24 juin 2022

Données pour la maintenance du projet MIMI

Ce document à pour but d'aider à maintenir et développer le projet MIMI. Ce document est axé sur le développement back-end et sur la gestion de la base de données.

Node js

Lancement du serveur

En local

Installer node si ce n'est pas déjà fait : https://nodejs.org/fr/download/
Pour vérifier que node soit installé, entrez la commande ci-dessous, si c'est le cas la version s'affichera.

```
node -v
```

Se positionner dans le répertoire "backend" depuis une invite de commande pour démarrer le serveur (en mode production):

```
npm start
```

ou

```
node index.js
```

Pour démarrer le serveur en mode développeur (le serveur se relance à chaque changement de code) :

```
nodemon index.js
```

Sur la machine distante Debian

Se connecter sur Linux, entrez dans un invite de commande ceci :

```
ssh mimi@connected-health.fr
```

puis renseigner le mot de passe quand on vous le demande.

Depuis Windows, utiliser un logiciel comme Putty (https://www.putty.org/) ou BitVise pour avoir une interface graphique.

Voir le statut du serveur

```
pm2 status
```

Démarrer le serveur (se positionner dans le répertoire backend)

```
pm2 start index
```

Démarrer le socket de la caméra ou du morpion

```
pm2 start camera ou titctactoe
```

Pour redémarrer le serveur, il faut écrire `restart` à la place de `start.`

Librairies

Packages installés :

- express : pour le routage de l'application
- express-session : pour les sessions/ cookies
- bcrypt : pour le hachage du mot de passe
- cors : pour les requêtes qui n'ont pas la même origine
- dotenv : pour récupérer un fichier .env (pour stocker les variables d'environnement)
- sequelize : ORM (permet de changer de base de données sans avoir à changer de code)
- cookie-parser : pour les cookies
- nodemon : pour démarrer en mode développement
- jsonwebtoken : pour les tokens d'authentification
- mariadb: pour se connecter à mariadb avec sequelize
- fs : pour la manipulation de documents
- @google-cloud/storage : pour les documents enregistrés sur le stockage de google cloud
- multer : pour récupérer les documents téléchargés dans un formulaire côté client
- fast-folder-size : pour récupérer la taille d'un dossier et ainsi déterminer l'espace de stockage restant
- nodemailer : pour l'envoi d'email pour la récupération de mot de passe.
- jsdoc et doc dash : pour la génération de la documentation

installation de package précis :

```
npm install nompackage
```

installation de tous les packages

```
npm install
```

ou juste

```
npm i
```

Pour avoir des informations complémentaires sur un package :
https://www.npmjs.com/

Le fichier .env

Ce fichier, situé à la racine du dossier backend, contient toutes les variables d'environnements. Il est nécessaire de le remplir adéquatement afin d'assurer la pérennité du projet.

Variables pour les tokens de session

Les secrets sont très importants, car ils permettent de vérifier la validité d'une session On les utilise pour créer les tokens, il faut donc qu'ils restent cachés des utilisateurs, il faut aussi les changer fréquemment.

```
ACCESS_TOKEN_SECRET = access_secret
REFRESH_TOKEN_SECRET = refresh_secret

SECRET = secret_session
```

Variables pour la base de données

Il faudra faire les changements nécessaire en cas de modification de la base de données :

```
# L'adresse de la base de données auquelle on va se connecter
# dans notre cas, elle est sur la même machine que le serveur, l'adresse
est donc 127.0.0.1
DB_HOST = 127.0.0.1 #localhost
# le nom de la base de données
DB_DATABASE = "db_mimi"
# nom de l'utilisateur de la base de données avec qui on fera les
requêtes
# utiliser un utilisateur spécial et pas root qui a tous les droits
DB_USER = "nomutilisateur"
# le mot de passe de l'utilisateur
DB_PASSWORD = "motdepasseutilisateur"
# le port utilisé par la base de données
# cette variable n'est pas utilisée car la base de données est sur le
port par défaut 3306
DB_PORT = 3306
# le SGBD utilisé, ici MariaDB
# si on change de SGBD, il faudra modifier cette ligne
DB_DIALECT ="mariadb"
```

