Projet Forum

ALEM Lina

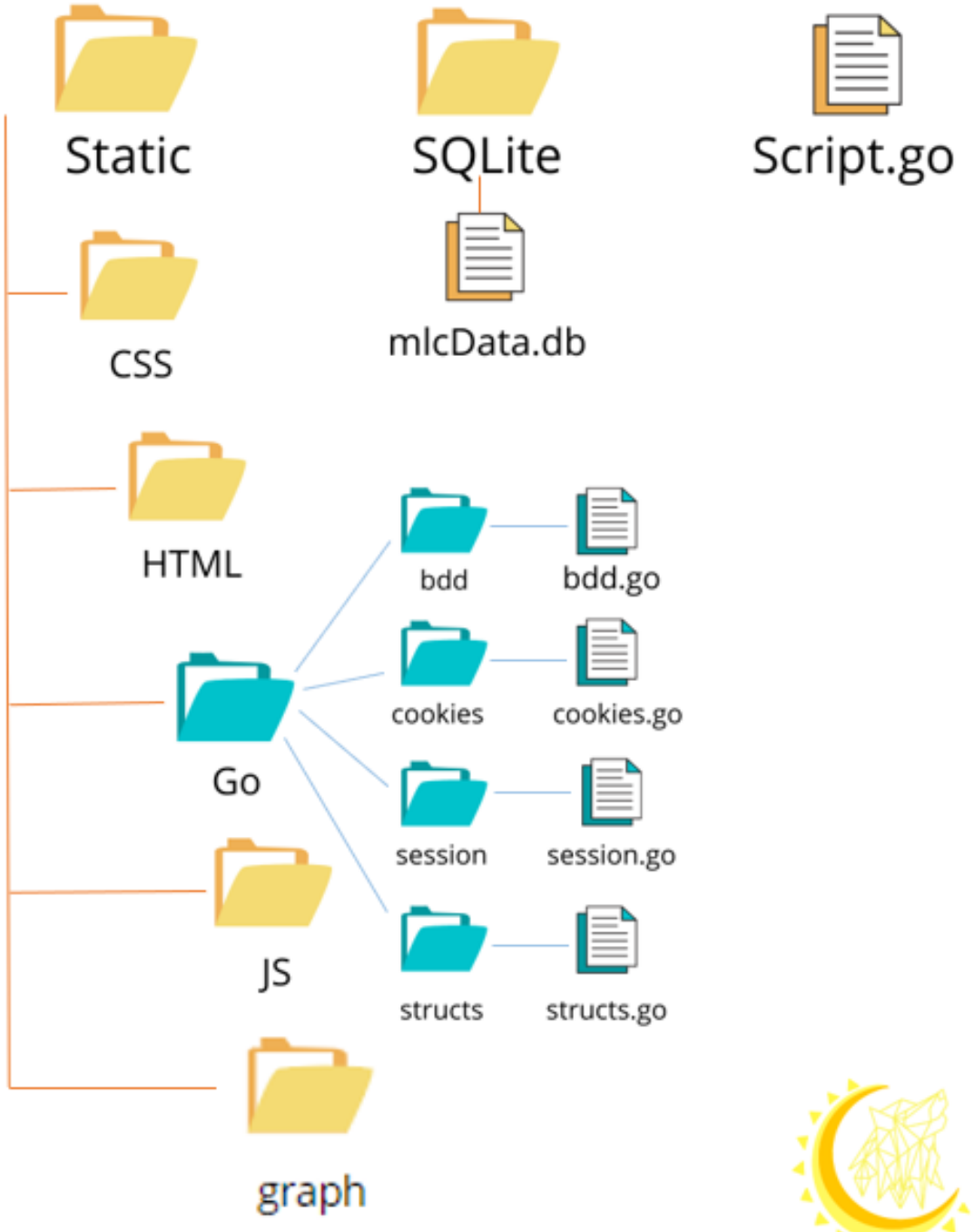
SARTORI-BOUTY Marie-Laure

VIAL--SIMON Benjamin

SIAUD Cyprien

SOMMAIRE

ARBORESCENCE



PRÉAMBULE

Les sites de type forums / réseaux sociaux sont devenus des outils inestimables en matière d'éducation, de marketing et de communication. Jouant un rôle important dans la vie de chacun, il est souvent plus facile et plus pratique d'accéder et de fournir des informations grâce à ces plateformes. Ce partage permet d'être des lieux de discussion anonyme. En tant que tels, ils nous montrent un aspect plus ludique des échanges parfois à but éducatif.

Nous avons choisi de créer un forum qui, se rapprochant d'un réseau social, permet de poster des articles dans différentes catégories se rapportant au quotidien. Les fonctionnalités additionnelles que nous avons choisi d'apporter nous semblaient être nécessaire à la bonne dynamique du site.

L'équipe est composée de quatre membres : Marie-Laure SARTORI BOUTY, Cyprien CIAUD, Benjamin VIAL—SIMON et Lina ALEM. Afin de mener à bien le projet, nous nous sommes partagé les tâches de façon à pouvoir avancer efficacement. Bien qu'une seule personne possède la base de données officielle, chaque membre du groupe a apporté son aide. Ainsi, grâce à l'extension LiveShare de VSCode, nous avons tous codé sur le même espace de travail, chacun gérant la partie qui lui a été assigné et commentant chaque fonction pour plus de détails.

