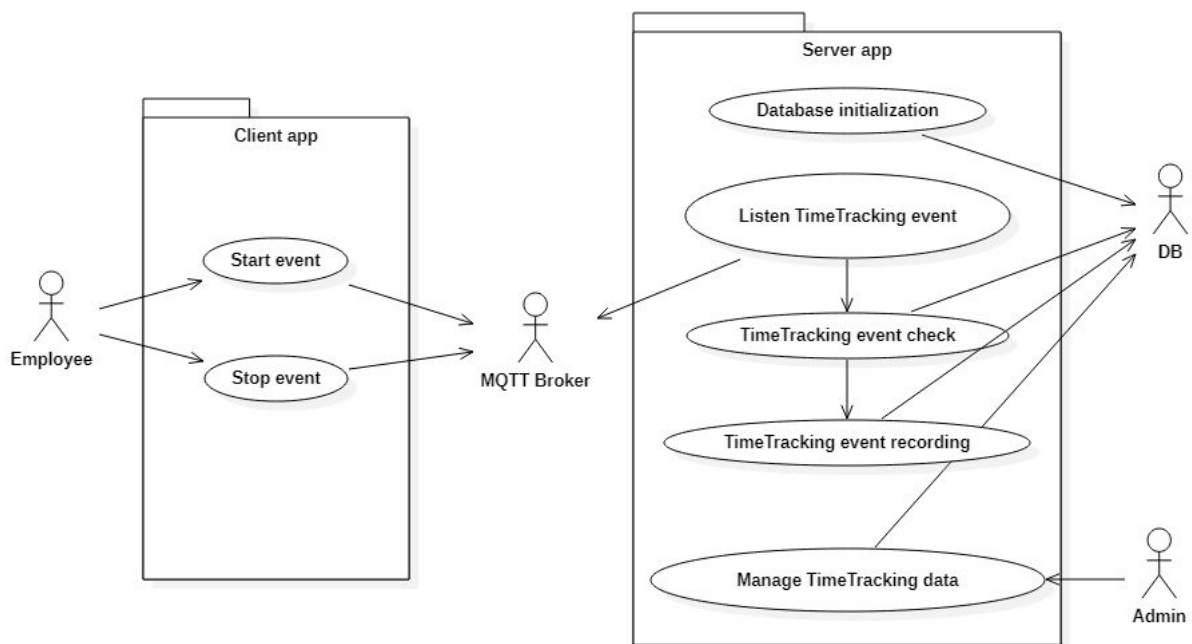


Time Report

Technologie utilisé:

Java, Hibernate, JDBC, Maven & MQTT

Use Case:



Application client:

L'employé peut lancer deux types d'événements: START ou STOP.

