PAC Time Report

Table des matières

Project Overview

Nom : My Virtual Planner (mvp) **Personne ressource** : P. Vanderheyden

Développeur : Sana Marc-Henry **Commanditaire** : P. Vanderheyden

Lien Github : https://github.com/M-H92/TimeReport

Technologies utilisées :

- MySQL, Java, Maven, Hibernate, MQTT, JSON, Swing

Contextualisation :

- Dans une entreprise, chaque employé doit rentrer une fiche de prestation (timesheet) reprenant la tâche réalisée ainsi que le temps passé.
- Ceci permettant de calculer le nombre d'heures et le coût de chaque opération.

Demande :

- Développer une interface permettant d'optimiser l'encodage des fiches de prestation.
- Utilisation de technologies vues au cours : MySQL, Java, Maven, Hibernate, MQTT.
- Fournir un rapport écrit (ce document)

Solution :

- Implémentation d'une base de données MySQL capable de prendre en charge les employés de l'entreprise, les différentes tâches à remplir et les fiches de prestations.
- Développement d'une application « serveur » ayant pour tâche de récupérer et sauvegarder en base de données des fiches de prestation formatées. Dans ce but,

l'application « serveur » utilisera les technologies Hibernate, MQTT et JSON.

- Développement d'une application « client » permettant d'encoder *facilement* une fiche de prestation et de l'envoyer vers l'application « serveur ». Pour cela, les technologies suivantes seront utilisées : MQTT, SWING, JSON.

- Une application « POC » (Proof Of Concept) simulant l'utilisation d'un système RFID, bien qu'out of scope, sera ajoutée sous réserve qu'assez de temps soit disponible.

Nota bene :

- L'utilisation de la technologie JSON, bien que n'étant pas requise par le professeur P. Vanderheyden, a été intégrée au projet pour simplifier le transfert d'informations via MQTT et la lecture de ces informations pendant les sessions de test. L'intégration de JSON au projet est faite via les librairies Jackson dont la documentation est disponible ici.

- L'utilisation de la technologie SWING (et AWT) pour l'application client a pour but de rendre plus simple l'encodage d'une fiche de prestation. À noter que l'implémentation de cette technologie est sommaire et largement sujette à amélioration. Le choix a néanmoins été fait de ne pas investir trop de temps sur cette technologie car elle n'était ni demandée ni requise ainsi que fortement dépassée sur le marché de l'emploi actuel. Ce dernier point peut remettre en question la légitimité du choix de swing pour l'interface graphique. À cela, il sera donné comme réponse que le développeur de l'application disposait déjà de bases en lui permettant de fournir une implémentation rapide et fonctionnelle bien que peu esthétique.

- En comparaison au schéma de base de données fournie par la personne ressource, le schéma de la base de données MVP ne dispose pas de table distinctes « Project » et « Activities ».
  Effectivement, les informations contenues dans ces deux tables étant très proches, il semblait redondant de les implémenter toutes. Une unique table « Task » sera préférée.
  Cette table contient une référence vers elle-même permettant de créer des arbres de tâches et de différencier un projet d'une activité.
  En pratique, si une tâche ne référence aucune tâche parente, il s'agit d'un projet.

- Le POC RFID représente l'utilisation d'un boitier de lecture RFID permettant l'utilisation d'un chariot élévateur dans l'entrepôt X.
  L'employé dispose d'une carte RFID nominative.
  Déposer la carte sur le boitier allume le chariot élévateur et récupère la date et l'heure de début d'utilisation.

Le retrait de la carte récupère la date et l'heure de fin d'utilisation, compile les données récupérées en une timesheet dont la tâche est programmée à l'avance dans la machine (rangement entrepôt X) et dont l'employé est remplacé par l'id de la carte.
Une fois cela fait, la timesheet est envoyée sur le serveur et ajoutée en base de données.
L'ajout de ce poc prouve l'extensibilité du projet en terme de nouvelles fonctionnalités et d'optimisation de l'automatisation du remplissage/ dépôt de fiches de prestation.

- L'interface graphique a été grandement calquée sur celle fournie sur le site de geeksforgeeks.

Use case

MVP client :

L'employé ne peut faire que deux choses via l'application « client » :

Écrire une fiche de prestation,

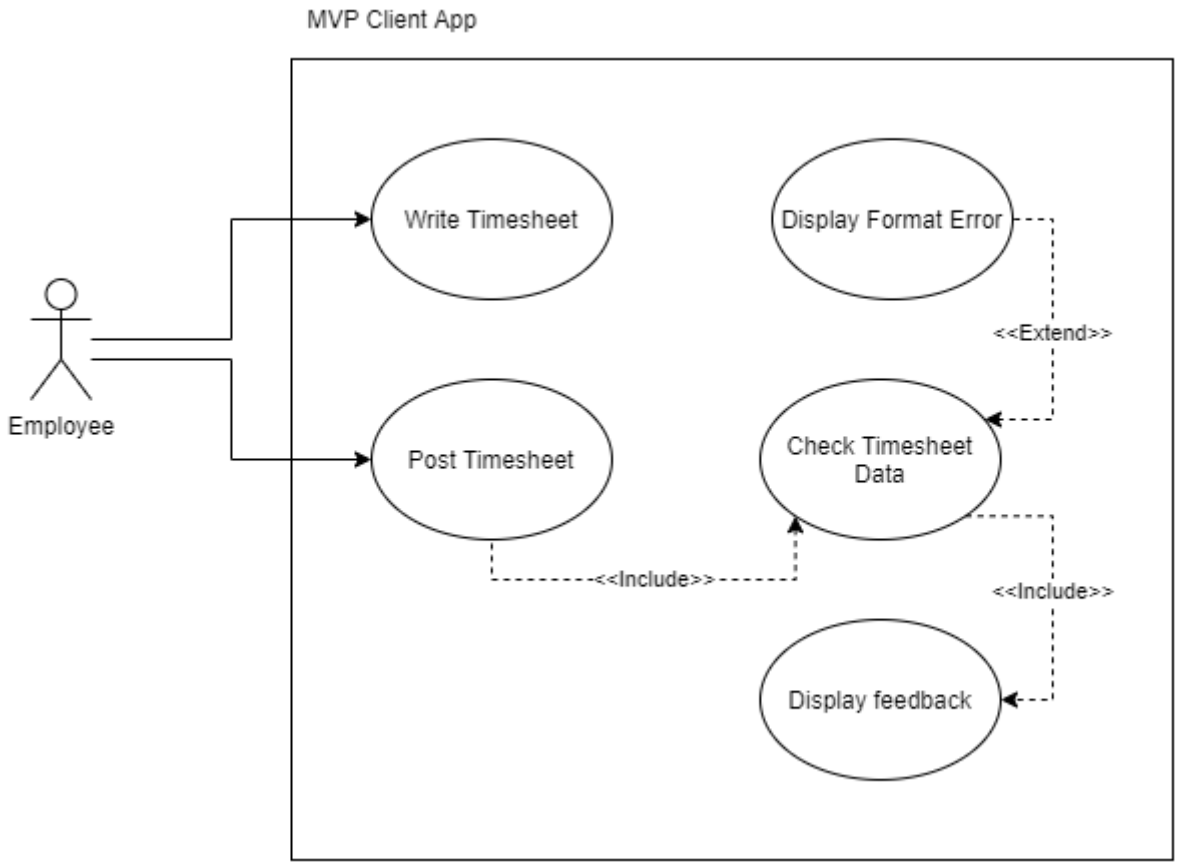
Envoyer une fiche de prestation vers le serveur.

Avant d'effectivement envoyer la fiche de prestation vers le serveur, l'application vérifie le format des données.

Si le format n'est pas respecté, affichage d'une erreur d'encodage,

Sinon, la fiche est envoyée sur le serveur et le programme attend une réponse.

Une fois la réponse arrivée, le résultat est affiché à l'utilisateur.



MVP Serveur :

L'application client interagit avec l'application serveur.