Variables pour l'envoi de mail

Pour envoyer des mails de récupération de mot de passe, nous utilisons la librairie nodemailer (https://nodemailer.com/about/). Il se peut que le service que vous utilisez ne soit pas compatible avec nodemailer : la liste des services supportés sont disponibles à l'adresse : https://nodemailer.com/smtp/well-known/. Si vous voulez configurer une adresse mail, il faudra remplir les informations suivantes :

```
# le nom du service d'envoi de mail utilisé, par défaut nous avons mis
gmail
MAIL_SERVICE = "gmail"
# l'adresse mail avec laquelle on enverra les mails
MAIL_ID = "testmimi.health@gmail.com"
# le mot de passe de l'adresse mail
MAIL_PASSWORD = "mdp"
# le secret pour générer le token qui sera envoyé par mail
PASSWORD_TOKEN_SECRET=AypTtGxv94A2OUuW
```

Documentation des fichiers

Pour faciliter la maintenance, les classes et constantes sont annotés en JSDoc. Une documentation a été générée séparément à l'aide de la librairie JSDoc. Vous la trouverez dans un dossier "JSDoc" joint à cette documentation. Pour l'instant, il y a les contrôleurs et les middlewares qui y sont décrits. Il est possible d'embellir et de changer la présentation de la documentation.
La mise en page a été fournie par docdash.
La documentation est aussi disponible à l'adresse ci-dessous :
https://github.com/RomainReghem/MiMi/tree/main/doc

Base de données

SGBD utilisé : MariaDB (changeable au besoin grâce à Sequelize, un ORM)
Ram nécessaire : 2go
Le nom des tables est sensible à la casse, leur nom doit être écrit en minuscule.
Pour de l'aide sur la gestion de la base de données : https://mariadbtips.com/

