dans les fichiers de configuration build.sbt et build.sc en septembre 2024, au moment de la rédaction du présent cahier des charges. Il se peut qu'elles aient été changées, dans ce cas les nouveau numéro de version se trouvent dans la documentation technique de l'API v4 et dans les notes de mises à jour interne.*

Le frontend

La nouvelle version du frontend, qui s'appelle tempora-angular, sera réalisée à l'aide des technologies suivantes :

- node et npm pour la gestion des packages
- Angular, qui à l'avantage de permettres aux développeurs de travailler en modules et composants
- Clarity design pour le framework css.

La base de données

Base de données relationnelle, qui utilise le sgbd postgres. La base de données est toujours la même que pour la version 3, donc pas d'export de données.

Autre

Pour tous les projets un outil de versioning est utilisé : gitlab.

Exigences non fonctionnelles

Pour son utilisation et afin de protéger les données inscrites chez chacun des clients de Tempora, l'application utilise le protocole HTTPS que ce soit pour l'accès au frontend ou pour l'envoi des requêtes à l'api.

Les mots de passe des utilisateurs sont cryptés en base de données, et les utilisateurs doivent se connecter pour accéder à leurs données. Un utilisateur ne peut avoir accès qu'à un seul client Tempora.

L'équipe projet

Comme pour tout projet de développement, celui-ci à besoin d'une équipe de développeurs. Chaque membre de l'équipe aura un rôle prédéfinie, cette répartition n'est pas exhaustive et pourront évoluer. Les rôles sont les suivants :

- Chef de projet
- Développeur/Développeuse principale
- Développeur/Développeuse (stagiaire ou non)
- Alpha testeur

Même si dans ce projet, certains membres de l'équipe se verront attribuer le rôle d'alpha testeur, cela n'enlève en rien le fait que le développeur ou le développeur principal teste son code avant de le soumettre.

Les membre de l'équipe de développement et leur rôles sont les suivant :

- Tellier Ange-Marie : Chef de projet, développeuse principale et alpha testeur.
- Connault Benoit : Développeur principal et alpha testeur.
- Le dantec Evariste : Développeur.
- Bialgues Marion : Développeuse.
- Médoc Adèle : Développeuse.

Il est possible que les stagiaires travaillent sur un ou plusieurs morceaux de l'API, le rôle qui leur sera attribué est celui de développeur.

***Attention** : Les développeurs stagiaires et en apprentissage devront dans les premiers temps proposer leur code aux développeurs principaux pour de potentielles corrections d'algorithmes et de syntaxes du code.*

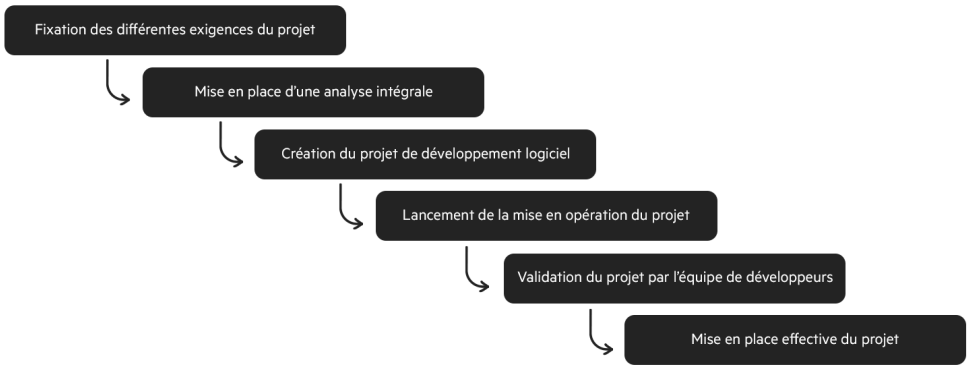
La méthode de développement de l'équipe

Il existe plusieurs méthodes de développement, les plus connues sont : la méthode en cascade, la méthode lean, la méthode itérative et la méthode agile (Cf.

https://www.eurotechconseil.com/blog/quelles-sont-les-methodologies-de-developpement-logiciels/)

La méthode en cascade

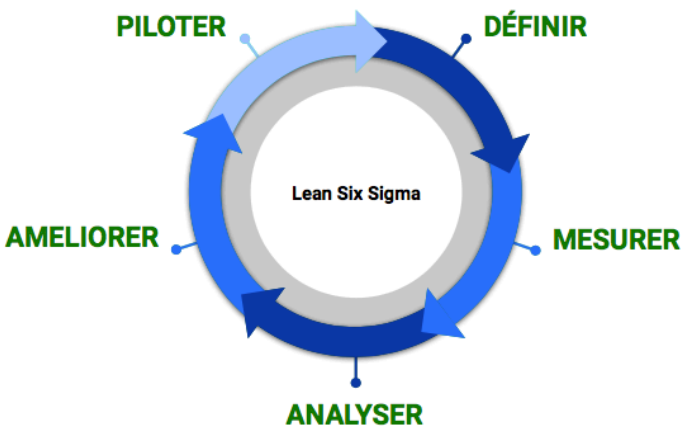
C'est une méthode fragmentée en 6 étapes. Elle a pour avantage d'avoir un passage rapide entre les différentes étapes et d'avoir un processus pertinent. Cependant, cette méthode n'est pas adaptée à la manière de travailler qu'utilise la société Tempora, car elle ne permet pas de revenir sur une étape.



La méthode lean

Cette méthode s'inspire "des procédés et activités de conception éthique". Elle repose ces fondamentaux :

1. la suppression du gaspillage
2. le renforcement de l'apprentissage
3. le report de la prise de décision
4. la livraison rapide
5. la responsabilisation des développeurs
6. l'intégration du processus



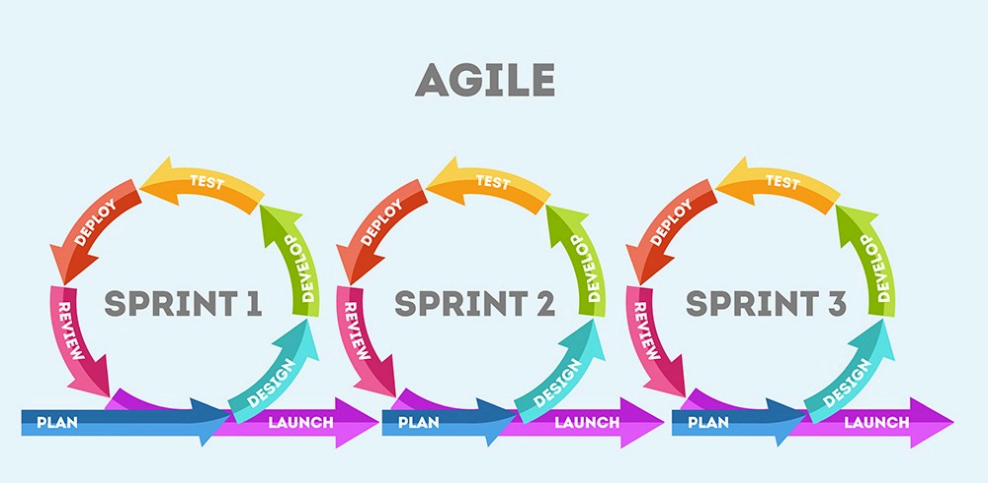
Cette méthode de développement ne donne pas lieu au multi-tâche, car elle consiste à se concentrer sur une tâche à réaliser à un moment T. Avec cette méthode, il est question "d'attendre la performance en matière de qualité de produit, de délais de livraison, de productivité ainsi que de réduction des coûts. Des

objectifs qui peuvent être atteints grâce à un process d'amélioration soutenue ainsi que la suppression du gaspillage".

Le simple fait de ne pas pouvoir faire du multi-tâche retire l'intérêt de cette méthode pour l'équipe de Tempora, qui pourrait avoir besoin de travailler sur différentes parties du projet de manière simultanée.

La méthode agile

La méthode de développement agile permet une gestion de projet itérative et collaborative. Elle met l'accent sur la flexibilité, la réactivité et l'adaptation au changement au sein même de l'équipe et du projet. Elle favorise une collaboration étroite entre tous les membres de l'équipe et encourage la communication ouverte et constante. Ce qui permet de s'assurer que les attentes sont clairement définies et comprises.



Cette méthode est adaptée à l'équipe de Tempora, car elle permet de se concentrer sur une fonctionnalité à livrer tout en permettant l'adaptation et les changements fréquents. Elle permet aussi de revenir en arrière sur une itération tout au long du processus de développement, ce qui permet une adaptation du code et des fonctionnalités en cours de développement.

Les informations sur cette méthodologie, ont été trouvé sur le lien suivant : https://www.eurotechconseil.com/glossaire/agile/

La méthode itérative

Cette méthode ressemble beaucoup à l'agile, qui utilise aussi des itérations. La différence réside dans le fait que la méthode itérative

satisfait deux besoins essentiels : le respect des délais de livraison et la détermination des coûts du projet à l'avance. Elle se concentre également sur la possibilité d'avoir accès à une frange d'innovation.

Cette méthode de développement s'adressera plutôt aux équipes qui ont tendance à préférer les méthodes d'organisation classique et qui interviennent dans un contexte plutôt rigide. Cette méthode s'adresse principalement aux projets qui ont un cahier des charges défini avec peu de flexibilité.

Les points relevés dans le paragraphe précédent, à savoir le manque de flexibilité dans la gestion du projet et du cahier des charges, font que cette méthode n'est pas adaptée à l'équipe de Tempora et au projet actuel.

En conclusion, l'équipe de Tempora travaillera plutôt avec la méthode agile.