Le client dispose de deux types d'interactions :

Demander une liste de tâches

Envoyer une Fiche de prestation.

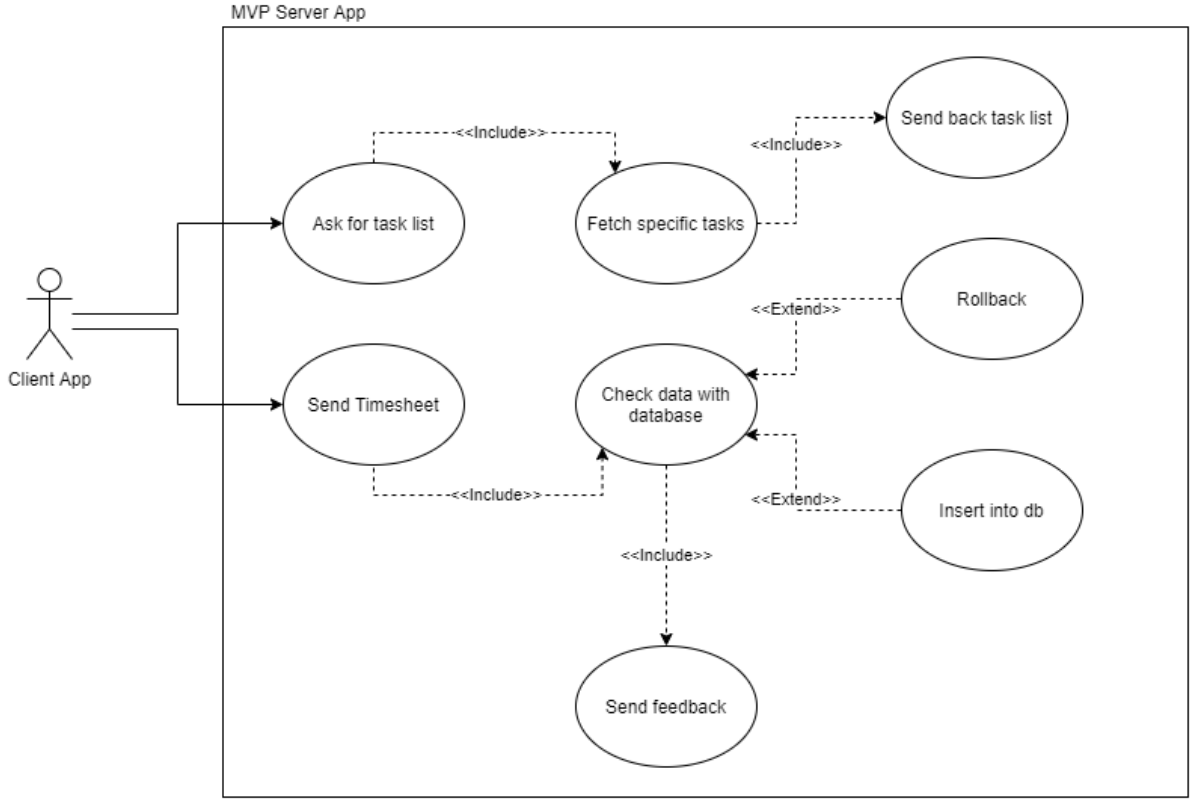
Quand il demande une liste de tâches, le client utilise un mot clé permettant de récupérer uniquement les informations intéressantes.

Quand il envoie une fiche de prestation, le serveur tente de l'insérer en base de données.

Cette opération échoue si les données ne sont pas adéquates.

L'application fait alors un rollback.

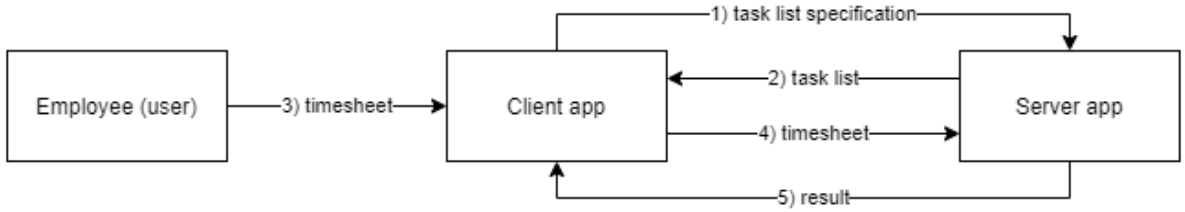
Dans tous les cas, un feedback est envoyé à l'application client.



Communication

Le diagramme de communication met en évidence le peu de messages communiqués.

1) L'application client envoie un mot clé pour récupérer une liste de tâches
2) Le serveur retourne une liste de tâches spécifiques récupérées depuis la base de données
3) Le client encode une fiche de prestation
4) L'application client transfère la fiche de prestation au serveur
5) Le serveur communique un message pour confirmer/infirmer l'enregistrement de la fiche

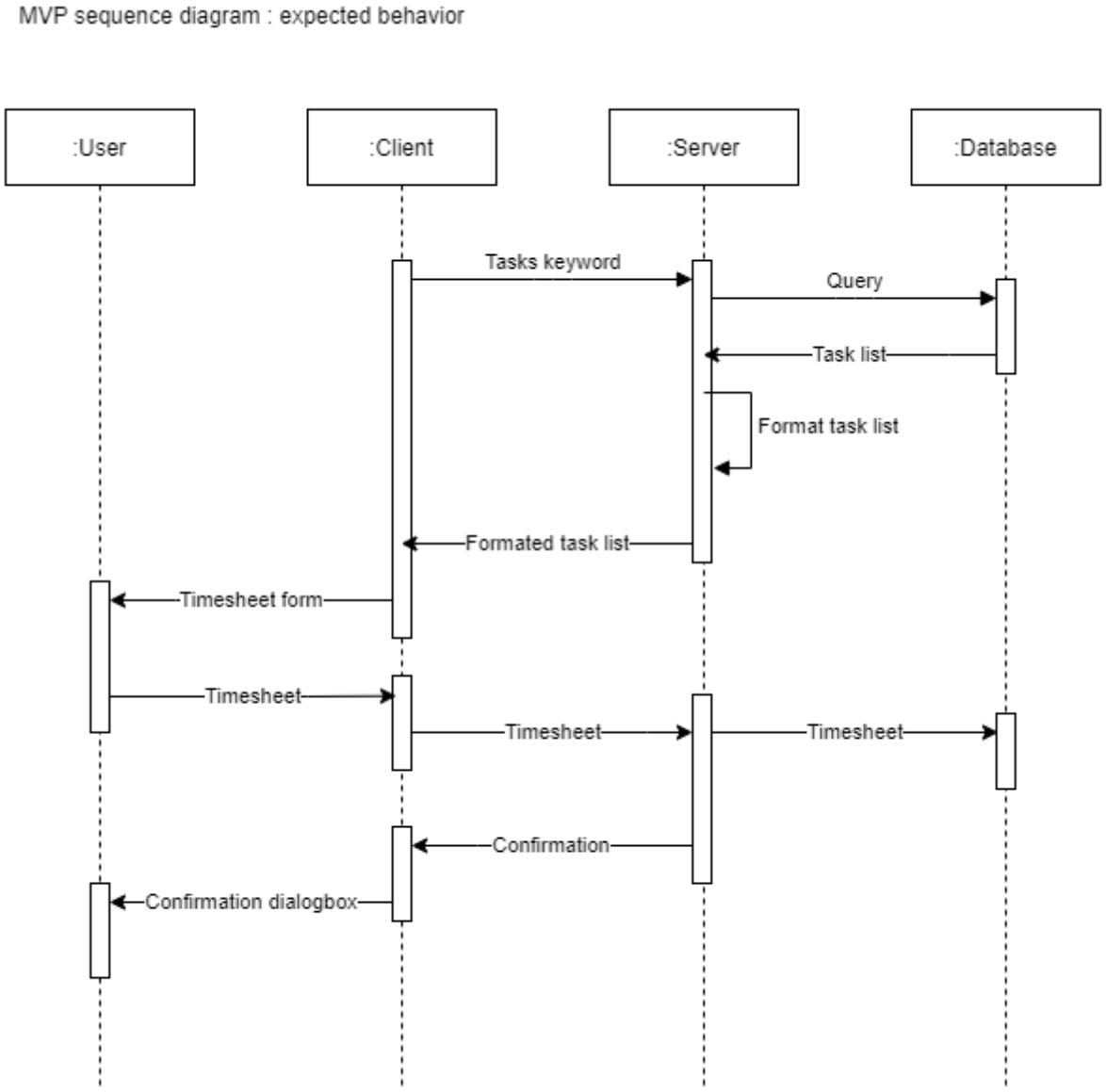


Sequence

Le diagramme de séquence ci-dessous représente le comportement « normal » de l'application lorsqu'elle est utilisée « comme prévu ».