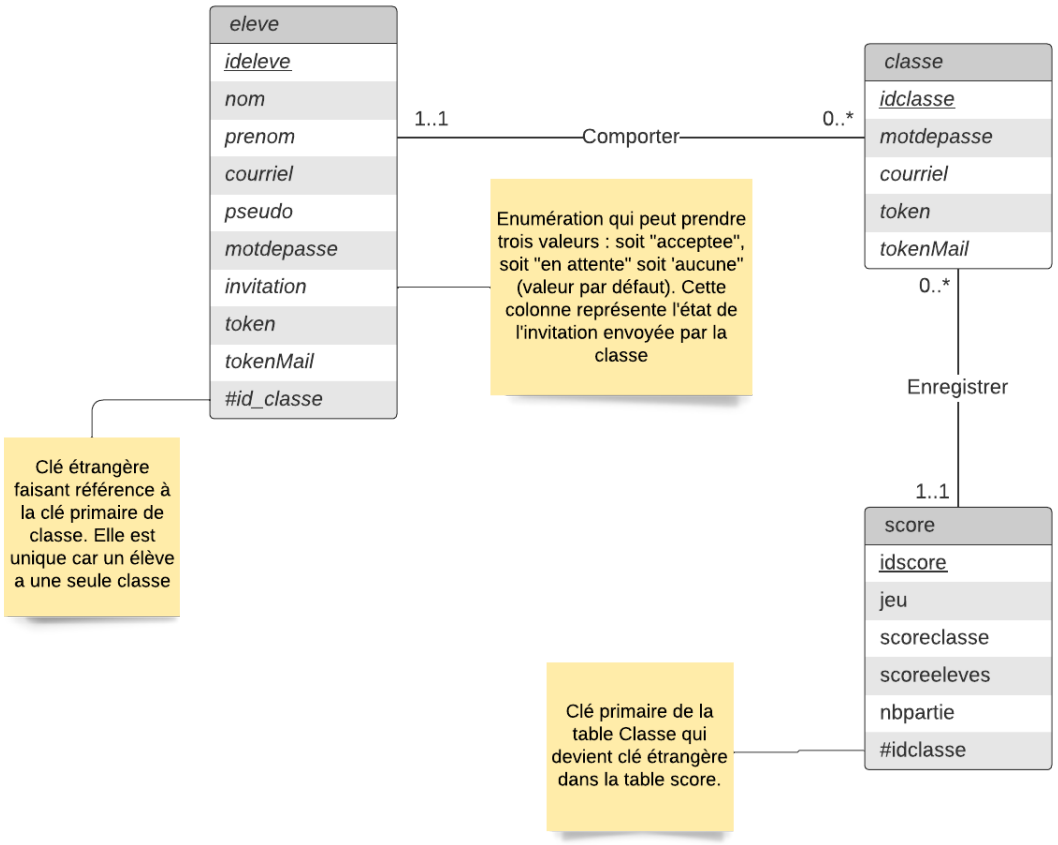


Diagramme UML relationnel de la base de données db_mimi

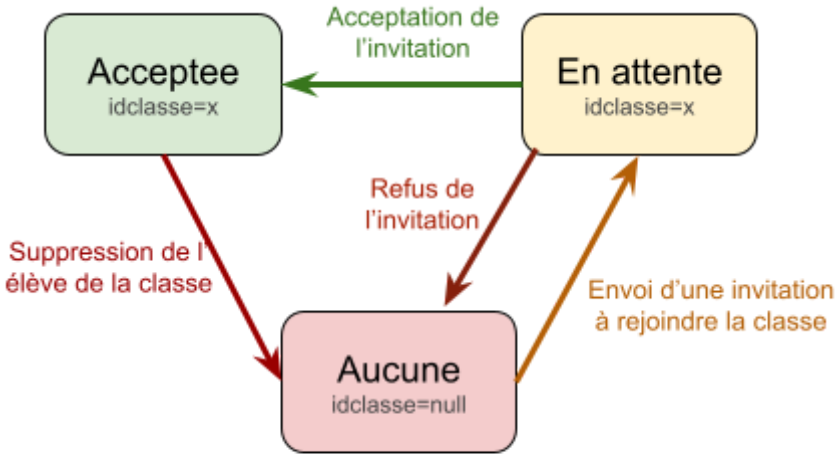
Définition des données

eleve

C'est la table qui contient notamment toutes les informations qu'un élève a renseigné.
Elle est composée des attributs suivants :

- *ideleve* : clé primaire de la table Eleve, est générée automatiquement sous la forme d'un entier qui s'incrémente de un à chaque nouvelle insertion.
- *nom* : nom de famille de l'élève, est une chaîne de caractères non nulle.
- *prenom* : prénom de l'élève, est une chaîne de caractères non nulle.
- *courriel* : adresse mail de l'élève, doit être unique dans toute la base de données (c'est-à-dire que même une classe ne peut pas avoir la même adresse mail qu'un élève). C'est une chaîne de caractères non nulle.

- *pseudo* : le pseudo d'un élève, il n'est pas unique. Il sera affiché sur le profil de l'élève. C'est une chaîne de caractère non nulle.
- *motdepasse* : le mot de passe de l'élève, sert lors de la connexion. Il n'est pas en clair dans la base de données, mais encodé grâce à la librairie bcrypt. C'est une chaîne de caractère non nulle.
- *invitation* : représente le statut d'une invitation à rejoindre une classe. C'est une énumération composée de trois valeurs: "aucune", "en attente" et "acceptee" ; la valeur par défaut est "aucune". Le schéma ci-dessous représente quand chaque valeur est utilisée.
- *idclasse* : clé étrangère représentant la clé primaire de la table classe. C'est donc un entier qui peut être nul (un élève n'a pas forcément de classe), plusieurs élèves peuvent être liés à la même classe. Idclasse est lié à l'invitation, il est nul sans invitation, et il ne doit pas changer quand l'invitation passe de 'en attente' à 'acceptee'.
- *token* : le refresh token de l'élève, s'il s'est connecté il y a moins d'un jour, sinon c'est une chaîne de caractères vide.
- *tokenMail* : le token permettant à un élève de réinitialiser son mot de passe. Il a une durée de 30 minutes et contient l'email ainsi que le rôle (eleve) de l'utilisateur.