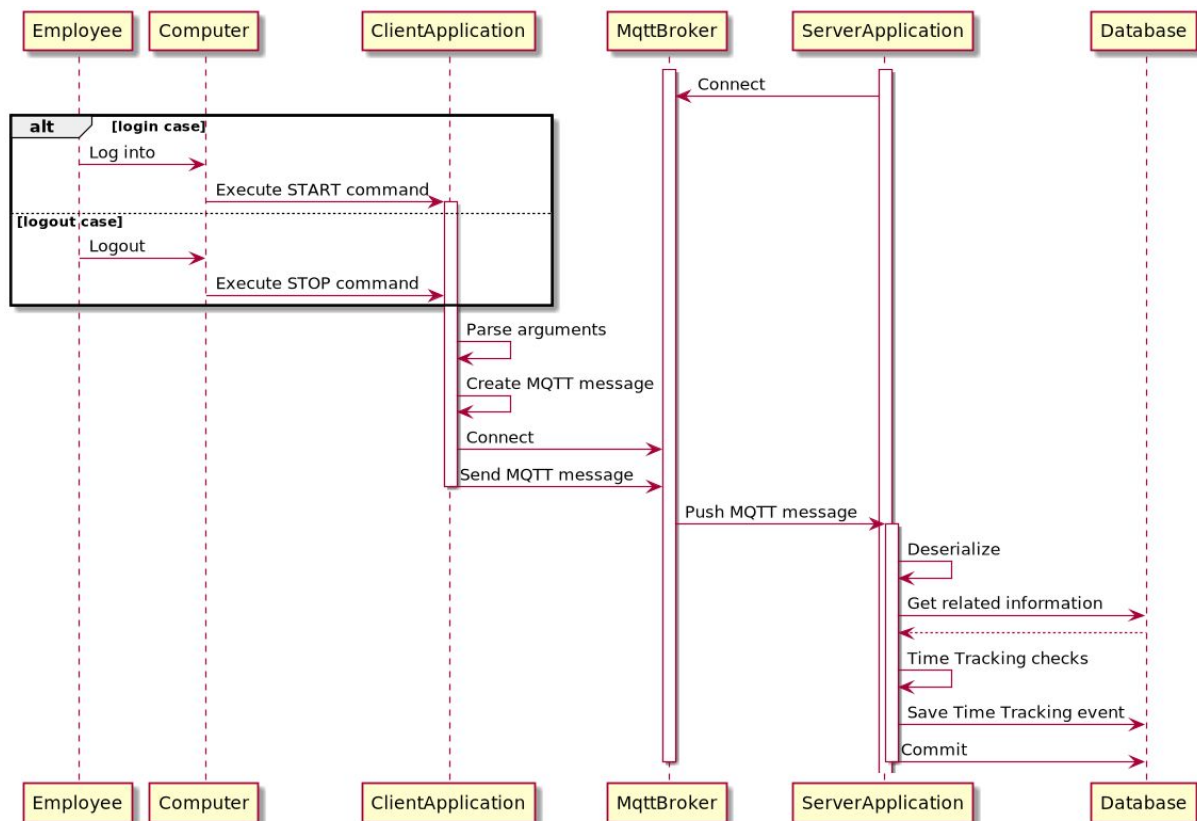
Dans ces deux cas l'application communique l'événement au Broker avec l'Id de l'employé ainsi que celui de la tâche.

Application server:

L'administrateur peut insérer des données dans la DB telle que des employés, des projets ou des tâches.

Un second thread permet d'écouter le Broker et lorsqu'un event lui parvient elle vérifie si les informations sont cohérentes avant de les insérer dans la base de données.

Diagramme de séquence:



Lorsque l'employé commence à travailler, l'application client devra être lancée avec 3 arguments: son ID suivis de START et enfin l'ID de la tâche sur laquelle il travaille. Même procédé lorsque l'employé arrête de travailler mais le deuxième argument sera STOP.

L'application cliente crée un message MQTT avec les informations qui lui sont fournies en argument. Elle se connecte ensuite au broker MQTT afin d'envoyer son message.