Lorsque l'exécution de l'application ne se déroule pas comme prévu, le comportement général est d'arrêter l'application après avoir affiché un message d'erreur à l'utilisateur.

Il ne semble pas nécessaire de détailler ces comportements via d'autres diagrammes de séquence.



Database

Les tables « role » et « status » ont pour but de définir un nombre limité de choix pour les données qu'elles représentent. Elles sont représentées par des énumérations au niveau du code.

Bien que les valeurs des colonnes « name » de ces tables soit uniques et permettrait d'utiliser ces colonnes comme identifiant, une colonne « id » de valeur numérique a tout de même été ajoutée par cohérence avec le reste des tables de la base de données. Faire passer la cohérence et la simplicité d'utilisation avant la performance n'est pas ici un problème vu que ces tables sont et resteront très légères.

Comme expliqué plus haut, la table « task » dispose d'une référence pointant vers elle-même.
Ceci permet deux choses :

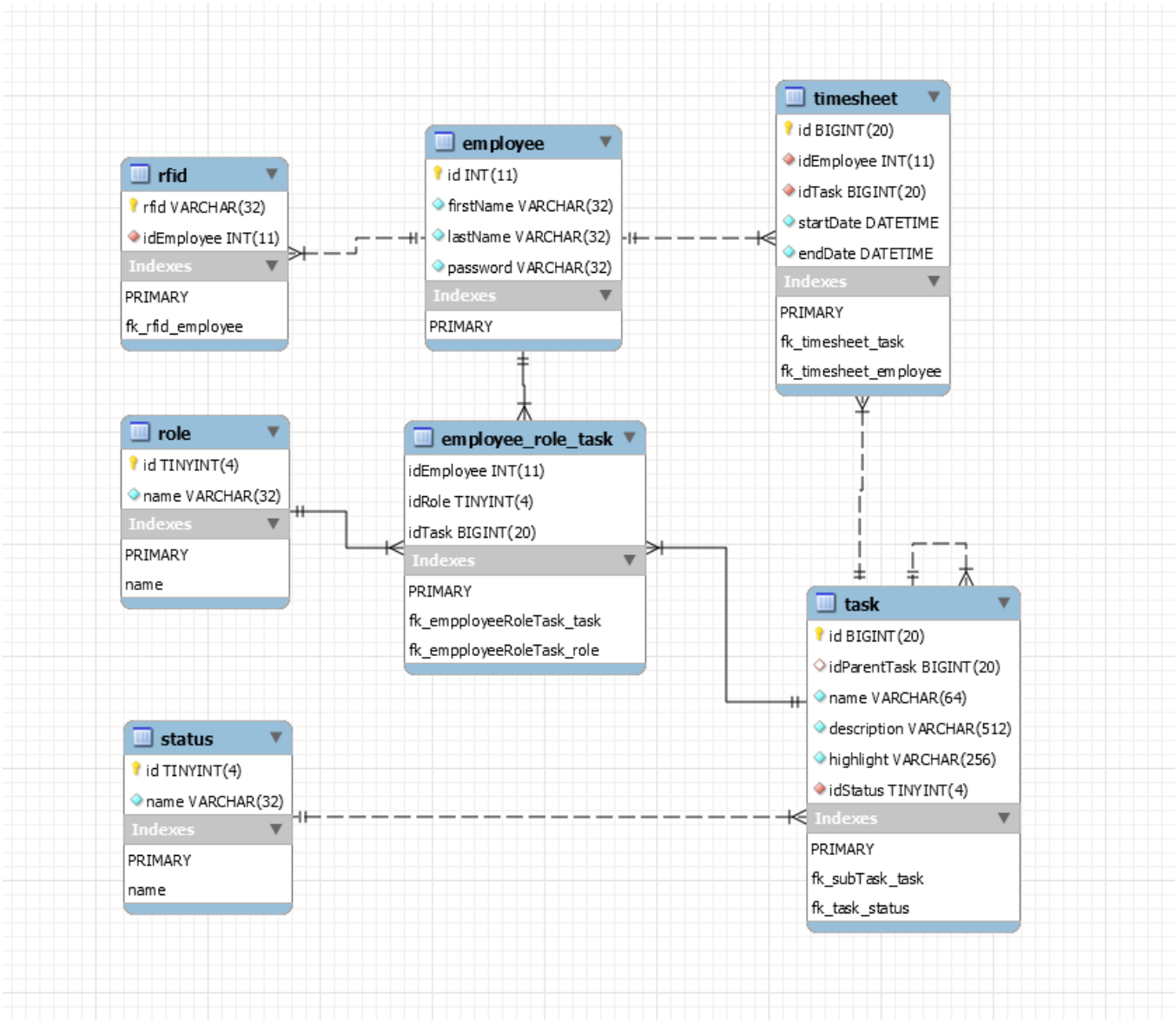
Créer une arborescence de tâches, sous-tâches, etc,… Plus flexible que le système « project »---« activity »

Distinguer une tâche simple d'un projet. Si la référence a un parent est nulle, il s'agit d'un projet.

Une procédure stockée permet de récupérer toutes les heures passées par un employé sur un projet en détaillant les tâches effectuées, le temps passé et la date à laquelle les tâches ont étés effectuées.

À noter que l'utilisation de tâches auto - référencées a demandé l'utilisation de requêtes MySQL récursives. Au niveau du code, l'utilisation d'un fetch type eager a largement facilité la récupération des arbres de tâches mais peut rendre le traitement plus lent que nécessaire. L'augmentation des données transférées par MQTT peut permettre des récupérations plus fines en base de données via hibernate.

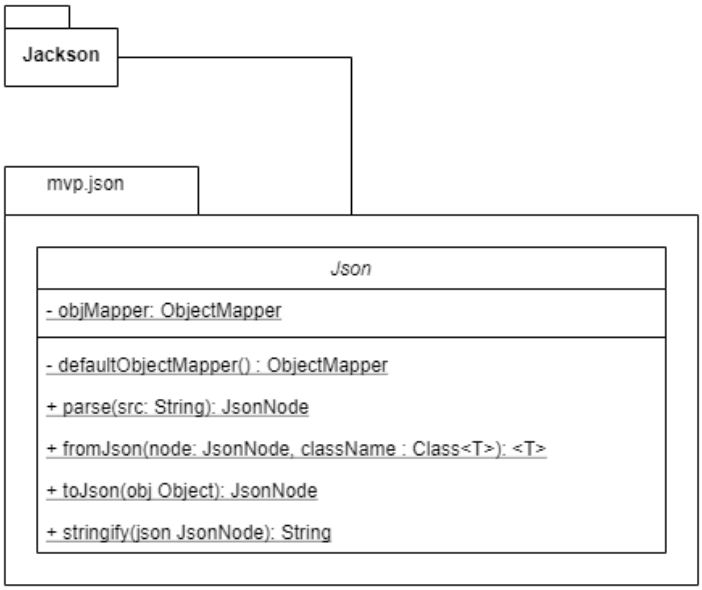
Une version précédente du schéma de base de données est disponible plus bas dans ce document sous la section « Legacy »