Les trois états du statut de l'invitation

```
MariaDB [db_mimi]> describe eleve;
+------------+-----------------------------------------+------+-----+---------+----------------+
| Field      | Type                                    | Null | Key | Default | Extra          |
+------------+-----------------------------------------+------+-----+---------+----------------+
| ideleve    | int(11)                                 | NO   | PRI | NULL    | auto_increment |
| prenom     | varchar(45)                             | NO   |     | NULL    |                |
| nom        | varchar(45)                             | NO   |     | NULL    |                |
| courriel   | varchar(100)                            | NO   | UNI | NULL    |                |
| pseudo     | varchar(45)                             | NO   |     | NULL    |                |
| motdepasse | varchar(100)                            | NO   |     | NULL    |                |
| idclasse   | int(11)                                 | YES  | MUL | NULL    |                |
| invitation | enum('aucune','en attente','acceptee')  | NO   |     | aucune  |                |
| token      | varchar(200)                            | YES  |     |         |                |
| tokenMail  | varchar(250)                            | YES  |     | NULL    |                |
+------------+-----------------------------------------+------+-----+---------+----------------+
```

classe

C'est la table qui contient les informations sur le compte des classes.
Elle est composée de :

- *idclasse* : clé primaire de la table Classe ; elle est générée automatiquement sous la forme d'un entier qui s'incrémente de un à chaque nouvelle insertion.
- *courriel* : adresse mail du représentant de la classe, doit être unique dans toute la base de données (c'est-à-dire que même un élève ne peut pas avoir la même adresse mail). C'est une chaîne de caractères non nulle.
- *motdepasse* : le mot de passe de la classe, sert lors de la connexion. Il n'est pas en clair dans la base de données, mais encodé grâce à la librairie bcrypt. C'est une chaîne de caractère non nulle.
- *token* : le token de l'élève, s'il s'est connecté il y a moins d'un jour, sinon c'est une chaîne de caractères vide.
- *tokenMail* : le token permettant à un élève de réinitialiser son mot de passe. Il a une durée de 30 minutes et contient l'email ainsi que le rôle (eleve) de l'utilisateur.

```
MariaDB [db_mimi]> describe classe;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| idclasse   | int(11)      | NO   | PRI | NULL    | auto_increment |
| courriel   | varchar(100) | NO   | UNI | NULL    |                |
| motdepasse | varchar(100) | NO   |     | NULL    |                |
| token      | varchar(200) | YES  |     |         |                |
| tokenMail  | varchar(250) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

score

C'est une table qui contient tous les scores des jeux auxquels ont joué ensemble une classe et les élèves de cette classe. Le score est relatif à la classe; si la classe était supprimée, alors le score aussi serait supprimé, mais si un élève de la classe était supprimé, les scores ne changeraient pas.
Elle est composé de :

- *idscore* : entier. Clé primaire de la table Classe ; elle est générée automatiquement et s'incrémente de un à chaque nouvelle insertion.
- *jeu* : énumération composée de "tictactoe", … . C'est le nom du jeu duquel on enregistre le score.
- *scoreclasse* : entier naturel, par défaut 0. C'est le nombre de parties remportées par la classe.
- *scoreeleves* : entier naturel, par défaut 0. Cela correspond au nombre de parties du jeu remportées par les élèves de la classe. Même si un élève quitte la classe, ce nombre ne change pas car il est rattaché à la classe.
- *nbpartie* : entier naturel, par défaut 0. Cela correspond aux nombre de parties jouées par une classe pour un jeu avec ses élèves. Il est incrémenté à chaque fin de partie par la classe.

- idclasse : entier. C'est une clé étrangère qui fait référence à la table Classe depuis son identifiant. Une classe peut avoir plusieurs scores (différents jeux) et un score a obligatoirement une classe.