L'environnement de travail

Pour développer l'environnement de travail des membres de l'équipe est principalement constitué d'un poste de travail sous Linux ainsi que de nombreux outils utiles à la gestion de projet, le développement et de communication. Pour les stagiaires, la machine physique sert uniquement de terminal, car l'entreprise leur fournit une machine virtuelle Ubuntu (système d'exploitation linux) et leur permet de travailler en ssh[4].

Les outils utilisés par l'équipe et le projet

Certains des outils mentionnés ci après, ne sont pas obligatoires : les membres de l'équipe peuvent en choisir d'autres. Seuls les outils de gestion de projet et de communication sont non négociables.

Pour développer, l'équipe de Tempora utilise généralement vsCode qui peut être télécharger en suivant le lien suivant : https://code.visualstudio.com/download

Pour ce qui concerne la communication distante, elle se fera principalement par discord, depuis le serveur de la société sur le salon tempora-api-v4. L'échange de fichier quant à lui sera fait par le drive si ce n'est pas un fichier code. Pour ces derniers nous utiliserons l'outil de gestion de projet et de versionning git.

[4] Secure Shell, protocole sécurisé de communication entre machine (virtuelle et/ou physique). Cf Annexe 4 pour utiliser ssh

Pour gérer au mieux les tâches à accomplir, leur répartition et leur durée, l'équipe de Tempora utilise l'application web yookkan, disponible gratuitement en suivant le lien suivant : https://app.yookkan.com/. Dessus se trouve un tableau sur lequel les membres de l'équipe de développement sont ajoutés. Le tableau est composé de 5 colonnes :

- A faire : Pour les tâches à réaliser, qui ne sont pas encore attribuée.
- En cours : Il ne faut pas hésiter à mettre une date de début et une date de fin en fonction des capacités du développeur. Il faut aussi modifier le niveau d'avancement de chaque tâche en cours afin de permettre au chef de projet et aux autres membres de l'équipe d'avoir une idée de l'avancement de vos tâches et du projet en sa globalité. Attention : La durée d'une tâche doit être établie en fonction des capacités réelles des membres de l'équipe qui en sont chargés. De plus, si une tâche est dépendante d'autre tâche en amont ou que des cartes (tâches) en aval de celle- ci dépendent d'une fonction ou d'un morceau de code, il faut que ce dernier soit réalisé en priorité, cela évitera de prendre du retard.
- En retard : Pour les tâches qui ne sont pas terminées dans les temps. Il est vivement demandé aux membres de l'équipe de ne pas changer la date de fin d'une tâche après qu'elle a été attribuée. Cela permettra aux développeurs, et au chef de projet, d'avoir une idée de l'évolution réelle du travail accompli et de faire une meilleure estimation de temps d'un développement.
- Réalisé : Pour les tâches terminées. Attention une tâche terminée n'est pas forcément validée et peut revenir dans la case WIP.
- Validé : Pour les tâches qui sont finies et validées par le chef de projet et les alphas testeurs.

Pour la modélisation des UML, avant le développement Modelio ou plantuml seront à utiliser.

Installation de l'environnement de dev

Avant d'installer l'environnement de développement, il est demandé d'avoir installer git et de l'avoir paramétrer. L'installation et le paramétrage de git doivent se faire depuis la machine linuxe que ce soit une machine physique ou virtuelle. Pour se connecter à la VM (Machine virtuelle) en ssh voir l'annexe 2.

Installer git

Vérifier la version de git installé

```
$ git --version
```

Installer git sur le poste linuxe

```
$ sudo apt-get install git
```

Configuration locale de git

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

Attention : Remplacer John Doe par votre nom et johndoe@example.com par votre email git.

Configuration de gitlab

Pour configurer votre compte gitlab avec une clé ssh, se connecter sur gitlab et suivez la procédure suivante :

1. Aller sur votre profil (à droite) → « Edit profile »
2. Cliquer sur SSH Keys
3. Générer une clé ssh pour la VM, dans un terminal

```
$ ssh-keygen -t rsa -b 2048 -C "your@emailfrom.git"
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
$ vim .ssh/id_rsa.pub
```

4. Copier-coller la clé publique dans l'éditeur de gitlab
5. Cliquer sur « Add key »

S'il n'est pas déjà existant, il faudra créer un dossier sur le poste de développement.

```
$ mkdir directory_name
```

***Attention** : Pour les stagiaires, les identifiants de connexion à git vous sont fournis par Ange-Marie et/ou Benoit. Ce sont des identifiants uniques.*

Installation de l'environnement projet de l'api

L'installation de l'environnement de dev du projet se fait en suivant les étapes et les lignes de commandes mises sur le fichier du lien suivant :

https://gitlab.com/tempora/tempora-api/-/blob/master/README.md?ref_type=heads

Pour le fichier d'environnement (.env) de base : cf Annexe 5.

Installation de l'environnement du frontend

Pour ce projet il sera nécessaire de faire l'installation de plusieurs programmes et modules sur la VM : node, npm et angular.

Installation de NodeJs et Angular

```
$ sudo apt-get update
```

```
$ sudo apt-get install nodejs npm
```

Pour vérifier la version de node et npm faire `node -v` et `npm -v`. Pour information, il est possible d'utiliser nvm pour installer node et choisir la version que l'on veut installer. Pour installer et utiliser nvm, voir : https://snowpact.com/blog/nvm-guide-d-installation

```
$ npm install -g @angular/cli
```

Cloner le projet

Depuis le dossier précédemment créer (`cd path_to_directory`, pour s'y rendre en ligne de commande)

```
$ git clone git@gitlab.com:tempora/tempora-angular.git
```

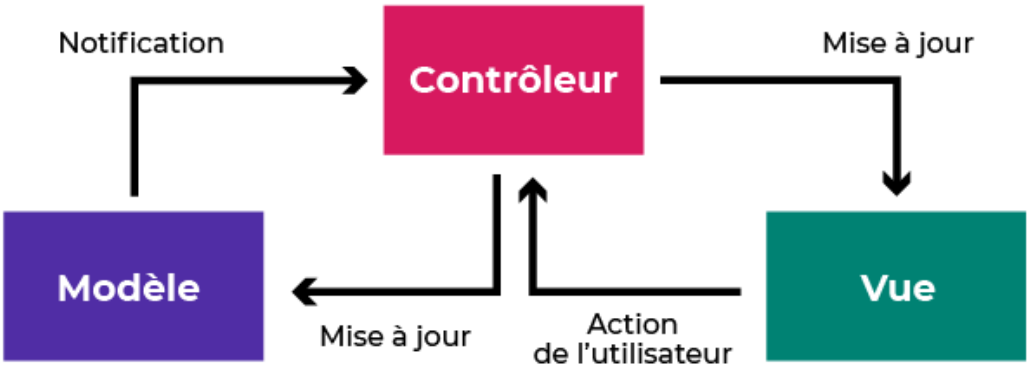
Architecture du projet

Les deux projets utilisent l'architecture MVC. Dans ce type de projet les responsabilité sont divisés entre trois composant :

- **Modèle**
  Contient les informations relatives au système, ce sont les fonctionnalités brute de l'application. Dans le cas de Tempora, les modèles contiennent les classes objet et les fonctionnalités de conversion des données.

- **Vue**
  Sert d'interface entre l'utilisateur et le programme pour transmettre les informations des modèles à l'utilisateur. Dans le cas de l'api, les vue sont uniquement pour l'envoi par mail. L'interfaçage entre le programme et l'utilisateur se fait par le biais d'un autre projet.

- **Contrôleur**
  Ce sont les contrôleurs qui garantissent que les commandes de l'utilisateur n'impact que les modèles souhaité et mettent à jour l'application. Ils servent de passerelle entre la vue et le modèle.



Dans ce projet ont va retrouver deux autres types de fichiers principaux : les dbdata et les routes.

Le projet étant une API REST, l'interface avec l'utilisateur n'est pas prise en compte par celui-ci, toutes les commandes de l'utilisateur sont envoyées par le biais de requêtes avec des url. C'est à ce moment qu'interviennent les fichiers de routes. Lorsque des requêtes d'un type et d'une url particulière sont envoyées à l'api, les fichiers de routes les interceptent et définissent quels contrôleurs doivent les traiter.

Les dbdata quant à eux, contiennent des fonctions appelées par les contrôleurs pour l'exécution de requêtes en base de données. Ce sont eux qui vont récupérer un modèle ou le modifier.

D'autres fichiers secondaires ont été créés pour le fonctionnement de l'api.

Pour avoir une idée de la communication entre les différents fichiers de la nouvelle api voir en Annexe 6.

Les fonctionnalités

Le projet va avoir deux types de fonctionnalités. Les principales qui doivent être développées en priorité et les secondaires.

Les fonctionnalités principales

La fonctionnalité principale de ce module est : l'affichage et le filtre de l'historique. L'historique pouvant contenir un grand nombre d'actions, l'utilisateur qui la consulte doit pouvoir filtrer en fonction de plusieurs paramètres :

- La période

- ➢ date du jour
- ➢ mois en cours
- ➢ mois précédent
- ➢ entre deux dates

- ❖ La catégorie (type d'objet)
- ❖ L'opération (type d'action effectué)

Les fonctionnalités secondaires

Ces fonctionnalités seront composées de nouveaux types de filtre ainsi que des potentielles nouvelles catégories proposées par les développeuses.

Développement des fonctionnalités

Certaines fonctionnalités sont déjà présentes dans la v3, elles devront être reprises en ne nécessitant pas ou peu de modification. D'autres fonctionnalités qui sont déjà présentes devront être complètement réécrite car réévaluées.

En amont (avant le développement)

Pour chaque fonctionnalité, il est demandé de réaliser une gestion des risques au niveau de la sécurité, mais aussi de son utilisation. Afin de prendre en compte tous les cas possibles, les développeurs chargés du développement de cette fonctionnalité devront réaliser des diagrammes UML à faire valider par le chef de projet. Les diagrammes demandés sont : le diagramme de séquence et le diagramme d'activités. Le diagramme de classes pourra aussi être demandé si la tâche demande l'ajout d'un nouveau modèle.