rfid
rfid VARCHAR(32)
idEmployee INT(11)
Indexes
PRIMARY
fk_rfid_employee
employee
id INT(11)
firstName VARCHAR(32)
lastName VARCHAR(32)
password VARCHAR(32)
Indexes
PRIMARY
timesheet
id BIGINT(20)
idEmployee INT(11)
idTask BIGINT(20)
startDate DATETIME
endDate DATETIME
Indexes
PRIMARY
fk_timesheet_task
fk_timesheet_employee
role
id TINYINT(4)
name VARCHAR(32)
Indexes
PRIMARY
name
employee_role_task
idEmployee INT(11)
idRole TINYINT(4)
idTask BIGINT(20)
Indexes
PRIMARY
fk_empployeeRoleTask_task
fk_empployeeRoleTask_role
task
id BIGINT(20)
idParentTask BIGINT(20)
name VARCHAR(64)
description VARCHAR(512)
highlight VARCHAR(256)
idStatus TINYINT(4)
Indexes
PRIMARY
fk_subTask_task
fk_task_status
status
id TINYINT(4)
name VARCHAR(32)
Indexes
PRIMARY
name

Class

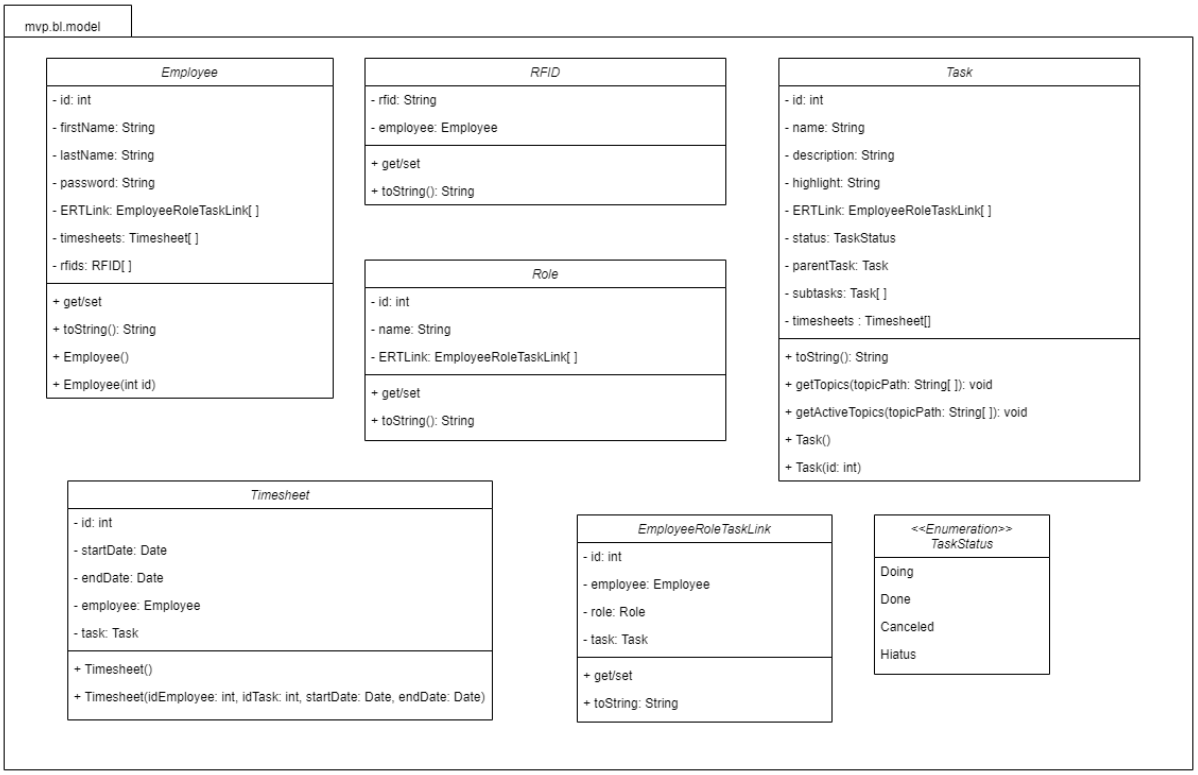
Packages communs aux deux applications

Package Json



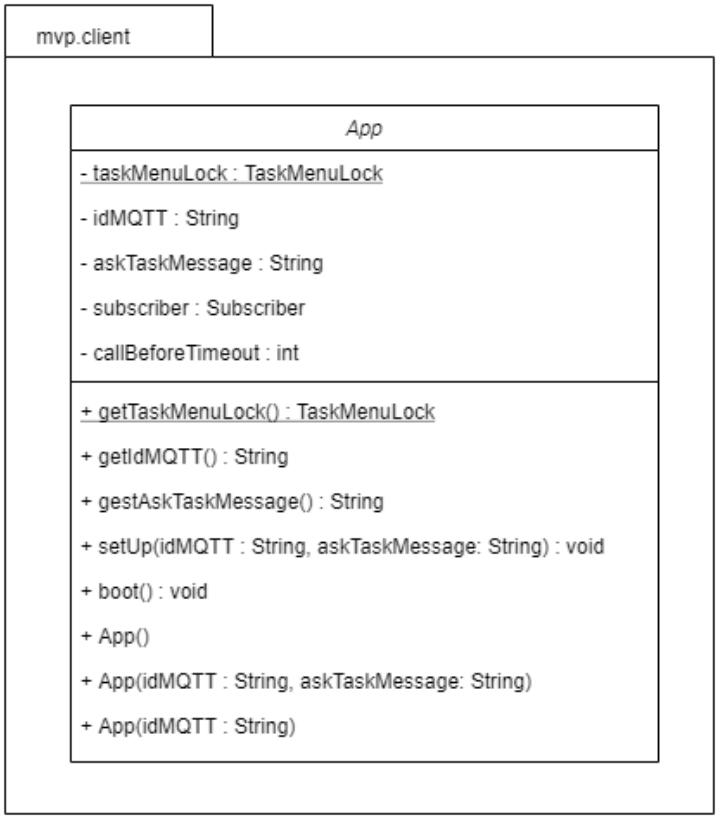
Package model

Bien que techniquement différents, les modèles DAL et BL sont représentés par un seul schéma. Effectivement, la différence provient des annotations Hibernate qui ne sont pas représentées en UML.