```
MariaDB [db_mimi]> describe score;
+-------------+--------------------+------+-----+---------+----------------+
| Field       | Type               | Null | Key | Default | Extra          |
+-------------+--------------------+------+-----+---------+----------------+
| idscore     | int(11)            | NO   | PRI | NULL    | auto_increment |
| jeu         | enum('tictactoe')  | NO   |     | NULL    |                |
| scoreclasse | int(11)            | NO   |     | 0       |                |
| scoreeleves | int(11)            | NO   |     | 0       |                |
| nbpartie    | int(10) unsigned   | NO   |     | 0       |                |
| idclasse    | int(11)            | NO   | MUL | NULL    |                |
+-------------+--------------------+------+-----+---------+----------------+
```

Mise en place de la base de données sur le serveur

Installation

Ce tutoriel permet la mise en place d'une base de données sur une machine sous Debian 10 : https://www.digitalocean.com/community/tutorials/how-to-install-mariadb-on-debian-10

Création d'un utilisateur

Si jamais vous voulez utiliser les utilisateurs déjà présents dans la base de données, vous trouverez le nom d'utilisateur ainsi que son mot de passe dans le fichier .env.

```
MariaDB [db_mimi]> SELECT User, Host FROM mysql.user;
+---------+-----------+
| User    | Host      |
+---------+-----------+
| devmimi | 127.0.0.1 |
| mimi    | 127.0.0.1 |
| root    | localhost |
+---------+-----------+
```

Les utilisateurs présent dans la base de données

Script pour créer un utilisateur :

```
CREATE USER 'utilisateur'@'127.0.0.1' identified BY 'mdp';
```

Script pour attribuer des droits CRUD à un utilisateur sur la base de données db_mimi :

```
GRANT CREATE, DELETE, INSERT, SELECT, UPDATE, TRIGGER
ON db_mimi
TO 'utilisateur'@'127.0.0.1';
```

Script pour créer un utilisateur et lui attribuer des droits :

```
GRANT SELECT, CREATE, UPDATE, INSERT, DELETE, TRIGGER
ON nom_base_de_données.*
```

```
TO 'utilisateur'@'127.0.0.1'
IDENTIFIED BY 'mot_de_passe';
```

Pour créer un utilisateur avec des droits administrateur (pour pouvoir modifier la base de données) :

```
GRANT ALL on *.* TO 'utilisateur'@'127.0.0.1' IDENTIFIED BY
'mdpmimi@DEV!' WITH GRANT OPTION;
```

Pour valider les modifications et les attributions de privilèges :

```
FLUSH PRIVILEGES;
```

Pour voir les privilèges d'un utilisateur :

```
SHOW GRANTS FOR mimi@127.0.0.1;
```

Attention : le mot de passe ne doit pas contenir de caractères spéciaux, cela pourrait rendre impossible la connexion à la base de données.

Si vous créez un nouvel utilisateur pour faire des requêtes, dans le fichier .env, il faudra changer la valeur de la variable DB_USER et DB_PASSWORD.

Connexion à la base de données

Depuis l'invite de commande du serveur où est située la base de données, on peut se connecter à la base de données.
Pour se connecter avec l'utilisateur root (tous les droits) :

```
sudo mysql
```

Pour se connecter avec un autre utilisateur (vous devez avoir le mot de passe) :

```
mysql -u nomutilisateur -p -h "127.0.0.1"
```

Suppression des données

Pour supprimer toutes les données d'une table, si jamais ce ne sont que des données de test (commande à exécuter avec un utilisateur ayant les droits) :

```
truncate table score;
```

ou

```
delete from score;
```

Attention : lors de la suppression des données de la classe, il faut veiller à bien supprimer celles des scores et des élèves y faisant référence avant.

Démarrer, redémarrer et vérifier le statut de la base de données

Pour lancer la base de données, entrez dans une invite de commande :

```
sudo systemctl start mariadb
```

Pour vérifier le statut du SGBD (afin de savoir s'il est en marche ou non) :

```
sudo systemctl status mariadb
```

Pour redémarrer la base de données, après avoir effectué des changements dessus :

```
sudo systemctl restart mariadb
```

Script MySQL

Création des tables

Table eleve :