Pendant le développement

Le développement de certaines fonctionnalités seront majoritairement faites avec du copier coller, avec de l'adaptation du code de la v3 pour correspondre à la syntax et à l'architecture de la v4.

Il est demandé au développeur de mettre à jour régulièrement l'avancement de leurs tâches sur yookkan et de faire des commits, push et pull régulier sur git. Cela permettra au chef de projet et autres membres de l'équipe de suivre l'avancement du projet et d'avoir leur code à jour.

Des tests réguliers en cours de développement sont requis pendant le développement de la fonctionnalité, mais aussi après avoir fait un pull. Cela permettra de vérifier que l'intégration du code récupéré ne casse rien dans le fonctionnement de l'API.

D'autres paramètres seront aussi à prendre en compte pendant le développement d'une fonctionnalité. Par exemple :

- doit-on mettre un cache sur le résultat de la requête ?
- doit-on enregistrer l'action dans le log (historique) ?

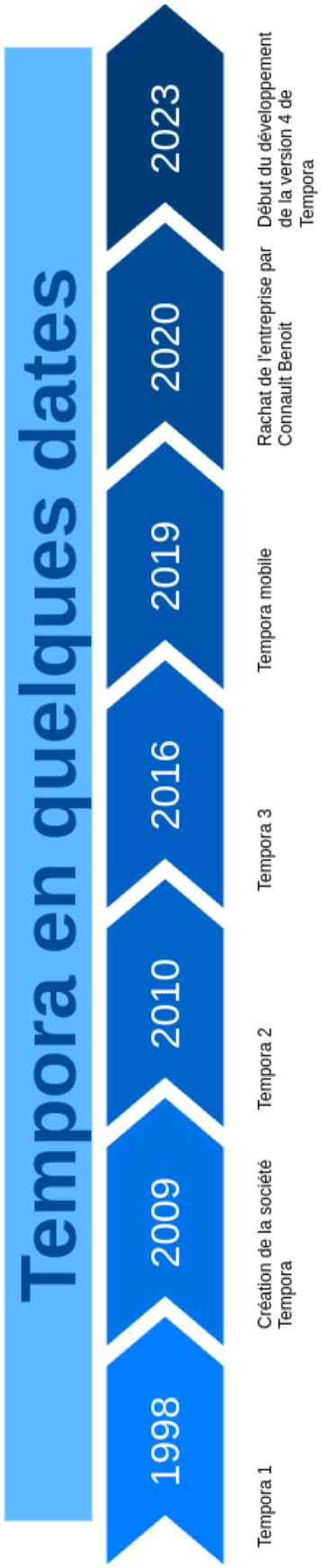
En aval (après le développement)

Une fois la tâche terminée, la fonctionnalité et l'application en elle-même doivent être testées dans un premier temps par le développeur puis par un alpha testeur pour être validé. Cette étape de validation permettra de trouver tous les potentiels bugs et de faire une revue du code. Suite à cela, si la validation a été confirmée, les personnes chargées de la tâche pourront commencer une autre itération. Dans le cas contraire, ils pourront corriger les bugs et apporter les modifications remontées, avant de soumettre à une nouvelle validation.

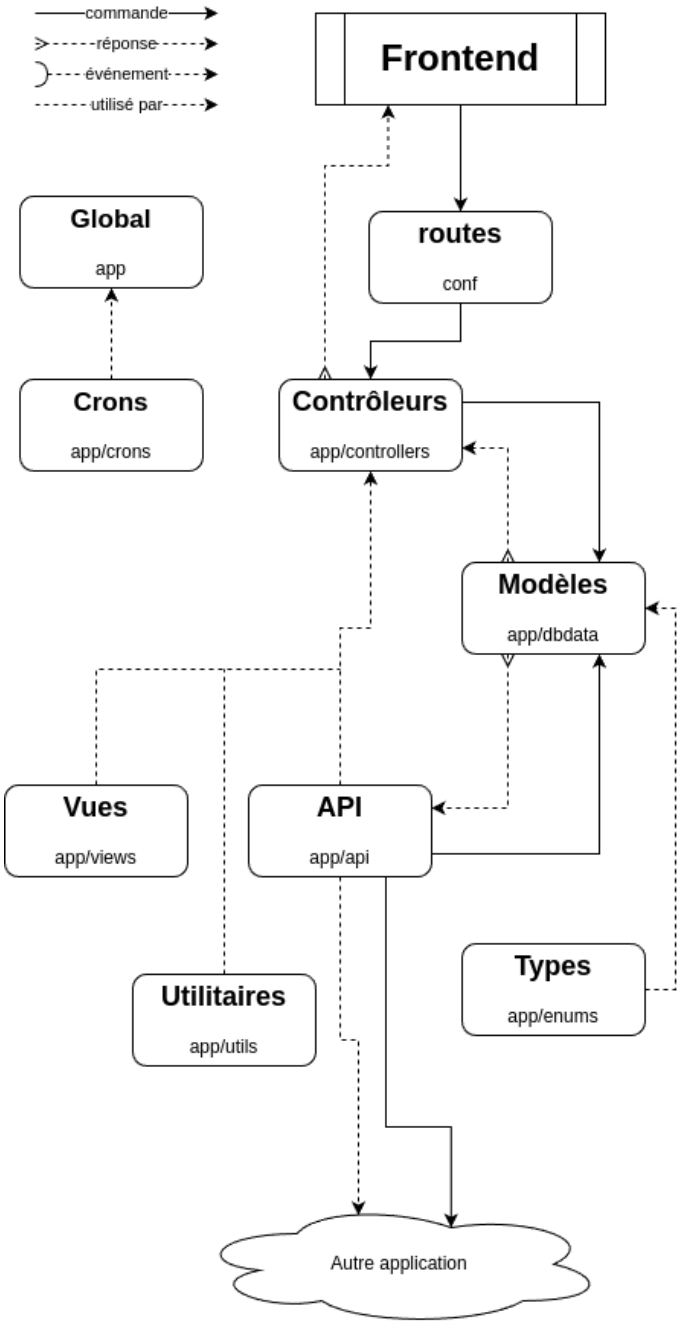
Le déroulement du stage

Pendant ce stage vous aurez le statut de développeur au sein de l'équipe. Des points réguliers seront réalisés afin non seulement d'avoir un suivi de votre stage, mais aussi du projet.
Des réunions hebdomadaires ont lieu le lundi et le vendredi. Les réunions du début de semaine ont pour but d'établir les tâches à réaliser pendant la semaine. Alors que celles du vendredi permettent d'avoir un suivi des réalisations de la semaine, des avancés et de ce qu'il reste à faire.
Pour le suivi de votre stage, un point sera réalisé le vendredi, mais si vous en avez besoin vous pouvez demander un point à tout moment.

Annexe 1 : Frise chronologique de Tempora



Annexe 2 : API 3 - Architecture



API

Les fichier d'api servent d'interfacage entre Tempora et d'autres application comme google calendar, par exemple.

Modèles

Ils sont l'interfaçage entre les contrôleurs et la base de données. Ils exécutent des requêtes en base de données en fonctions de la demande faites par le contrôleur et traduisent le résultat obtenu en objet fourni au contrôleur.

Contrôleurs

Ils sont au centre des requêtes envoyés par le frontend.

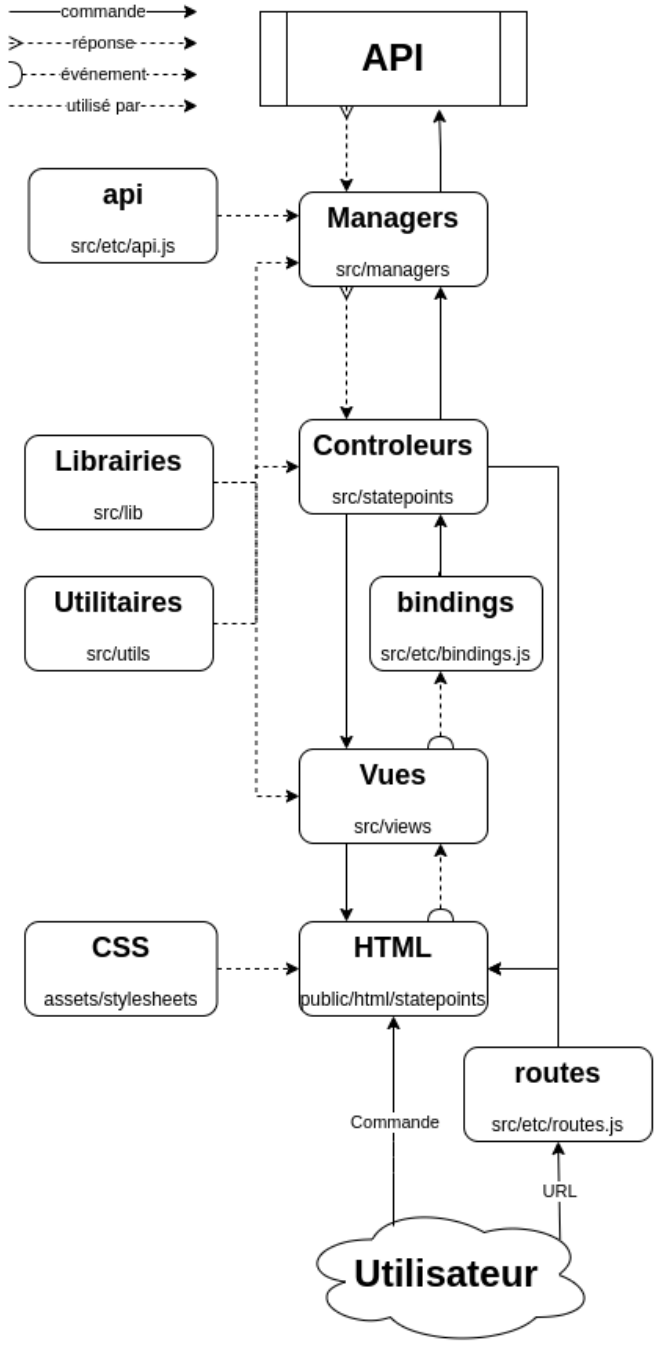
Ils reçoivent les demandes du frontend par le biais des fichier de routes et font appel au dbdata pour les executer dans la base de données.