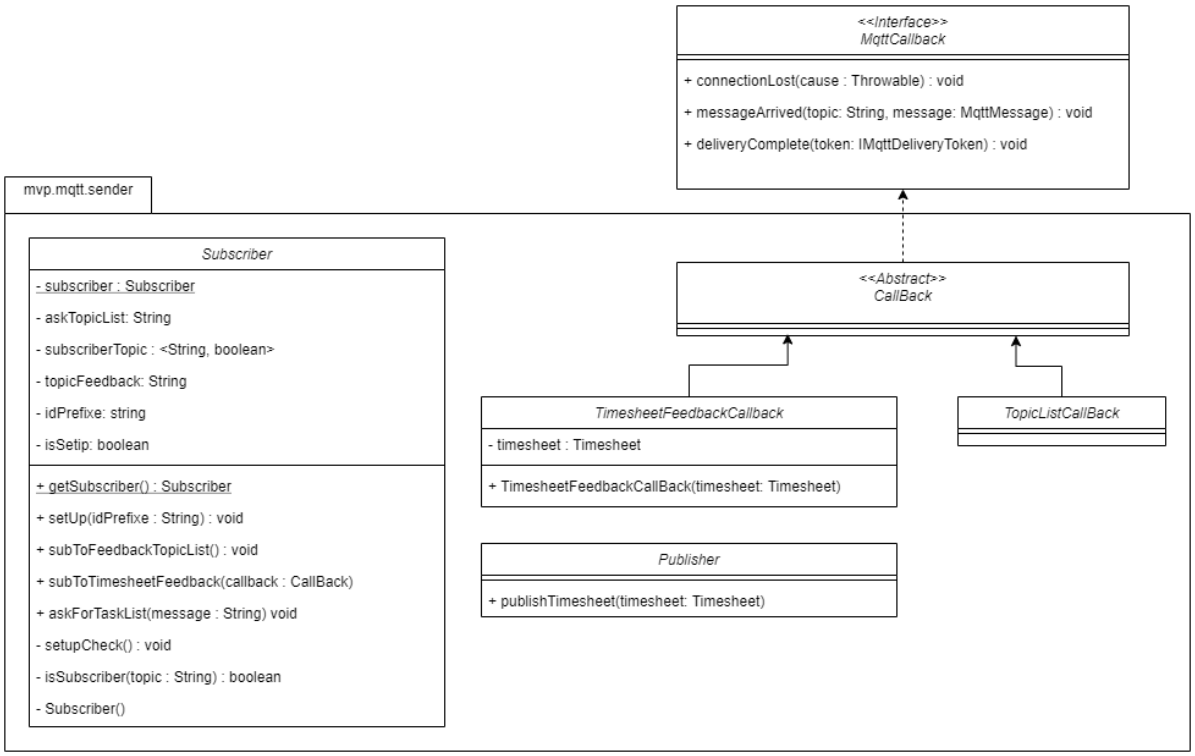


Application client

Package client

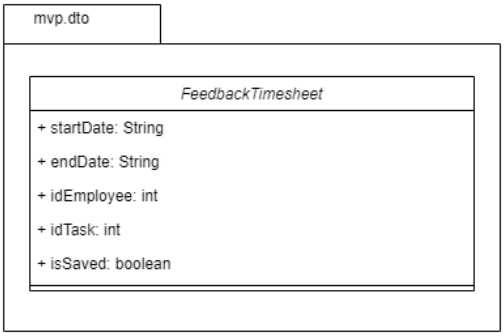


Package mqtt.sender

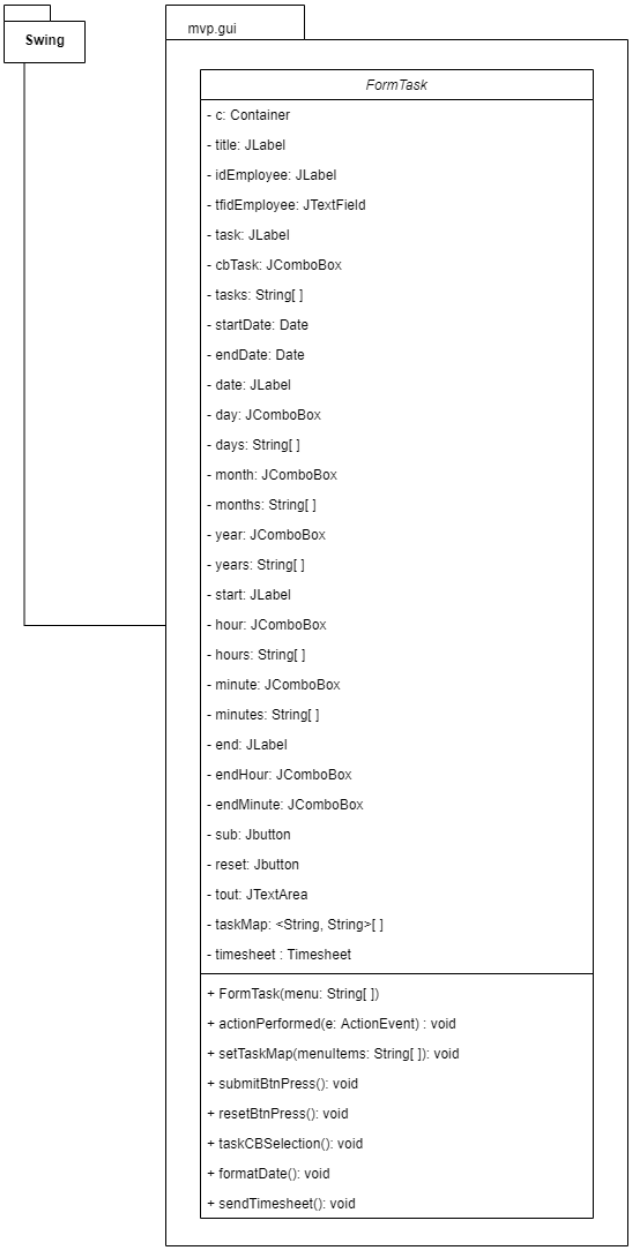


Package dto (Data Transfert Objet)

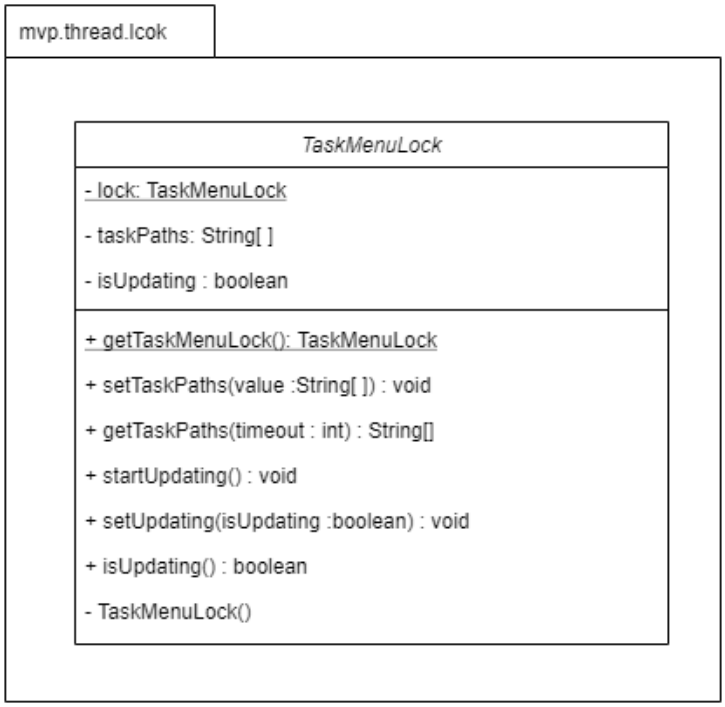
Les objets du package dto sont des POJO dont le seul but est de définir des conteneurs de données. Ceci explique l'absence de méthodes dans les objets de ce package.