```
CREATE TABLE `eleve` (
  `ideleve` int NOT NULL AUTO_INCREMENT,
  `prenom` varchar(45) NOT NULL,
  `nom` varchar(45) NOT NULL,
  `courriel` varchar(100) NOT NULL,
  `pseudo` varchar(45) NOT NULL,
  `motdepasse` varchar(100) NOT NULL,
  `idclasse` int DEFAULT NULL,
  `invitation` enum('aucune','en attente','acceptee') NOT NULL DEFAULT
'aucune',
  `token` varchar(200) DEFAULT '',
  PRIMARY KEY (`ideleve`),
  UNIQUE KEY `courriel_UNIQUE` (`courriel`),
  KEY `fk_classe_eleve_idx` (`idclasse`),
  CONSTRAINT `fk_classe_eleve` FOREIGN KEY (`idclasse`) REFERENCES
`classe` (`idclasse`)
) ENGINE=InnoDB AUTO_INCREMENT=50 DEFAULT CHARSET=utf8mb3;
```

Table classe :

```
CREATE TABLE `classe` (
  `idclasse` int NOT NULL AUTO_INCREMENT,
  `courriel` varchar(100) NOT NULL,
  `motdepasse` varchar(100) NOT NULL,
  `token` varchar(200) DEFAULT '',
  PRIMARY KEY (`idclasse`),
  UNIQUE KEY `courriel_UNIQUE` (`courriel`)
) ENGINE=InnoDB AUTO_INCREMENT=55 DEFAULT CHARSET=utf8mb3;
```

Table score :

```
CREATE TABLE `score` (
  `idscore` int NOT NULL AUTO_INCREMENT,
  `jeu` enum('tictactoe') NOT NULL,
  `scoreclasse` int NOT NULL DEFAULT '0',
  `scoreeleves` int NOT NULL DEFAULT '0',
  `nbpartie` int unsigned NOT NULL DEFAULT '0',
  `idclasse` int NOT NULL,
  PRIMARY KEY (`idscore`),
  KEY `fk_score_classe_idclasse_idx` (`idclasse`),
  CONSTRAINT `fk_score_classe_idclasse` FOREIGN KEY (`idclasse`)
REFERENCES `classe` (`idclasse`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb3;
```

Déclencheurs

Des déclencheurs (ou triggers) ont été définis pour vérifier qu'une adresse mail entrée soit bien unique. En effet, les classes ne peuvent pas avoir la même adresse mail qu'un élève et inversement.
Certains déclencheurs (lors de la mise à jour de la table élève ou lors de la suppression de la table classe) vérifient aussi la partie sur les invitations d'un élève par une classe.

Script du déclencheur MySQL/MariaDB avant insertion sur la table ELEVE :

```
CREATE TRIGGER `eleve_BEFORE_INSERT`
BEFORE INSERT ON `eleve`
FOR EACH ROW
BEGIN
      DECLARE nbcourriel INT;
      SET nbcourriel := (SELECT COUNT(*)
      FROM CLASSE
      WHERE courriel=NEW.courriel);

      IF nbcourriel > 0  THEN
            SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'L adresse existe
deja dans la base de donnees de la classe.';
      END IF ;
END;
```

Script du déclencheur MySQL/MariaDB avant modification sur la table ELEVE :

```
CREATE TRIGGER `eleve_BEFORE_UPDATE`
BEFORE UPDATE ON `eleve`
FOR EACH ROW
BEGIN
      DECLARE nbcourriel INT;
```

```
	SET nbcourriel := (SELECT COUNT(*)
	FROM CLASSE
	WHERE courriel=NEW.courriel);

	IF nbcourriel > 0  THEN
		SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'L adresse existe
deja dans la base de donnees de la classe.';
	END IF ;
END;
```

Script du déclencheur MySQL/MariaDB avant insertion sur la table CLASSE :

```
CREATE TRIGGER `classe_BEFORE_INSERT`
BEFORE INSERT ON `classe`
FOR EACH ROW
BEGIN
	DECLARE nbcourriel INT;
	SET nbcourriel := (SELECT COUNT(*)
	FROM eleve
	WHERE courriel=NEW.courriel);

	IF nbcourriel > 0  THEN
		SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'L\'adresse
existe deja dans la base de donnees de l\'eleve.';
	END IF ;
END;
```

Script du déclencheur MySQL/MariaDB avant modification sur la table CLASSE :