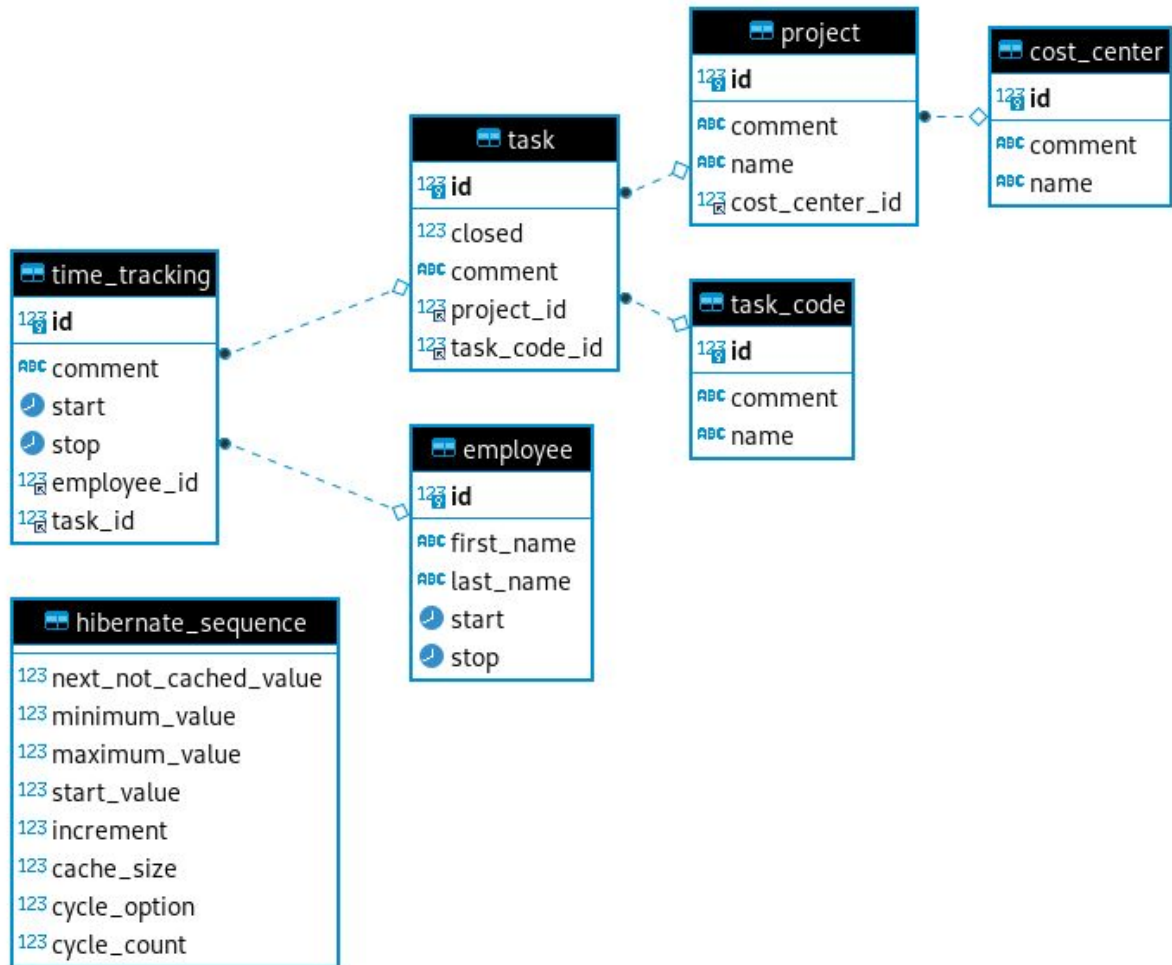
L'application serveur qui s'est connectée préalablement au broker MQTT dispose d'un thread qui écoute le broker. Lors de la réception d'un message, celui-ci va être désérialisé afin d'en exploiter le contenu. Ces informations vont être contrôlées avec les informations contenues dans la DB. Dans le cas des anomalies majeures :

1. Non existence de l'employé
2. Non existence de la tâche
3. Tâche non préalablement démarrée

L'application affichera un message d'erreur et les informations ne seront pas ajoutées à la base de données.

Dans le cas d'une anomalie mineure (démarrage d'une nouvelle tâche alors que l'employé a toujours des tâches en cours), l'information sera ajoutée à la base de données et un warning apparaîtra sur le serveur.

Schéma de base de données:



Chaque projet fait référence à un cost center dans lequel on pourrait implémenter les adresses de facturation.

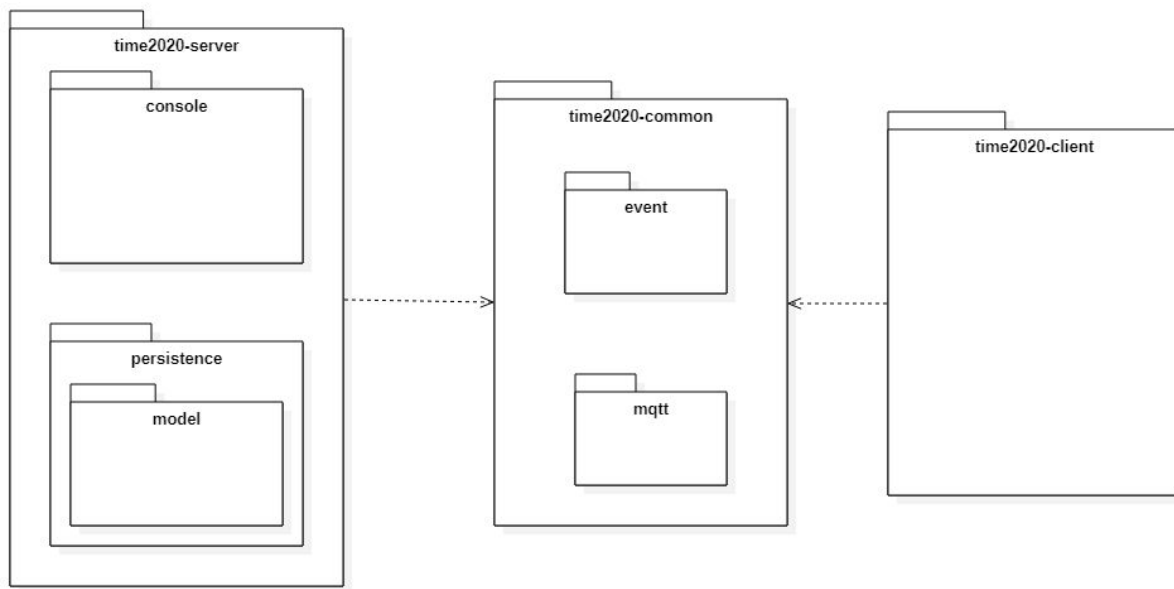
Les projets sont divisés en plusieurs tasks. On suppose que la nature de ces tâches sont récurrentes ce qui justifie la présence de la table task_code afin d'optimiser la taille de la base de données.