Ils traitent ensuite les résultats obtenu et renvoi une réponse au frontend.

Vues

Les fichiers de vues sont des fichier HTML, permettant d'envoyer des informations par emails suite à une requête de l'api. Ou bien d'avoir un aperçu avant l'impression.

Annexe 3 : Frontend 3 - Architecture



Managers

Ils sont la passerelle entre les contrôleurs et le serveur API.

Un manager est instancié et appelé par un contrôleur, ce dernier souscrivant à un événement pour attendre la réponse.
Le manager transmet la demande au serveur avec une requête Ajax.
Quand il reçoit la réponse du serveur, il déclenche un événement, que le contrôleur récupère et traite.

L'adresse du serveur est fournie par api.js (en test, penser à modifier api.js pour mettre "local" à la place de "server")

Un manager n'a de référence qu'à lui-même. Il ne "connait" pas les controleurs, vues, etc...

Controleurs

Le coeur du logiciel. Chaque contrôleur gère un aspect/page de Tempora.

Les contrôleurs demandent des données aux managers, traitent les données, demandent d'afficher des données aux vues, et déclenchent le chargement d'une nouvelle page html quand besoin.

Ils reçoivent et traitent également les demandes utilisateurs à travers des bindings.

Un contrôleur "connait" et peut appeler n'importe quel manager et vue, Généralement, un contrôleur est dédié à un seul fichier html.

Vues

Une vue gère l'affichage dynamique d'une page. elle reçoit des données de la part d'un contrôleur, et les affichent en modifiant l'html.

Elle met en place également des événements pour réagir aux commandes de l'utilisateur, qui sont traitées et si besoin transmises au contrôleur en générant un événement, qui sera déclenché et transmis par les bindings.

Généralement, une vue est spécifique à un seul contrôleur, et un seul html.

Une vue ne "connait" que elle-même et l'html lié.

Annexe 4 : Connexion en ssh à une autre machine

```
$ ssh utilisateur@domainedelamachine -p nbPort
```

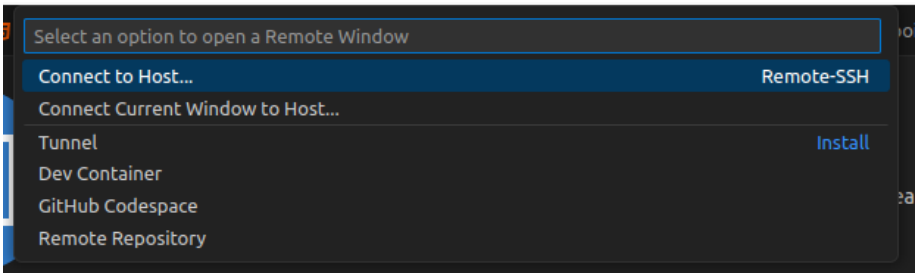
domainedelamachine : peut correspondre à l'adresse ip ou au nom de domaine de la machine sur laquelle il faut se connecter. Pour les vm de Tempora, celui ci est généralement dev.tempora.fr

L'option -p est à mettre lorsque la machine utilise un autre port pour se connecter. Au sein de Tempora, ce dernier est fourni par la personne qui crée la machine, en général Ange-Marie ou Benoit. nbPort, est à remplacer par le numéro de port fourni.

Après avoir exécuté la ligne de commande, un mot de passe pour la machine distante est demandé.

1. Pour ce connecter en ssh depuis vsCode, la ligne de commande fourni ci dessus vous sera utile. Pour initier la connexion ssh depuis vsCode et ainsi pouvoir coder directement sur la machine distante, il faut avoir installer le plugin suivant sur vsCode : Remote - SSH. Puis cliquer sur le bouton qui apparaît en bas à gauche de la fenêtre.

2. Après avoir cliquer dessus, une liste apparaît en haut il faut alors choisir "Connection to Host".



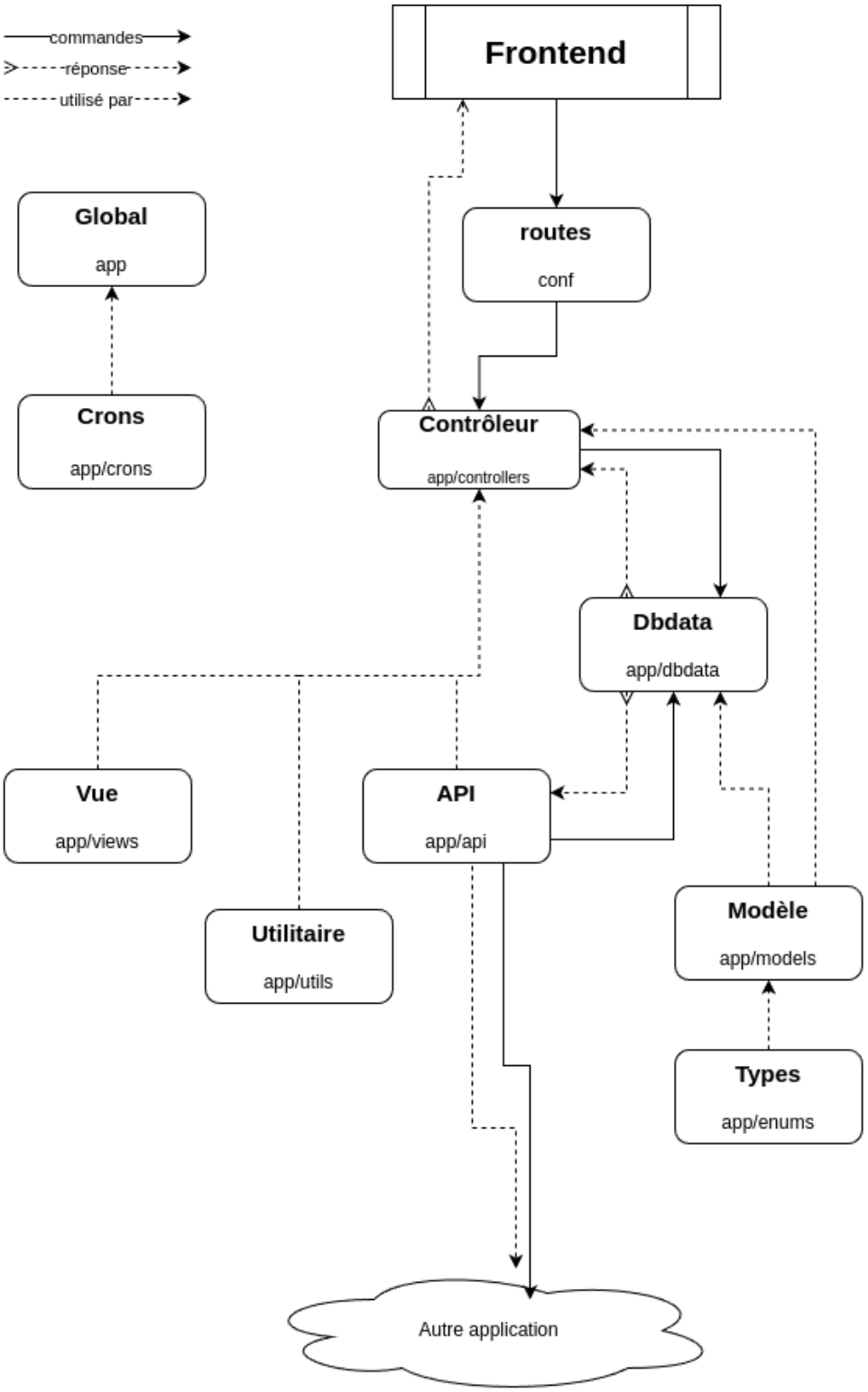
3. cliquer sur "+Add new SSH Host", c'est après cela qu'il faut saisir la ligne de commande précédente.
4. Après avoir entré la ligne de commande, il faut choisir un fichier pour enregistrer la configuration de la connexion ssh. Une fois cette opération validée, reprendre l'étape 1 et 2. Et choisir le domaine affiché, dans la liste.

