Structure détaillées

1. **Classe Abstraite : `M5lib`**

Classe de base commune à tous les types de M5. Elle définit les fonctionnalités fondamentales.

Fonctions membres

Fonction	Rôle
`setupstd()`	Effectue l'initialisation matérielle commune : écran, Wi-Fi, RFID, UHF. Attention : la fonction est lboquante tant que le M5 n'est pas connecté au réseau prédéfinit.
`reducejson(String json)`	Réduit et structure le JSON de réponse de l'API TimeTonic pour simplifier l'accès aux champs utiles. Ne garde que le contenu de "fields" (endroit où est stocké entre autres les données demandées) et "rowInfos" (qui contient notamment le rowId des informations demandées)
`scancard()`	Scanne une carte RFID classique, retourne l'UID du tag sous forme de String. Contient une UI intégrée. Attention : la fonction est bloquante tant qu'elle n'a pas lu une carte.
`scanuhf(String* urfid)`	Scanne les tags RFID UHF via le module UHF externe, remplit le tableau passé en paramètre avec les EPC lues (max 200 EPC, ce qui est le max supporté par le module RFID UHF). Fonction actuellement inutilisée dans le code.
`showchoice(String choices[])`	Affiche sur l'écran un menu de sélection contenant les String du tableau passé en paramètre : l'utilisateur peut naviguer puis valider un choix(jusqu'à 9 options), qui est retourné sous forme de String.
`virtual int uploadlog(String card, String action, String other) = 0`	Méthode virtuelle pure pour uploader un log dans la table correspondante au type de M5. Prend en paramètre une carte lue, l'action effectuée par l'utilisateur et un agument optionnel (au choix, le motif de la visite, l'ordinateur ou l'outil emprunté).
`virtual int changestatus() = 0`	Méthode virtuelle pure pour changer le statut d'emprunt/de présence (obsolète pour le moment car remplacé par automatisation TimeTonic).
`getuser(String* user)`	Lance un scan RFID (avec scancard()), puis interroge l'API TimeTonic pour récupérer le nom, prénom, et rowId de l'utilisateur scanné. Peuple ensuite le tableau passé en paramètre : [nom, prénom, tag RFID, rowId].

Fonction	Rôle
`borrow()`	Gère tout le cycle d’emprunt et retour d’un matériel/machine/ordinateur : identification (avec un appel getuser()), log (avec un appel uploadlog()), et contrôle utilisateur (gestion par l'utilisateur de la durée de son emprunt, avec à nouveau un scan de carte et un upload de log à la fin de l'emprunt. Boucles if/else à chaque étape du cycle pour gérer chaque type d'erreur (API, utilisateur inconnu, etc...).

Attributs

- `user[4]` : tableau de String contenant toutes les informations d'un utilisateur : [nom, prénom, tag RFID, rowId].
- `urfid[200]` : tableau de String pouvant contenir jusqu'à 200 EPC.
- `sesskey` : permet de modifier simplement et directement la clé d'API pour toutes les requêtes de la bibliothèques (en cas d'expiration par exemple)

2. **Classes Dérivées**

Chaque classe dérivée apporte des fonctionnalités propres à chaque typologie de M5 dans le FabLab.

a. **Classe : `accueil`**

Ajoute des méthodes spécifiques à la gestion d’accueil.

Fonction	Rôle
`int uploadlog(...) override`	Upload des logs d’accueil à l’API avec les bons id (table, champs...).
`int changestatus() override`	Change le statut d’accueil (placeholder : actuellement valeur fixe pour compatibilité).
`void entree()`	Gère la procédure d’entrée : identification (avec getuser), choix du motif de visite (avec showchoice), upload du log d’arrivée (avec uploadlog)
`void sortie()`	Gère la sortie d’un utilisateur (avec getuser) et log cet évènement (avec uploadlog).
`void regcard()`	Enregistre une nouvelle carte RFID (avec un scancard puis une requete d'API spécifique) pour un utilisateur après inscription sur Timetonic (liaison rangée TimeTonic <-> badge RFID). La requête d'API récupère la dernière rangée créée avec une vue filtrée (vue chronologique dans TimeTonic), il faut donc faire l'association de carte immédiatement après l'inscription.

Attributs spécifiques

`String motifs[9]`: Liste des motifs de visite disponibles à la sélection (préremplie pour plus de simplicité).`String name` : nom du M5, permet de définir l'action d'accueil effectuée : entrée, sortie ou 1ère inscription.

b. **Classe : `servante`**

Pour la traçabilité de l’usage des servantes.

Fonction	Rôle
`int uploadlog(...) override`	Upload des logs liés à l’utilisation d’une servante (carte, action, ""). L'argument other ne sert pas pour les servantes.
`int changestatus() override`	Change le statut de la servante (obsolète, pour compatibilité).

Attributs spécifiques

- `String rowid`: id de la rangée de cette servante dans la table Servantes du book Inventaire dans TimeTonic (unique à chaque servante, et donc chaque appareil, voir table d'équivalence dans la doc privée).
- `String name` : nom du M5, permet de définir la servante concernée dans les logs.

c. **Classe : `machine`**

Gère l’utilisation des machines/outils fixes.

Fonction	Rôle
`int uploadlog(...) override`	Upload logs liés à l’utilisation d’une machine (carte utilisateur, action).

Attributs spécifiques

- `String rowid`: id de la rangée de cette machine dans la table Machines du book Inventaire dans TimeTonic (unique à chaque machinr, et donc chaque appareil, voir table d'équivalence dans la doc privée).
- `String name` : nom du M5, permet de définir la machine concernée dans les logs.

d. **Classe : `ordinateur`**

Gère le suivi de l'emprunt des ordinateurs portables.

Fonction	Rôle
`int uploadlog(...) override`	Upload logs spécifiques à l'usage d'un ordinateur (par exemple : session ouverte/fermée).

Attributs spécifiques

- `WIP`

e. **Classe : `materiel`**

Gère les cycles d'emprunt et de retour d'outillage ou matériel divers.

Fonction	Rôle
`int uploadlog(...) override`	Upload logs liés à l'utilisation (emprunt/retour) de matériels.

Attributs spécifiques

- `WIP`

3. **Utilisation de méthodes virtual / override**

Les méthodes `uploadlog` et `changestatus` sont définies comme `virtual` dans la base, et nécessaires à la personnalisation du comportement selon chaque situation FabLab : chaque classe dérivée implémente les appels API avec les bons identifiants et formats de données.

Avec cette structure, chaque fonction de **M5lib** remplit un rôle précis dans la gestion, la traçabilité, l'ajout de dispositifs et l'intégration back-end dans l'écosystème du FabLab.

Revision #2
Created 21 July 2025 13:05:32 by Eyglier De Rooster Mathieu
Updated 22 July 2025 14:18:24 by Eyglier De Rooster Mathieu