La Table time_tracking contient:

- Les périodes travaillées par les employés (stop - start).
- L'ID de l'employé concerné.
- L'ID de la tâche concernée.

Dans le cas où l'employé travaille toujours dans l'entreprise, le champ stop de la table employee sera null. Même logique pour la table time_tracking si l'employé est en train de travailler sur une tâche.

Diagramme de Packages



Le projet a été structuré comme suit:

- Un package time2020-server contenant les éléments propre au serveur.
- Un package time2020-client contenant les éléments propre au client.
- Un package time2020-common contenant les éléments commun aux tiers serveur et client

Dans le package time2020-server on trouve deux autres packages:

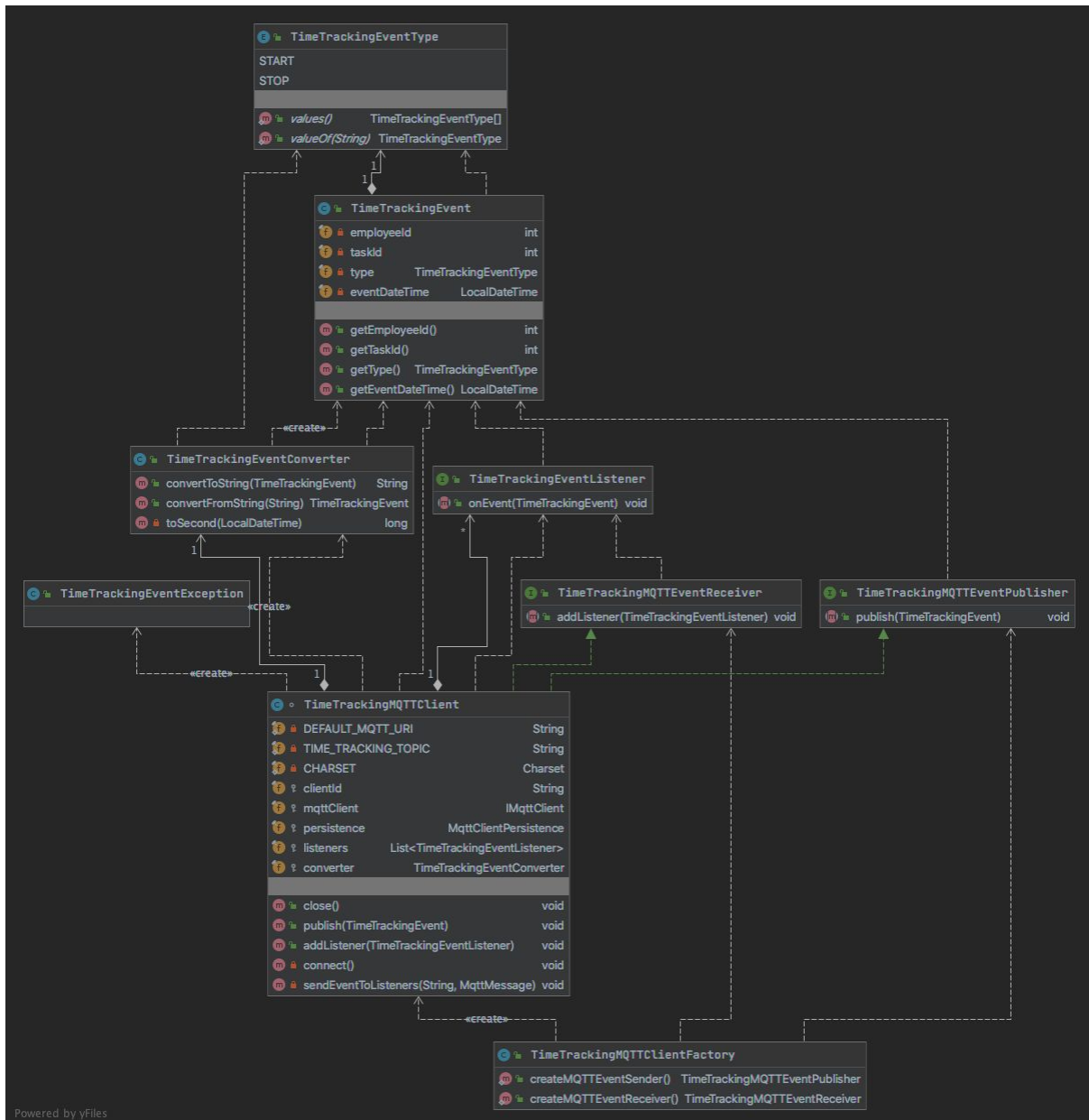
- console; contenant le "front" de l'application server.
- persistence; contenant les objets liés à la base de données (aux données persistantes)