Annexe 5 : Fichier d'environnement de l'API

Attention au retour à la ligne, lors du copier coller. Relire le fichier, un export par ligne

```
export POSTGRESQL_ADDON_MASTER_HOST=localhost
export POSTGRESQL_ADDON_MASTER_PORT=5432
export POSTGRESQL_ADDON_HOST=localhost
export POSTGRESQL_ADDON_PORT=5432
export POSTGRESQL_ADDON_DB=tempora
export POSTGRESQL_ADDON_USER=postgres
export POSTGRESQL_ADDON_PASSWORD=postgres

#export POSTGRESQL_ADDON_MASTER_HOST=db.tempora.fr
#export POSTGRESQL_ADDON_MASTER_PORT=5432
#export POSTGRESQL_ADDON_HOST=db.tempora.fr
#export POSTGRESQL_ADDON_PORT=5432
#export POSTGRESQL_ADDON_DB=tempora
#export POSTGRESQL_ADDON_USER=postgres
#export POSTGRESQL_ADDON_PASSWORD=Hd25x03vU8

# export POSTGRESQL_ADDON_MASTER_HOST=dev.tempora.fr
# export POSTGRESQL_ADDON_MASTER_PORT=5431
# export POSTGRESQL_ADDON_HOST=dev.tempora.fr
# export POSTGRESQL_ADDON_PORT=5431
# export POSTGRESQL_ADDON_USER=tempora
# export POSTGRESQL_ADDON_PASSWORD=UGfhHYGkd8MTpgOyRehu

export SMTP_HOST=in-v3.mailjet.com
export SMTP_PASSWORD=5780a8f90ae0e3f3668cb3bf49d41b24
export SMTP_PORT=587
export SMTP_USER=dc94e08e353dcc2c047e663d0d813e83

export EXTERNAL_HOST="http://localhost:9003"
export EXTERNAL_ANSWER="https://devapi2.tempora.fr"

export SMS_BASE_TOKEN=vl8SfaAYcBLB8XiOTogkBJ9bH5f8qG7w
export SMS_OTP_TOKEN=XKMCoLBXEiErahKelX8kWlhvMeVQfaGh

export
GOOGLE_CLIENT_ID="95853063798-cavqppabgm25qm21ttebeg6lndo481ae.apps.googleusercontent.co
m"
export GOOGLE_CLIENT_SECRET="DJGBtYphFGEGjbShnac2Qu2o"

export MEDAVIZ_CLIENT_ID="6_3d3yg7heluww40w00so0sscowskc88cksw4kwckk4swckkgsw8"
export MEDAVIZ_CLIENT_SECRET="28u8oy2y9k74s44w00kkwkw8so0o088wg0swocowgg8wwkgssg"

export CRONJOBS=false
export MOCK=true

export PUSH_NOTIFICATION_HOST="https://fcm.googleapis.com/fcm/send"
export
PUSH_NOTIFICATION_TOKEN="key=AAAAFlFJrnY:APA91bE6R4mTnNoEKOIiz_18MKvi4w1DiNLPl-scEpjiu67
hfZTKtoWY7pbbmD-0jIqpwpKwpd13BbFWN9SGBSIUDek3gW0_LogutkG7RcUFUefUHexgZvseUZnXecOquM8T-Ki
QcJU9mors6OCZNr95hjPmQds7sg"
```

Annexe 6 : API 4 - Communication des fichier



API

Les fichier d'api servent d'interfaçage entre Tempora et d'autre application comme google calendar par exemple

Dbdata

Il sont l'interfaçage entre les contrôleurs et la base de données en fonction des demandes faites par le premier.

Modèle

Ils sont les fichier servant à formater des données et traduisent le résultat obtenu par les dbdata en objet fourni au contrôleur

Contrôleur

Ils sont au centre des requêtes envoyés par le frontend.

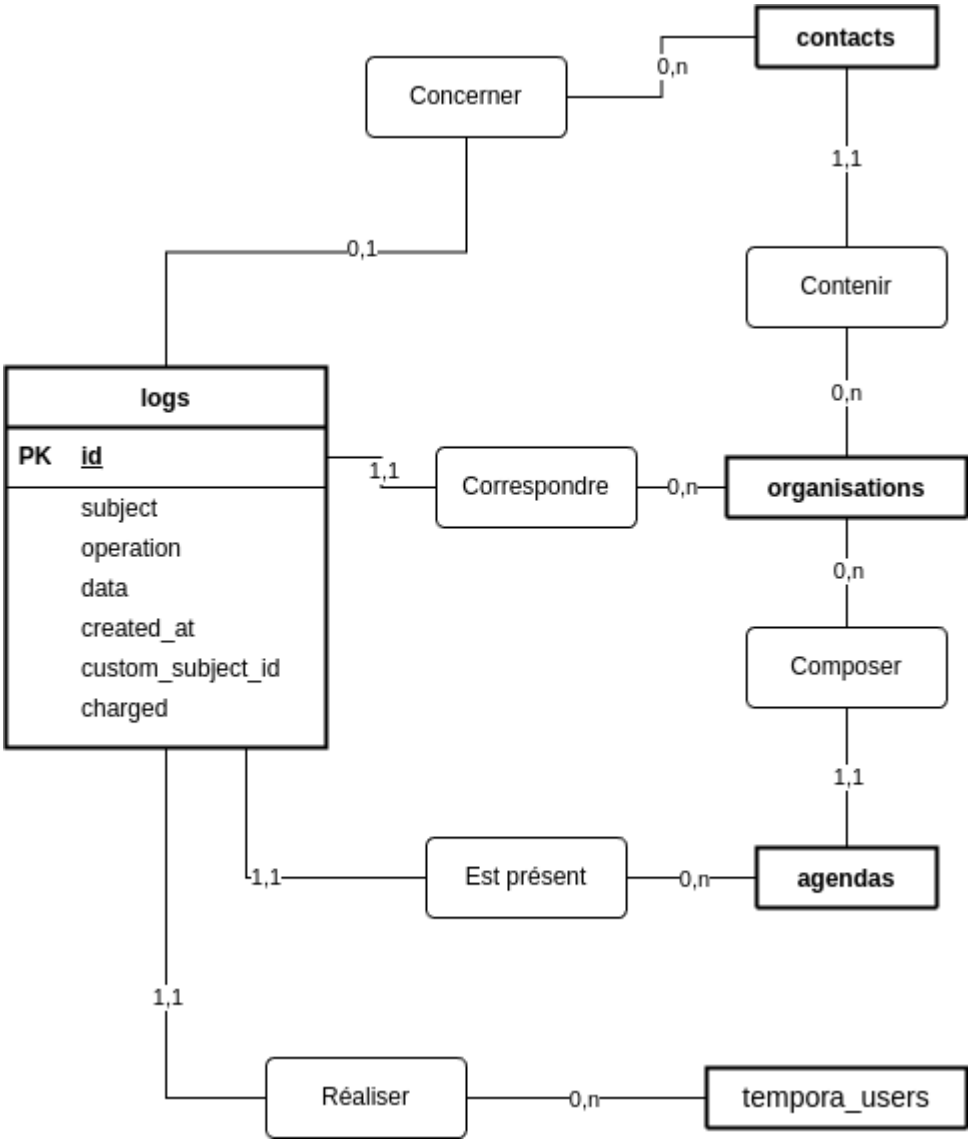
Ils reçoivent les demandes du frontend par le biais des fichiers de routes et font appel au dbdata pour les executer dans la base de données.

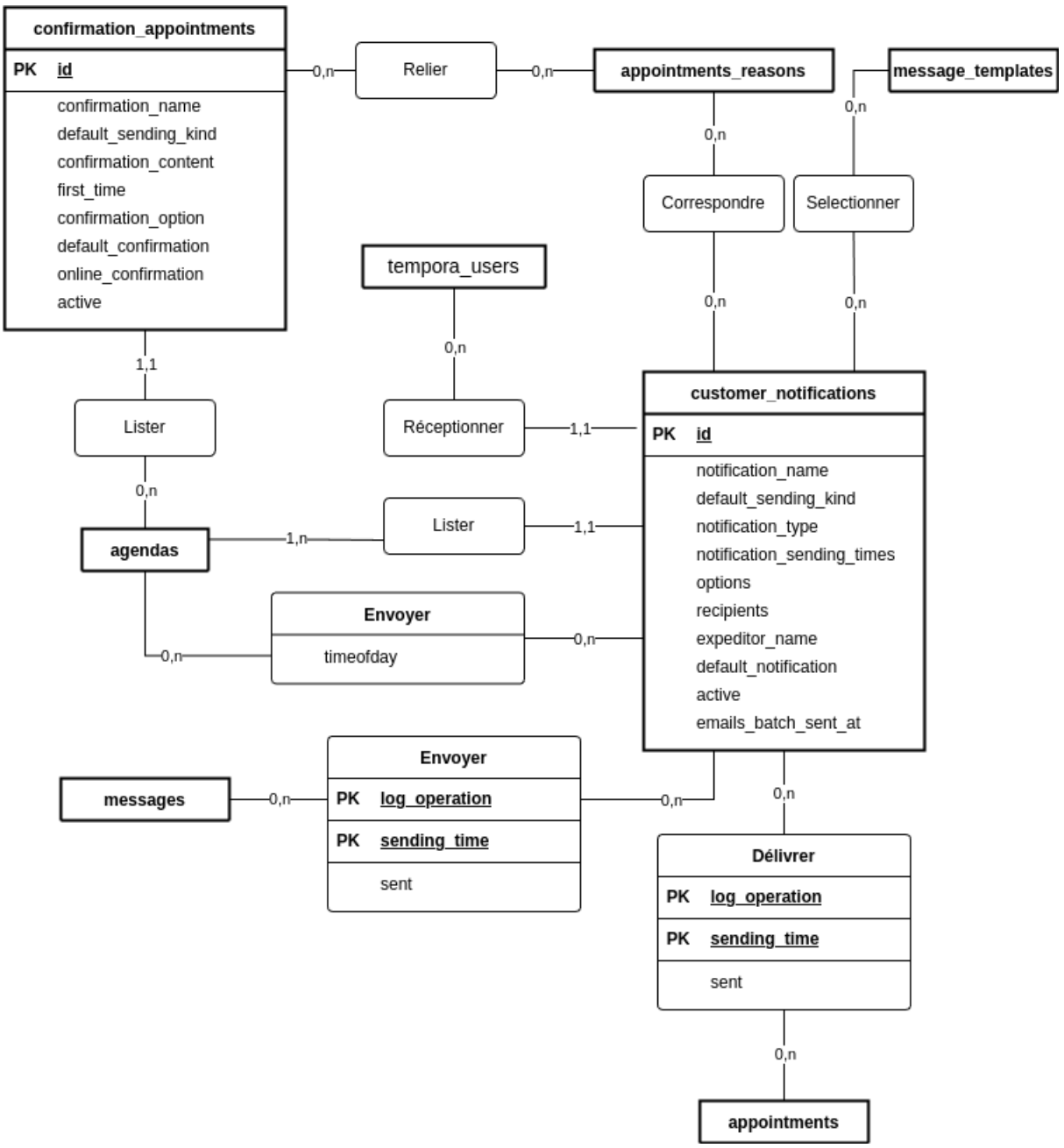
Ils traitent ensuite les resultats obtenu et renvoi une réponse au frontend