Package gui

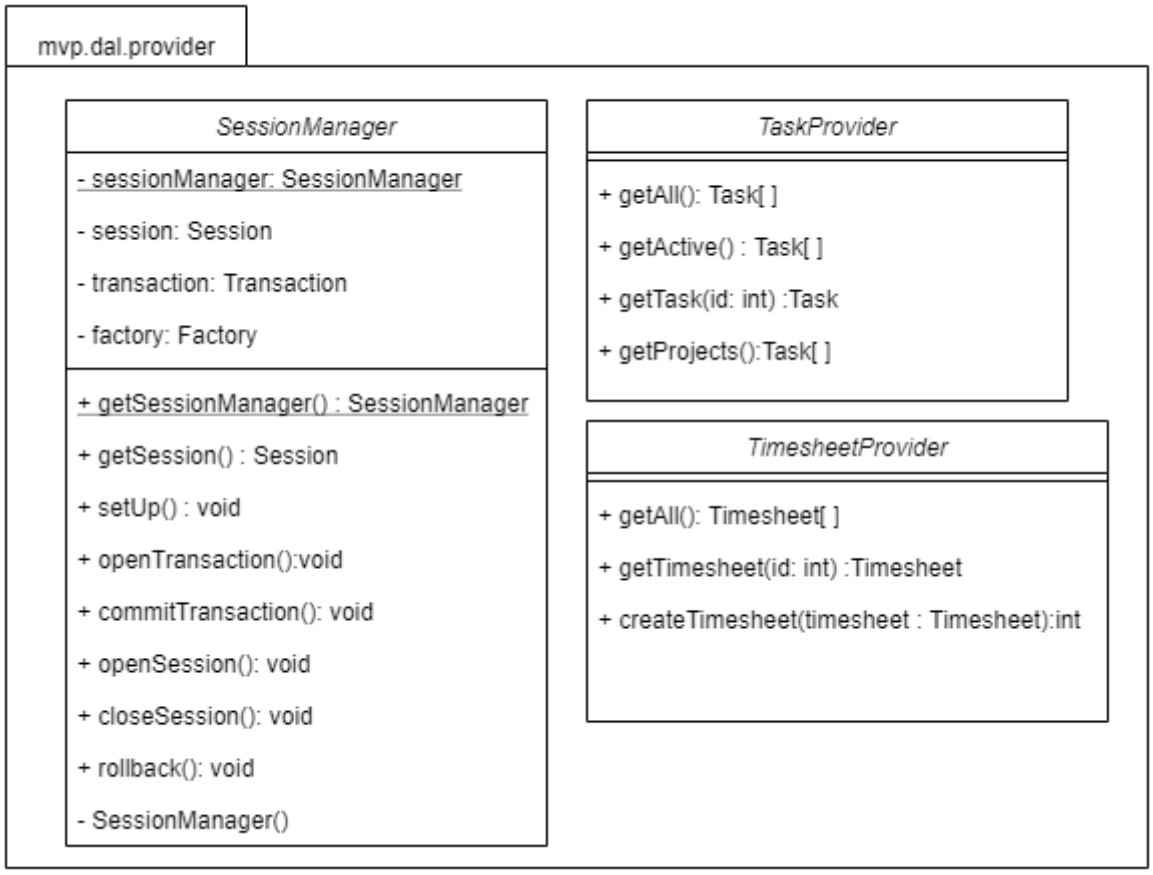


Package thread.lock

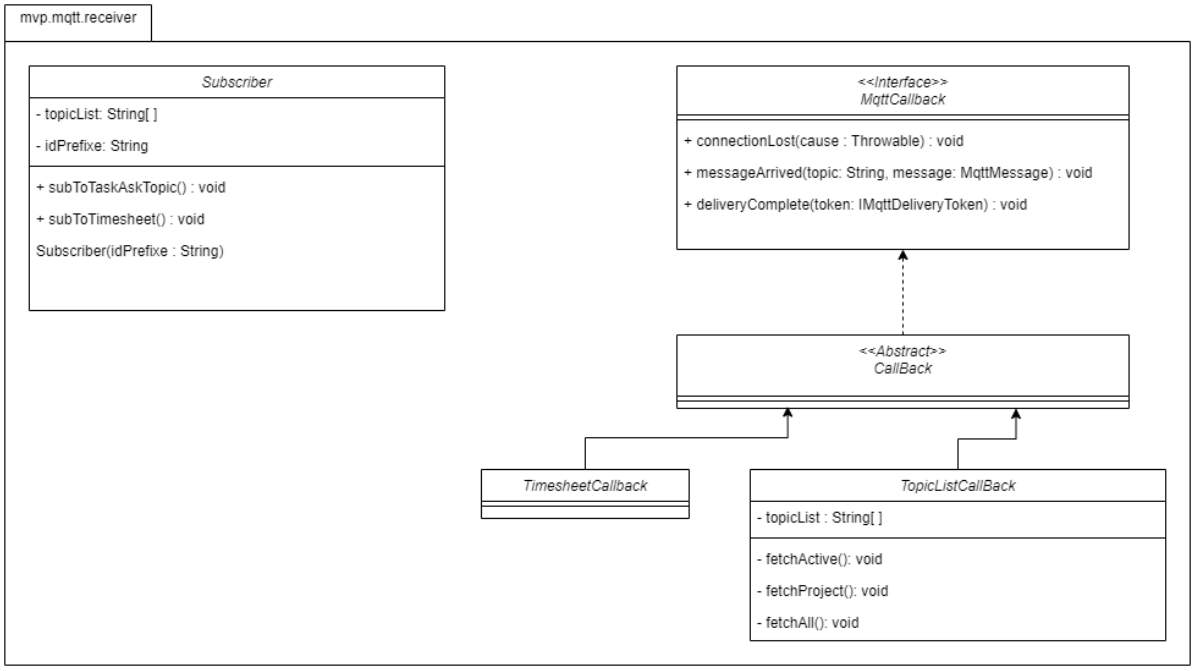


Application serveur

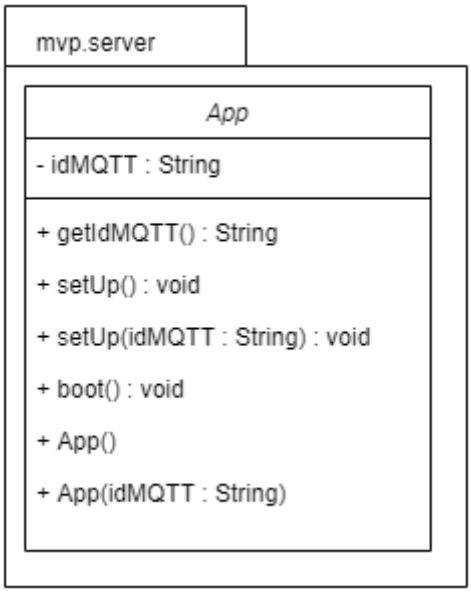
Package dal.provider



Package mqtt.receiver



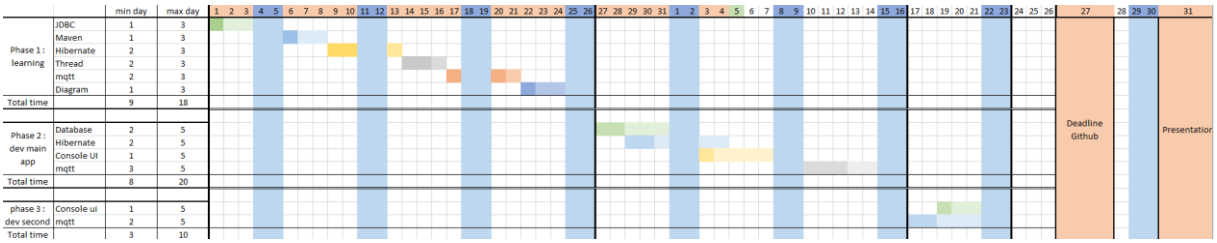
Package server



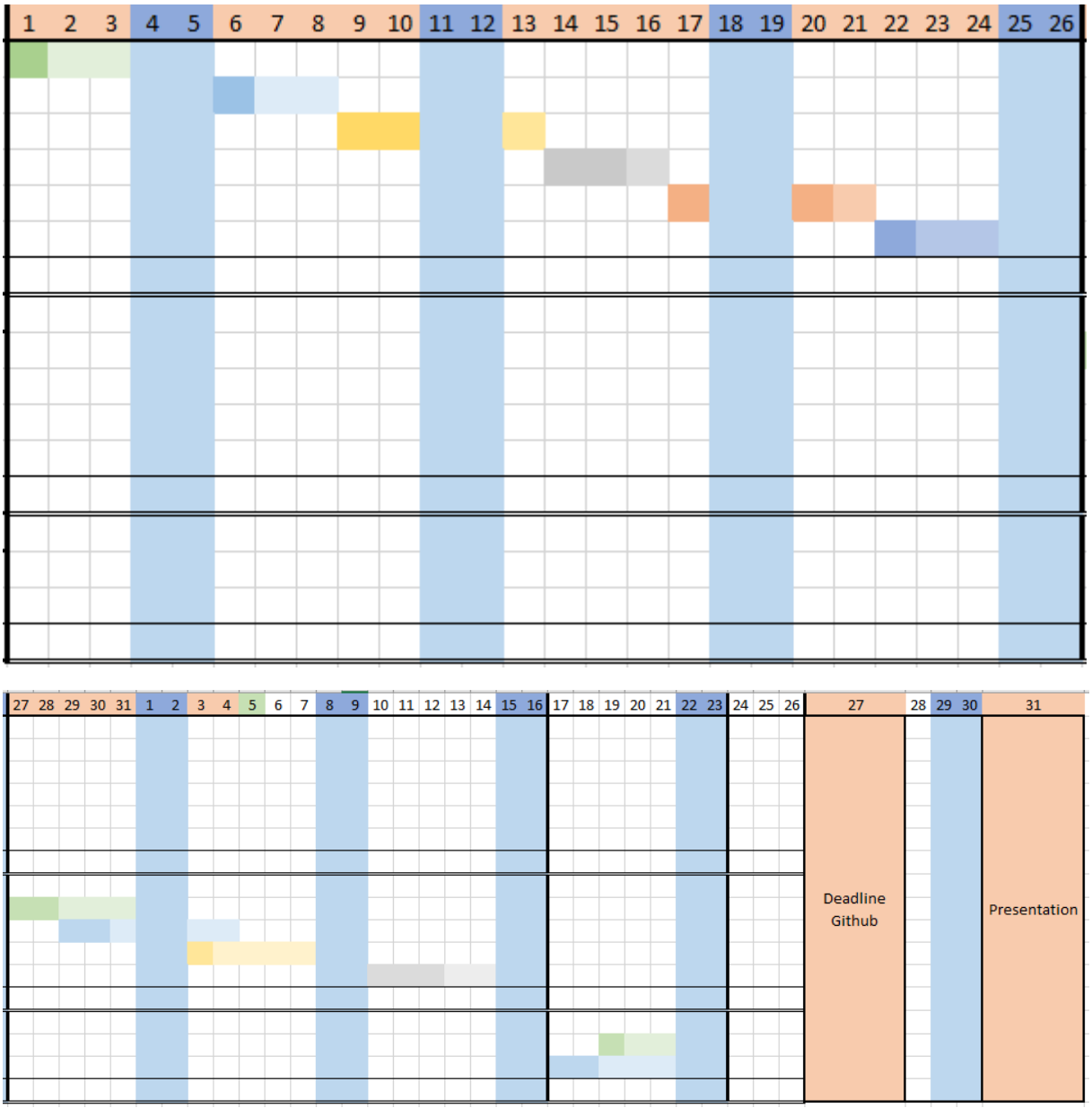
Gantt

Le diagramme de gantt ci-dessous reprend non seulement la période de développement prévue, mais aussi la période de révision.

Pour une raison de lisibilité, le diagramme sera présenté sous son entièreté d'abord, et en 3 parties par après.



		min day	max day
Phase 1 : learning	JDBC	1	3
	Maven	1	3
	Hibernate	2	3
	Thread	2	3
	mqtt	2	3