Le package model contient les entités Hibernate permettant le mapping avec la base de données.

Le package time2020-common se divise en deux sous packages:

- event; qui contient les éléments propres au message partagé entre le serveur et le client.
- mqtt; gère l'intégration MQTT grâce à la librairie Eclipse Paho pour communiquer avec le broker MQTT.

time2020-common



Package mqtt:

TimeTrackingMQTTClient permet de faire la liaison avec le broker MQTT et implémente l'interface TimeTrackingMQTTEventPublisher pour le client ainsi que TimeTrackingMQTTEventReceiver pour le serveur.
(TimeTrackingMQTTClient a une visibilité de type "package private")

TimeTrackingMQTTClientFactory est une classe publique qui permet au tiers serveur et client d'instancier un objet TimeTrackingMQTTClient sans dépendre directement de ce dernier. Les interfaces Receiver et Publisher permettent de cacher la dépendance avec

TimeTrackingMQTTClient et donc de créer un couplage faible avec ce dernier (il pourra être remplacé sans impact dans le reste de l'application).

Package event:

TimeTrackingEventType est une énumération permettant de lister les types d'événements gérés. (Actuellement "START" et "STOP".)

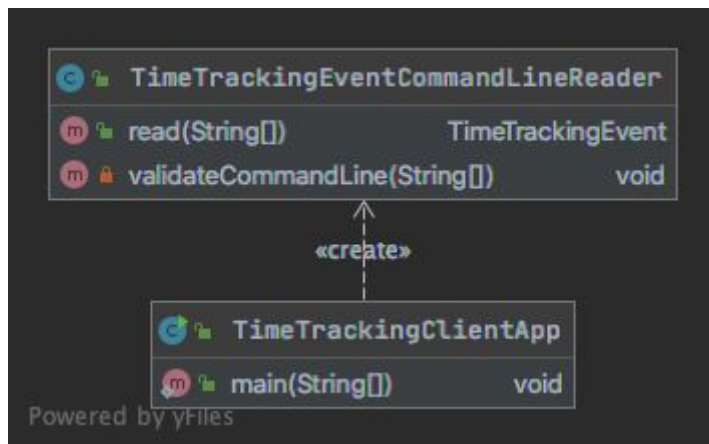
TimeTrackingEventException est une exception propre au projet qui est lancée lorsqu'un paramètre de l'événement n'est pas correct.

TimeTrackingEventConverter implémente la méthode utilisée pour convertir l'objet TimeTrackingEvent en chaîne de caractère qui sera incluse dans le message MQTT ainsi que la méthode inverse.

TimeTrackingEvent correspond à l'événement qui est transmis.

TimeTrackingEventListener est l'interface définissant la méthode "onEvent" qui doit être implémentée par le serveur pour gérer l'événement en s'enregistrant auprès du TimeTrackingMQTTEventReceiver.

time2020-client



TimeTrackingClientApp contient le “main” de l’application client.