Vue

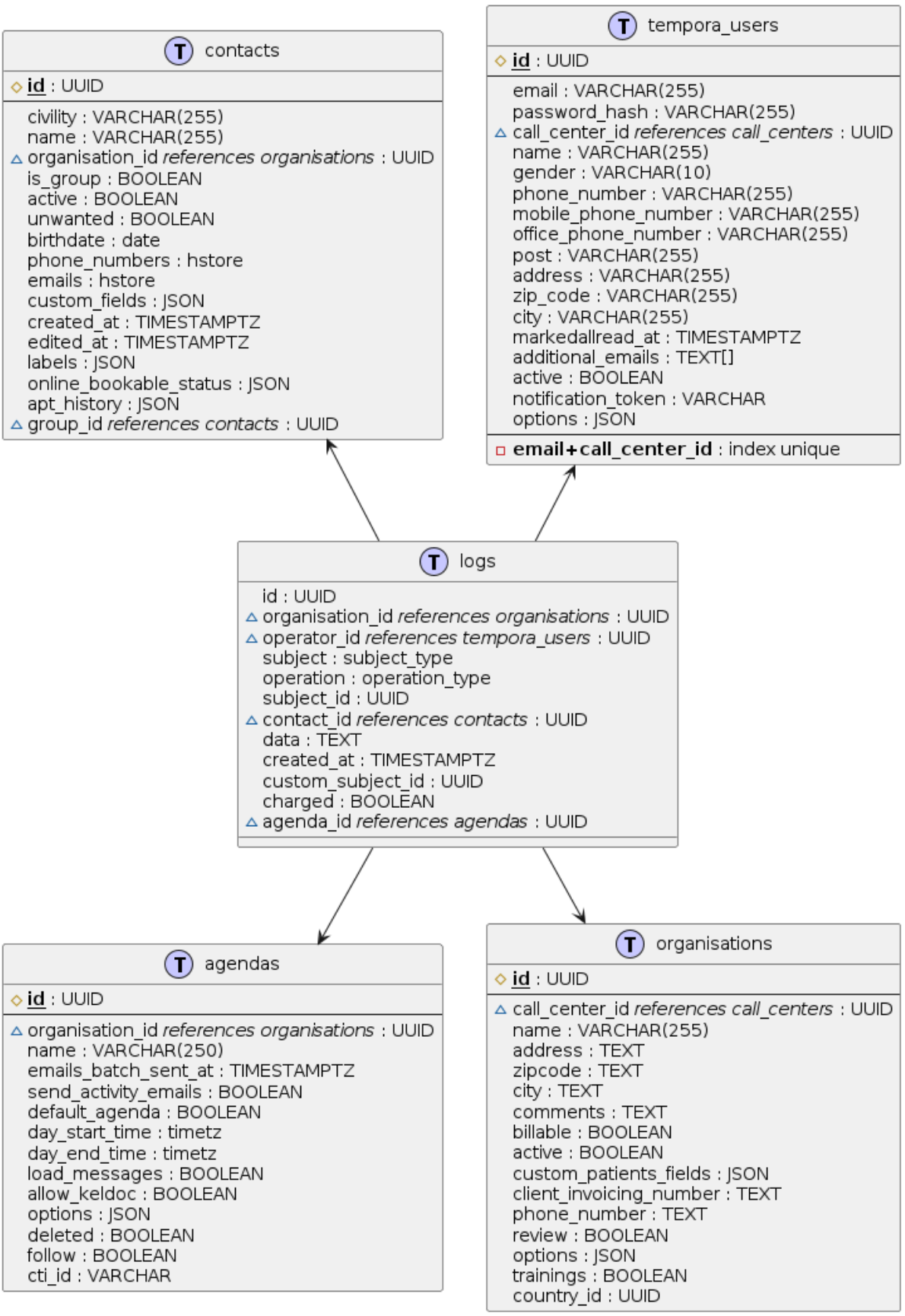
Les fichier de vues sont des fichier HTML, permettant d'envoyer des informations par email suite à une requêtes de l'api. Ou bien d'avoir un aperçu avant l'impression.