	Diagram	1	3
Total time		9	18
Phase 2 : dev main app	Database	2	5
	Hibernate	2	5
	Console UI	1	5
	mqtt	3	5
Total time		8	20
phase 3 : dev second	Console ui	1	5
	mqtt	2	5
Total time		3	10



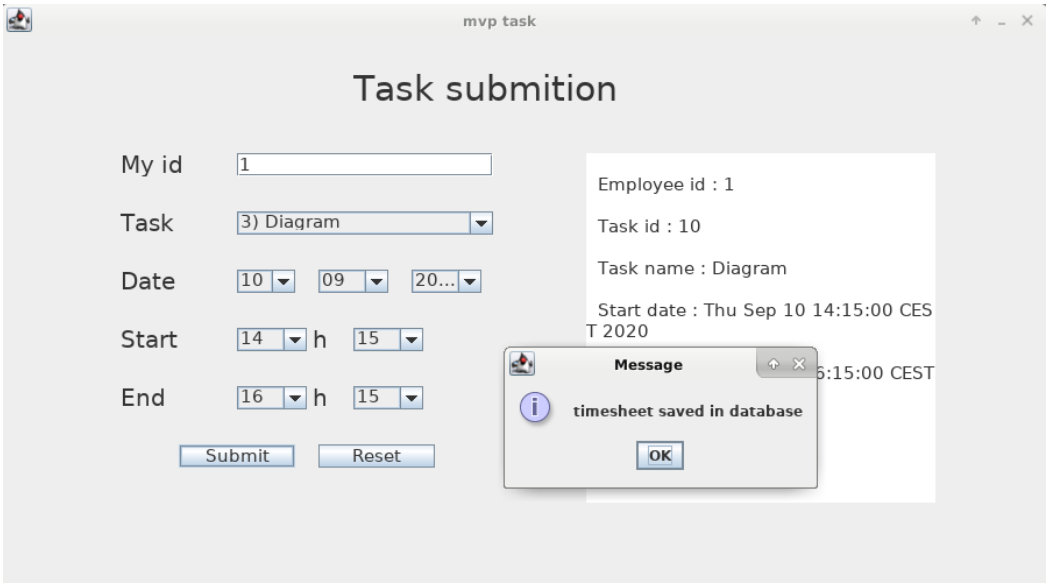
Code

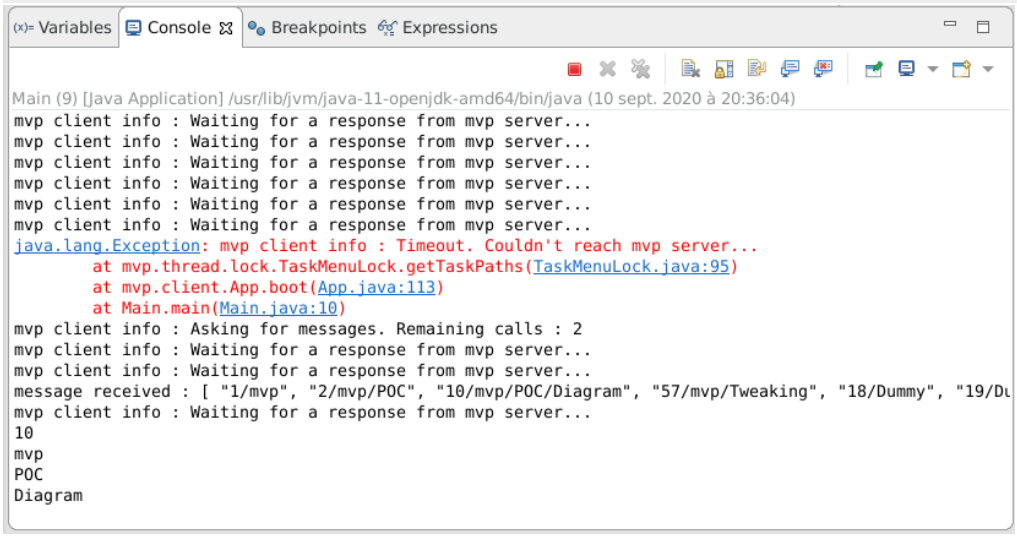
Le code de l'application étant disponible sur Github, il a semblé non cohérent d'alourdir le rapport en copiant l'entièreté du code ci-dessous (surtout si l'impression de ce rapport est requise pour l'archivage)

La demande du professeur P.Vanderheyden étant d'inclure un « fragment de code » et la notion de fragment étant vague, un document sera joint à postériori et après consultation avec le professeur.