```
CREATE TRIGGER `classe_BEFORE_UPDATE`
BEFORE UPDATE ON `classe`
FOR EACH ROW
BEGIN
DECLARE nbcourriel INT;
	SET nbcourriel := (SELECT COUNT(*)
	FROM eleve
	WHERE courriel=NEW.courriel);

	IF nbcourriel > 0  THEN
		SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'L''adresse
existe deja dans la base de donnees de l''eleve.';
END IF;
END;
```

Script du déclencheur MySQL/MariaDB avant suppression sur la table CLASSE :

```
CREATE TRIGGER `classe_BEFORE_DELETE`
BEFORE DELETE ON `classe`
FOR EACH ROW
BEGIN
UPDATE eleve
      SET idclasse=NULL AND invitation="aucune"
      WHERE old.idclasse = idclasse;
END;
```

Script du déclencheur MySQL/MariaDB avant modification sur la table SCORE:

```
CREATE TRIGGER `score_BEFORE_UPDATE`
BEFORE UPDATE ON `score`
FOR EACH ROW
 BEGIN
    IF new.scoreclasse>new.nbpartie OR new.scoreeleves>new.nbpartie THEN
            SIGNAL SQLSTATE '50006' SET MESSAGE_TEXT =  "Ce score est
impossible a obtenir !";
    END IF;
END;
```

Tableau des erreurs SQL définies dans les déclencheurs :

Code	Signification	Où
50001	On essaie d'insérer dans la table ELEVE une adresse mail qui appartient déjà à un compte CLASSE.	`BEFORE INSERT ON ELEVE` `BEFORE UPDATE ON ELEVE`
50002	On essaie d'insérer dans la table CLASSE une adresse mail qui appartient déjà à un compte ELEVE.	`BEFORE INSERT ON CLASSE` `BEFORE UPDATE ON CLASSE`
50003	On essaie de mettre le statut d'une invitation à "acceptee" sans qu'elle soit passée par le statut "en attente". (cf. diagramme " Les trois états du statut de l'invitation " plus haut)	`BEFORE UPDATE ON ELEVE`
50004	On essaie de mettre le statut d'une invitation à "acceptee" avec une classe différente de celle qui a envoyé l'invitation (statut "en attente").	`BEFORE UPDATE ON ELEVE`
50005	On essaie d'ajouter une invitation (statut "en attente"), alors que l'élève a soit déjà une invitation en attente, soit a déjà accepté une invitation.	`BEFORE UPDATE ON ELEVE`
50006	Le score de la classe ou de l'élève est supérieur au nombre de parties jouées, or le score ne peut dépasser le nombre de parties jouées, car à chaque partie jouée, le nombre de points maximal récolté est de 1.	`BEFORE UPDATE ON SCORE`

Mesures de sécurité mises en place

Pour assurer la sécurité des données, voici des mesures mises en place :
- Ne jamais publier le fichier .env, il contient entre autres les clés de chiffrement pour les tokens.
- Changer régulièrement les clés secrètes et le mot de passe de la base de données.
- Hasher le mot de passe donné par l'utilisateur afin de le rendre illisible dans la base de données.
- Utiliser des cookies ainsi que des tokens à durée limitée afin de vérifier l'identité des utilisateurs et valider leurs sessions.
- Toujours utiliser des variables lorsqu'on récupère des données envoyées par le client : cela évite les attaques par injection SQL.
- Vérifier la validité des données dans le serveur avant de les insérer dans la base de données : pour éviter que ce ne soit à la base de données de déclencher des erreurs.
- Utiliser des déclencheurs et des contraintes sur la base de données afin d'éviter l'insertion de mauvaises données, en dernier recours.
- Création d'un utilisateur spécifique pour le serveur, qui n'aura seulement le droit de requêtes CRUD sur la base de données, afin d'éviter que l'on puisse changer la base de données.
- La base de données ne doit être accessible que depuis le serveur.

Cependant, ils restent des mesures à prendre, notamment au niveau des stockages des données, de leur conservation et suppression. Il faut aussi faire des sauvegardes régulières de la base de données. Ajouter une vérification des adresses mails et une suppression des comptes inactifs.