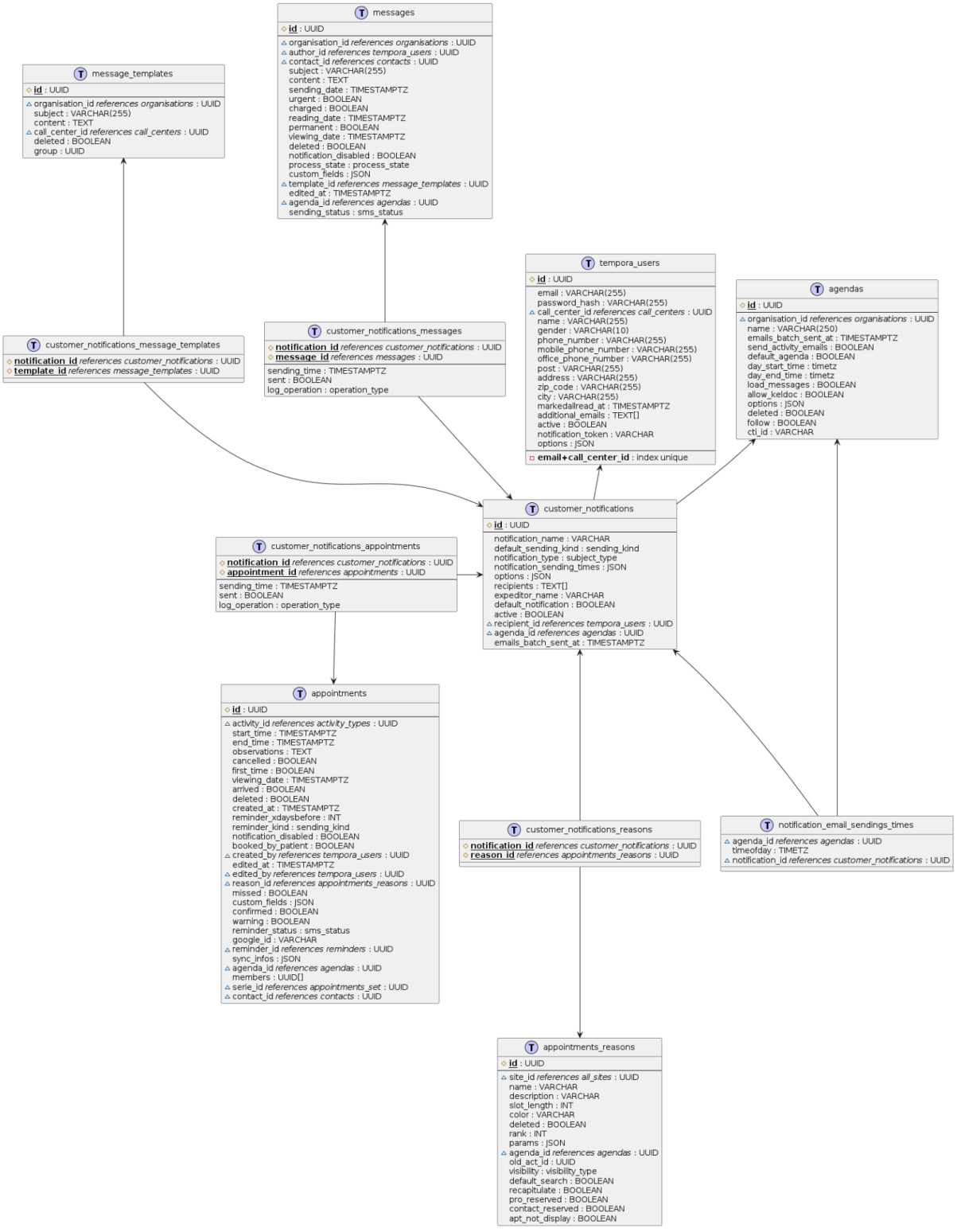
Annexe 7 : MCD - Historique

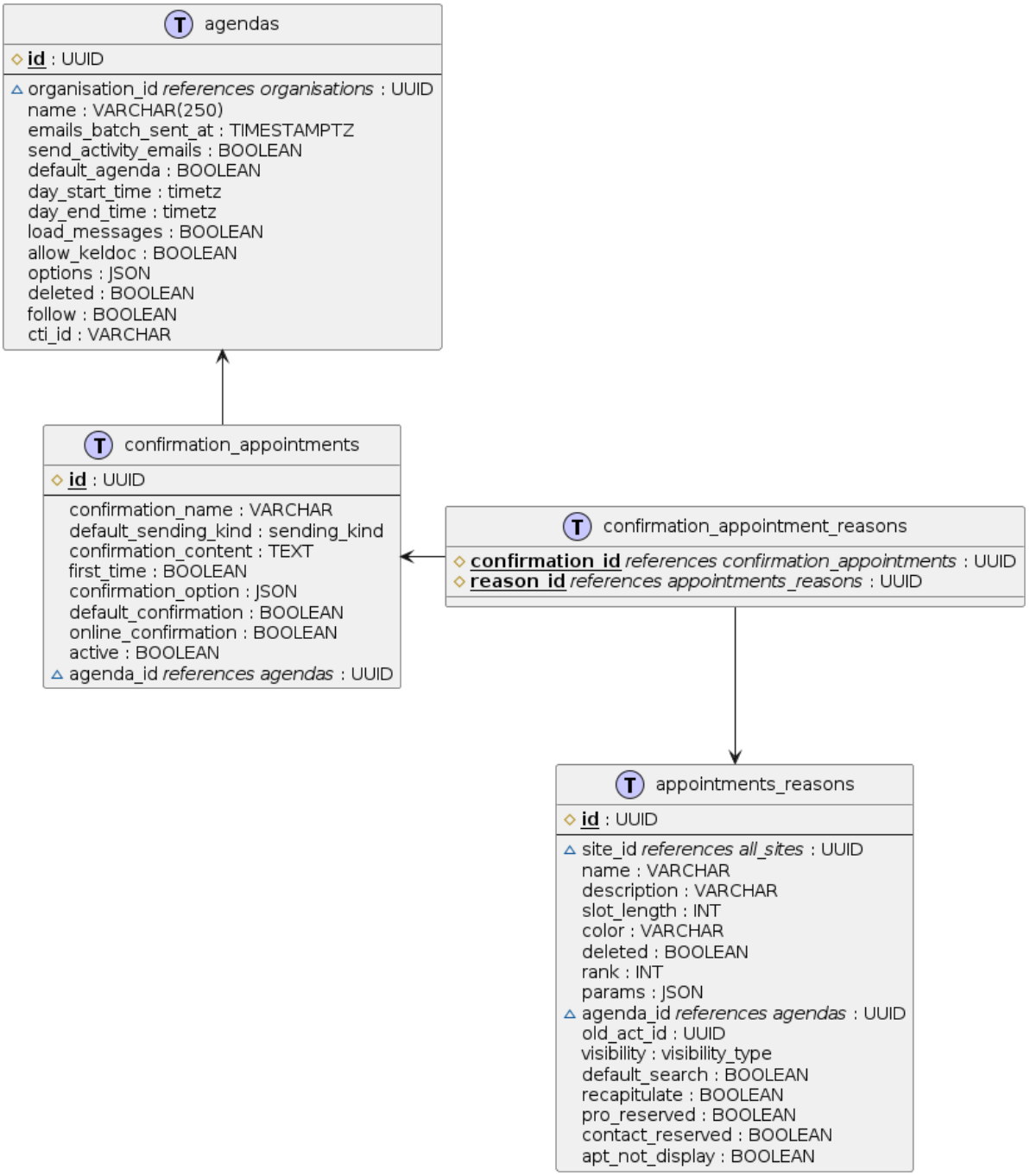



confirmation_appointments
PK id
confirmation_name
default_sending_kind
confirmation_content
first_time
confirmation_option
default_confirmation
online_confirmation
active
0,n
Relier
0,n
appointments_reasons
message_templates
0,n
0,n
Correspondre
Selectionner
tempora_users
0,n
0,n
1,1
0,n
Lister
Réceptionner
1,1
customer_notifications
PK id
notification_name
default_sending_kind
notification_type
notification_sending_times
options
recipients
expeditor_name
default_notification
active
emails_batch_sent_at
0,n
agendas
1,n
Lister
1,1
Envoyer
timeofday
0,n
0,n
messages
0,n
Envoyer
PK log_operation
PK sending_time
sent
0,n
0,n
Délivrer
PK log_operation
PK sending_time
sent
0,n
appointments