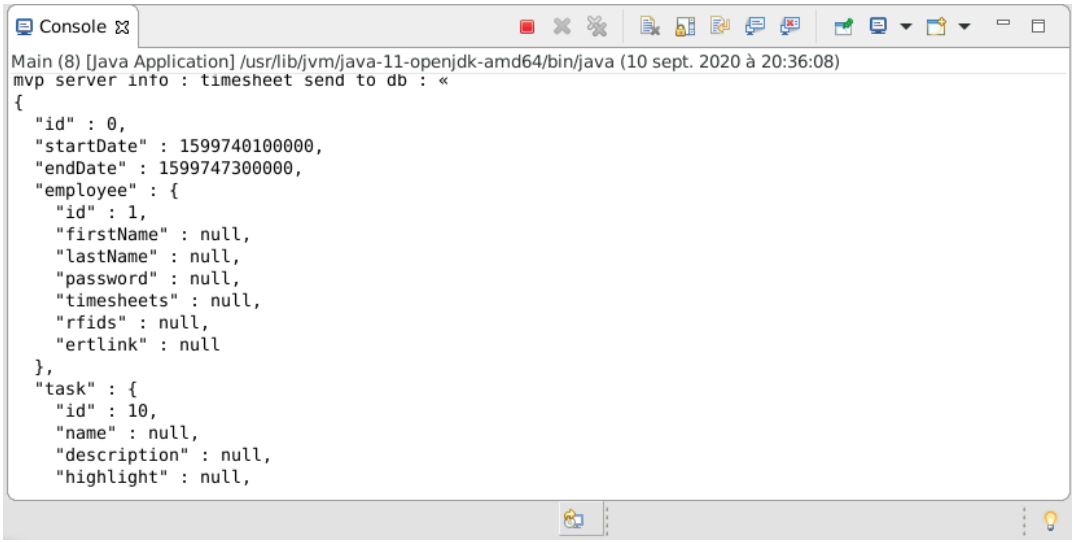
Résultat de la classe test

Application client :

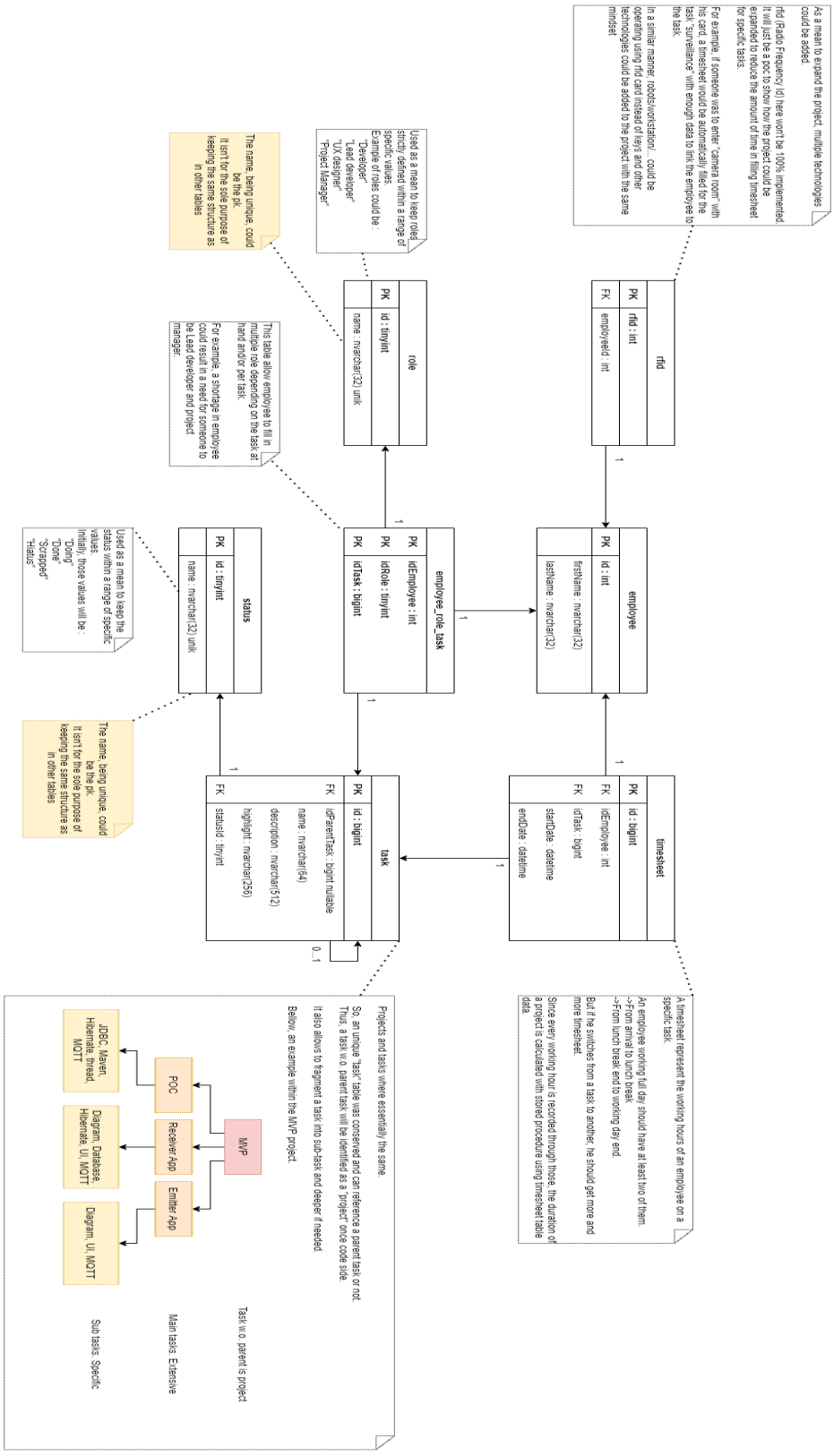




Application serveur



Legacy


As a mean to expand the project, multiple technologies could be added.
rfid (Radio Frequency id) here won't be 100% implemented it will just be a poc to show how the project could be expanded to reduce the amount of time in filling timesheet for specific tasks.
For example, if someone was to enter "camera room" with his card, a timesheet would be automatically filled for the task "Surveillance" with enough data to link the employee to the task.
In a similar manner, robots/workstation/... could be operating using rfid card instead of keys and other technologies could be added to the project with the same mindset
rfid
PK
FK
rfid : int
employeeId: int
1
Used as a mean to keep roles strictly defined within a range of specific values.
Example of roles could be :
"Developer"
"Lead developer"
"UX designer"
"Project Manager"
role
PK
id : tinyint
name : nvarchar(32) unik
The name, being unique, could be the pk.
It isn't for the sole purpose of keeping the same structure as in other tables
This table allow employee to fill in multiple role depending on the task at hand and/or per task.
For example, a shortage in employee could result in a need for someone to be Lead developer and project manager.
employee
PK
id : int
firstName : nvarchar(32)
lastName : nvarchar(32)
employee_role_task
PK
PK
PK
idEmployee : int
idRole : tinyint
idTask : bigint
1
1
1
status
PK
id : tinyint
name :nvarchar(32) unik
Used as a mean to keep the status within a range of specific values.
Initially, those values will be :
"Doing"
"Done"
"Scrapped"
"Hiatus"
The name, being unique, could be the pk.
It isn't for the sole purpose of keeping the same structure as in other tables
1
task
PK
FK
FK
id : bigint
idParentTask : bigint nullable
name : nvarchar(64)
description : nvarchar(512)
highlight : nvarchar(256)
statusId :tinyint
0..1
timesheet
PK
FK
FK
id : bigint
idEmployee : int
idTask : bigint
startDate : datetime
endDate : datetime
1
A timesheet represent the working hours of an employee on a specific task.
An employee working full day should have at least two of them.
->From arrival to lunch break
->From lunch break end to working day end.
But if he switches from a task to another, he should get more and more timesheet.
Since every working hour is recorded through those, the duration of a project is calculated with stored procedure using timesheet table data
Projects and tasks where essentially the same
So, an unique "task" table was conserved and can reference a parent task or not.
Thus, a task w.o. parent task will be identified as a "project" once code side.
It also allows to fragment a task into sub-task and deeper if needed.
Bellow, an example within the MVP project.
MVP
POC
Receiver App
Emitter App
JDBC, Maven, Hibernate, thread, MQTT
Diagram, Database, Hibernate, UI, MQTT
Diagram, UI, MQTT
Task w.o. parent is project
Main tasks: Extensive
Sub tasks: Specific