TimeTrackingEventCommandLineReader définit et implémente les méthodes permettant de contrôler les arguments fournis à l’application cliente et de convertir ces arguments en un objet TimeTrackingEvent valide.

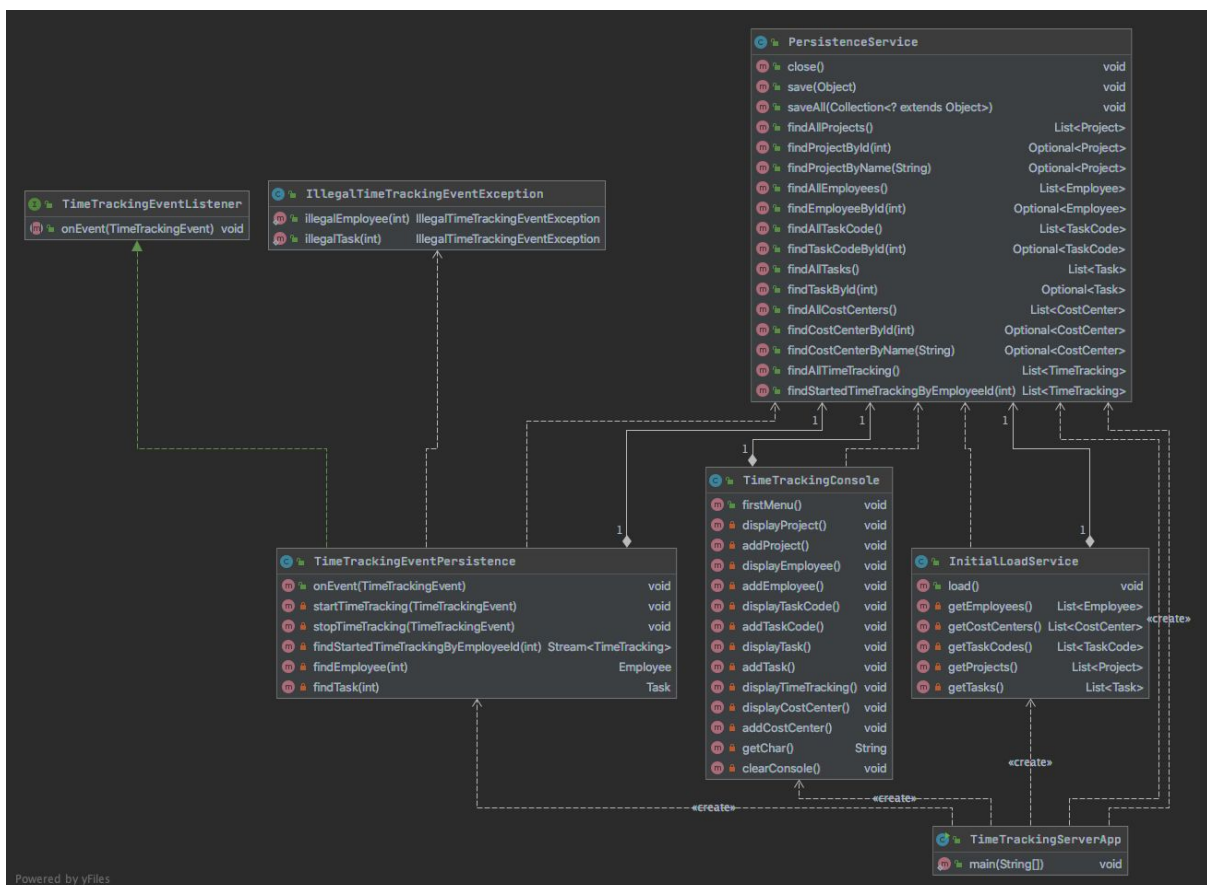
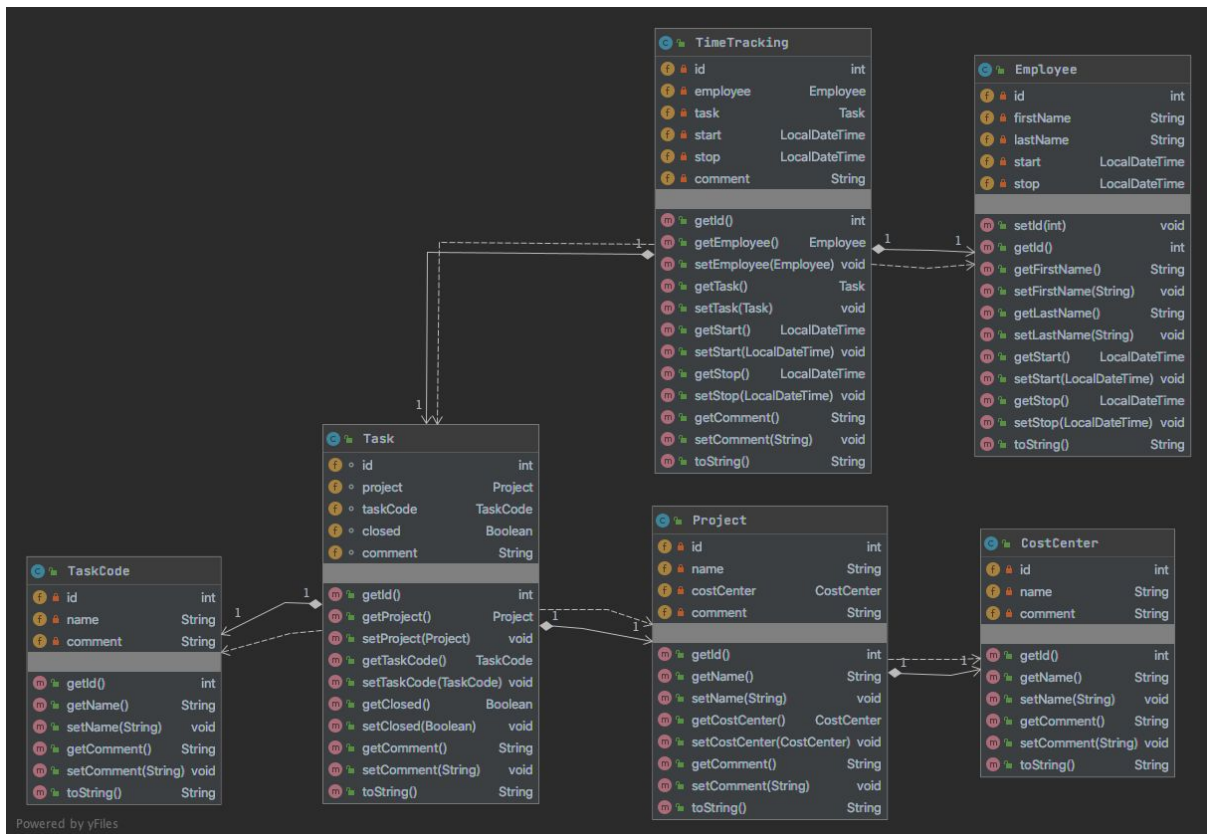
Exemple d’une commande START:

```
java -jar time2020-client-1.0-SNAPSHOT-jar-with-dependencies.jar 1 START 13
```

Exemple d’une commande STOP:

```
java -jar time2020-client-1.0-SNAPSHOT-jar-with-dependencies.jar 1 STOP 13
```

time2020-server



TimeTrackingServerApp contient le “main” de l’application serveur.

TimeTrackingConsole contient les éléments propres à l’interface de l’application serveur.

IllegalTimeTrackingEventException est une exception propre au tiers serveur qui sera lancé lorsqu’un événement fera référence à un employé ou une tâche existante.

PersistenceService permet de réaliser la liaison avec la base de données.

TimeTrackingEventPersistence est la classe qui implémente l’interface TimeTrackingEventListener. Elle implémente donc la méthode “onEvent” qui gère l’arrivée des événements.

InitialLoadService est une classe utilisée pour insérer des éléments dans la DB afin de faciliter les tests. En production cette classe ne serait pas utilisée. Actuellement hibernate est configuré pour recréer la DB à chaque lancement afin d’avoir une DB qui n’est pas polluée par les président test.

Actuellement: <property name="hbm2ddl.auto">create</property>

En production: <property name="hbm2ddl.auto">update</property>

Lancement du serveur:

```
java -jar time2020-server-1.0-SNAPSHOT-jar-with-dependencies.jar
```