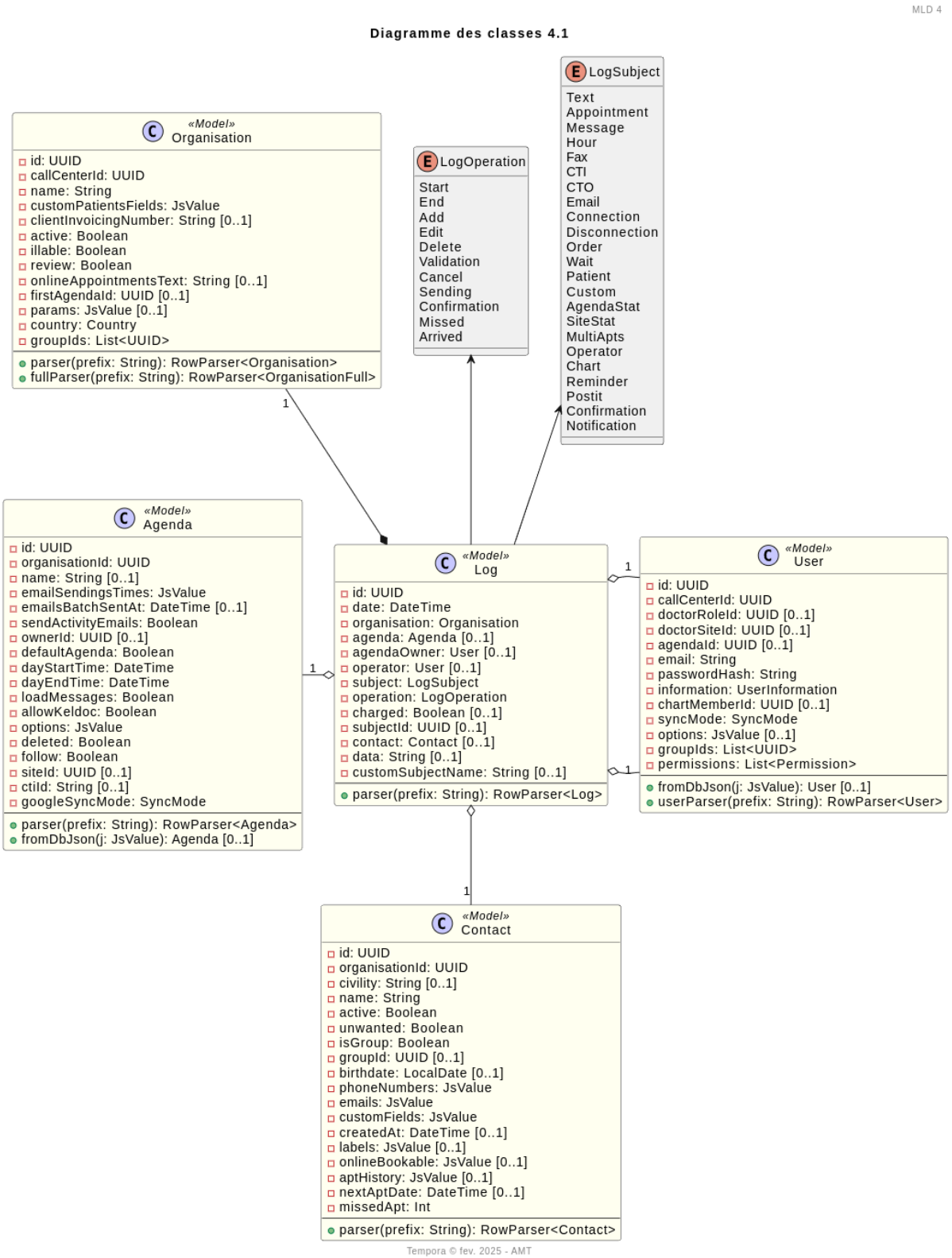
Annexe 8 : MLD - Historique

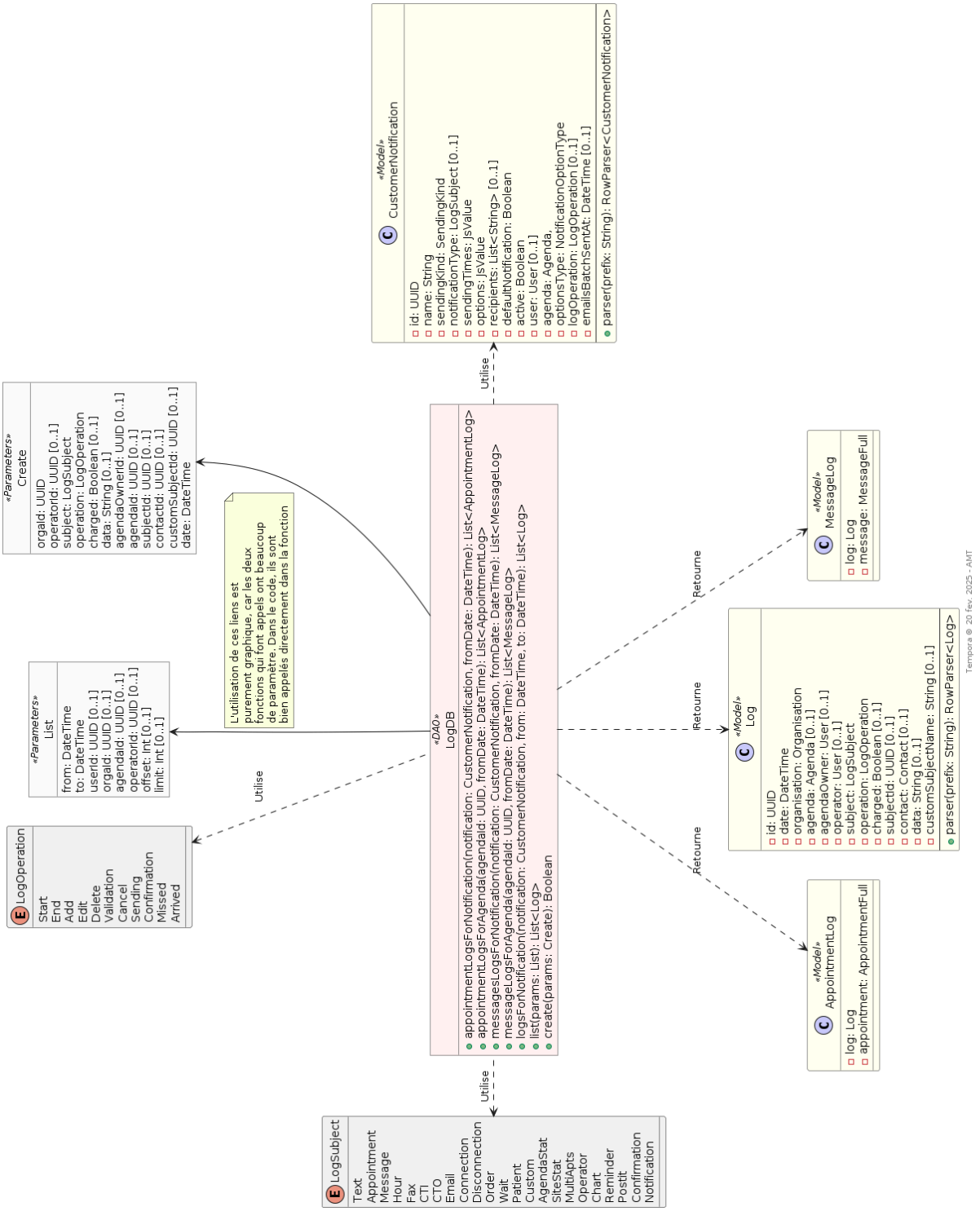



messages
id : UUID
organisation_id references organisations : UUID
author_id references tempora_users : UUID
contact_id references contacts : UUID
subject : VARCHAR(255)
content : TEXT
sending_date : TIMESTAMPTZ
urgent : BOOLEAN
charged : BOOLEAN
reading_date : TIMESTAMPTZ
permanent : BOOLEAN
viewing_date : TIMESTAMPTZ
deleted : BOOLEAN
notification_disabled : BOOLEAN
process_state : process_state
custom_fields : JSON
template_id references message_templates : UUID
edited_at : TIMESTAMPTZ
agenda_id references agendas : UUID
sending_status : sms_status
message_templates
id : UUID
organisation_id references organisations : UUID
subject : VARCHAR(255)
content : TEXT
call_center_id references call_centers : UUID
deleted : BOOLEAN
group : UUID
tempora_users
id : UUID
email : VARCHAR(255)
password_hash : VARCHAR(255)
call_center_id references call_centers : UUID
name : VARCHAR(255)
gender : VARCHAR(10)
phone_number : VARCHAR(255)
mobile_phone_number : VARCHAR(255)
office_phone_number : VARCHAR(255)
post : VARCHAR(255)
address : VARCHAR(255)
zip_code : VARCHAR(255)
city : VARCHAR(255)
markedallread_at : TIMESTAMPTZ
additional_emails : TEXT[]
active : BOOLEAN
notification_token : VARCHAR
options : JSON
email+call_center_id : index unique
agendas
id : UUID
organisation_id references organisations : UUID
name : VARCHAR(250)
emails_batch_sent_at : TIMESTAMPTZ
send_activity_emails : BOOLEAN
default_agenda : BOOLEAN
day_start_time : timetz
day_end_time : timetz
load_messages : BOOLEAN
allow_keldoc : BOOLEAN
options : JSON
deleted : BOOLEAN
follow : BOOLEAN
cti_id : VARCHAR
customer_notifications_message_templates
notification_id references customer_notifications : UUID
template_id references message_templates : UUID
customer_notifications_messages
notification_id references customer_notifications : UUID
message_id references messages : UUID
sending_time : TIMESTAMPTZ
sent : BOOLEAN
log_operation : operation_type
customer_notifications
id : UUID
notification_name : VARCHAR
default_sending_kind : sending_kind
notification_type : subject_type
notification_sending_times : JSON
options : JSON
recipients : TEXT[]
expeditor_name : VARCHAR
default_notification : BOOLEAN
active : BOOLEAN
recipient_id references tempora_users : UUID
agenda_id references agendas : UUID
emails_batch_sent_at : TIMESTAMPTZ
customer_notifications_appointments
notification_id references customer_notifications : UUID
appointment_id references appointments : UUID
sending_time : TIMESTAMPTZ
sent : BOOLEAN
log_operation : operation_type
appointments
id : UUID
activity_id references activity_types : UUID
start_time : TIMESTAMPTZ
end_time : TIMESTAMPTZ
observations : TEXT
cancelled : BOOLEAN
first_time : BOOLEAN
viewing_date : TIMESTAMPTZ
arrived : BOOLEAN
deleted : BOOLEAN
created_at : TIMESTAMPTZ
reminder_xdaysbefore : INT
reminder_kind : sending_kind
notification_disabled : BOOLEAN
booked_by_patient : BOOLEAN
created_by references tempora_users : UUID
edited_at : TIMESTAMPTZ
edited_by references tempora_users : UUID
reason_id references appointments_reasons : UUID
missed : BOOLEAN
custom_fields : JSON
confirmed : BOOLEAN
warning : BOOLEAN
reminder_status : sms_status
google_id : VARCHAR
reminder_id references reminders : UUID
sync_infos : JSON
agenda_id references agendas : UUID
members : UUID[]
serie_id references appointments_set : UUID
contact_id references contacts : UUID
customer_notifications_reasons
notification_id references customer_notifications : UUID
reason_id references appointments_reasons : UUID
notification_email_sendings_times
agenda_id references agendas : UUID
timeofday : TIMETZ
notification_id references customer_notifications : UUID
appointments_reasons
id : UUID
site_id references all_sites : UUID
name : VARCHAR
description : VARCHAR
slot_length : INT
color : VARCHAR
deleted : BOOLEAN
rank : INT
params : JSON
agenda_id references agendas : UUID
old_act_id : UUID
visibility : visibility_type
default_search : BOOLEAN
recapitulate : BOOLEAN
pro_reserved : BOOLEAN
contact_reserved : BOOLEAN
apt_not_display : BOOLEAN


agendas
id : UUID
organisation_id references organisations : UUID
name : VARCHAR(250)
emails_batch_sent_at : TIMESTAMPTZ
send_activity_emails : BOOLEAN
default_agenda : BOOLEAN
day_start_time : timetz
day_end_time : timetz
load_messages : BOOLEAN
allow_keldoc : BOOLEAN
options : JSON
deleted : BOOLEAN
follow : BOOLEAN
cti_id : VARCHAR
confirmation_appointments
id : UUID
confirmation_name : VARCHAR
default_sending_kind : sending_kind
confirmation_content : TEXT
first_time : BOOLEAN
confirmation_option : JSON
default_confirmation : BOOLEAN
online_confirmation : BOOLEAN
active : BOOLEAN
agenda_id references agendas : UUID
confirmation_appointment_reasons
confirmation_id references confirmation_appointments : UUID
reason_id references appointments_reasons : UUID
appointments_reasons
id : UUID
site_id references all_sites : UUID
name : VARCHAR
description : VARCHAR
slot_length : INT
color : VARCHAR
deleted : BOOLEAN
rank : INT
params : JSON
agenda_id references agendas : UUID
old_act_id : UUID
visibility : visibility_type
default_search : BOOLEAN
recapitulate : BOOLEAN
pro_reserved : BOOLEAN
contact_reserved : BOOLEAN
apt_not_display : BOOLEAN

Annexe 9 : Diagramme des classes




«Model»
CustomerNotification
id: UUID
name: String
sendingKind: SendingKind
notificationType: LogSubject [0..1]
sendingTimes: jsValue
options: jsValue
recipients: List<String> [0..1]
defaultNotification: Boolean
active: Boolean
user: User [0..1]
agenda: Agenda,
optionsType: NotificationOptionType
logOperation: LogOperation [0..1]
emailsBatchSentAt: DateTime [0..1]
parser(prefix: String): RowParser<CustomerNotification>
Utilise
«Parameters»
Create
orgaId: UUID
operatorId: UUID [0..1]
subject: LogSubject
operation: LogOperation
charged: Boolean [0..1]
data: String [0..1]
agendaOwnerId: UUID [0..1]
agendaId: UUID [0..1]
subjectId: UUID [0..1]
contactId: UUID [0..1]
customSubjectId: UUID [0..1]
date: DateTime
L'utilisation de ces liens est
purement graphique, car les deux
fonctions qui font appels ont beaucoup
de parametre. Dans le code, ils sont
bien appelés directement dans la fonction
«Parameters»
List
from: DateTime
to: DateTime
userId: UUID [0..1]
orgaId: UUID [0..1]
agendaId: UUID [0..1]
operatorId: UUID [0..1]
offset: Int [0..1]
limit: Int [0..1]
«DAO»
LogDB
appointmentLogsForNotification(notification: CustomerNotification, fromDate: DateTime): List<AppointmentLog>
appointmentLogsForAgenda(agendaId: UUID, fromDate: DateTime): List<AppointmentLog>
messagesLogsForNotification(notification: CustomerNotification, fromDate: DateTime): List<MessageLog>
messageLogsForAgenda(agendaId: UUID, fromDate: DateTime): List<MessageLog>
logsForNotification(notification: CustomerNotification, from: DateTime, to: DateTime): List<Log>
list(params: List): List<Log>
create(params: Create): Boolean
Utilise
Utilise
Retourne
Retourne
Retourne
LogOperation
Start
End
Add
Edit
Delete
Validation
Cancel
Sending
Confirmation
Missed
Arrived
«Model»
Log
id: UUID
date: DateTime
organisation: Organisation
agenda: Agenda [0..1]
agendaOwner: User [0..1]
operator: User [0..1]
subject: LogSubject
operation: LogOperation
charged: Boolean [0..1]
subjectId: UUID [0..1]
contact: Contact [0..1]
data: String [0..1]
customSubjectName: String [0..1]
parser(prefix: String): RowParser<Log>
«Model»
MessageLog
log: Log
message: MessageFull
«Model»
AppointmentLog
log: Log
appointment: AppointmentFull
LogSubject
Text
Appointment
Message
Hour
Fax
CTI
CTO
Email
Connection
Disconnection
Order
Wait
Patient
Custom
AgendaStat
SiteStat
MultiApts
Operator
Chart
Reminder
Postit
Confirmation
Notification
Tempora © 20 fev. 2025 - AMT