Bien que l'organisation de l'équipe semblait adaptée au début du projet, son appréhension et la gestion du temps nous a rendu la tâche davantage compliquée que ce que nous avions prévu. Nous avons donc pu constater un retard quant à la réalisation des fonctionnalités listées dans la grille de notation. Voilà pourquoi, nous n'avons pu valider certains points du projet.

PRÉREQUIS

En vue d'utiliser le forum en local, il est nécessaire d'installer en amont quelques paquets :

```
go install      github.com/google/uuid
go get          github.com/mattn/go-sqlite3
```

Nécessaire à l'utilisation des cookies et de la base de données SQLite3, leur présence en import permettra de les utiliser.

Pour une meilleure compréhension des structures et des noms de fonctions, nous vous recommandons d'ouvrir le fichier Draw.io ci-joint dans le dossier .ZIP, aussi présent sous forme de capture d'écran en Annexe (*page 12*).

Et afin que le serveur HTTPS puisse fonctionner en local, ces trois commandes sont obligatoires à rentrer dans un terminal WSL :

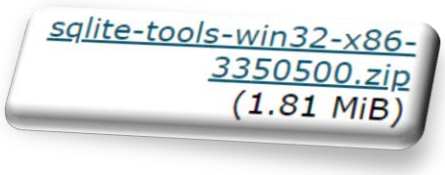
```
openssl genrsa -out https-server.key 2048

openssl ecparam -genkey -name secp384r1 -out
https-server.key

openssl req -new -x509 -sha256 -key https-
server.key -out https-server.crt -days 3650
```

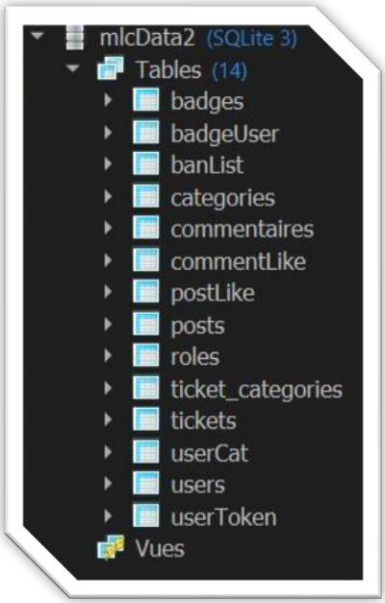
INTRODUCTION AU SQLITE

Afin de gérer la base de données du projet, nous avons dû télécharger le logiciel SQLite via sqlite.org en choisissant « Precompiled Binaries for Windows » sous la version suivante :



Une fois installée, il nous fallait créer notre base de données. Pour ce faire, et depuis une ligne de commande du .exe situé dans le dossier SQLite, nous y avons inséré des tables et des données rangées par colonnes. Un fichier .db y résulte : nous l'avons renommé mlcData.db.

Grâce au logiciel SQLite Studio, nous pouvons avoir une mise en vue de nos tables en base de données :



Ainsi, SQLite permet de répertorier chaque donnée de notre projet qu'on peut modifier à notre guise. Nous avons pour la plupart utilisé des clés étrangères, utiles pour lier plusieurs tables et garantir la cohérence de leurs données.

DOSSIER STATIC

Le dossier Static est le dossier permettant de gérer la structure du projet. Elle contient le squelette et le design du site, ainsi que toutes les fonctionnalités dynamiques que nous avons souhaité apporter. Nous avons suivi, comme ligne de conduite, un MCD et MLD réalisé en amont pour davantage d'organisation au sein de l'équipe. Comme vous pourrez le voir dans le code, le nom des fonctions est capitalisé et en anglais, de même que pour les variables. L'absence de nombre magique et la présence de commentaires ont pour but de faciliter la lecture du code et de respecter les bonnes pratiques et normes de codage.

i. CSS et HTML

Le dossier HTML contient chaque page de notre forum : ce sont des templates qui, contrairement à des fichiers HTML ordinaires, se composent des éléments de notre base de données, se remplissant ainsi automatiquement dès lors qu'un changement opère dans les données. Elles possèdent chacune leur fichier CSS voire un fichier JS en plus, en fonction du besoin. Dans l'optique d'alléger les fichiers, les éléments récurrents ont été placé dans un fichier HTML global qui sera appelé lorsque nécessaire grâce au Go : (header.partial.html, footer.partial.html)

De plus, de manière à remplir les pages de façon structurée, nous les avons générées en passant par le fichier layout.html qui se compose des balises obligatoires < !doctype html>, le <html>, <head> et <body> tout en définissant l'emplacement du header et du footer. Les fichiers autres, tels que le JS, ont été intégrés dans les templates HTML correspondant au besoin.

ii. JavaScript

Comme mentionné plus tôt, seulement quelques pages ont eu besoin d'un fichier JavaScript pour gérer certaines fonctionnalités utiles au site (*exemple : les popups de connexion et d'inscription, la recherche et la mise en page du tableau de bord des Administrateurs et des Modérateurs*). Ce langage a été un atout pour notre projet car il nous a été possible de réaliser une multitude d'opérations sur une page HTML sans avoir à la recharger dans son intégralité. L'ajout de plusieurs fonctionnalités nous a poussé à suivre le principe de Separation of Concerns qui consiste à découper un programme en plusieurs fichiers, pour qu'ils soient le plus indépendant possible.

iii. Golang

Structs

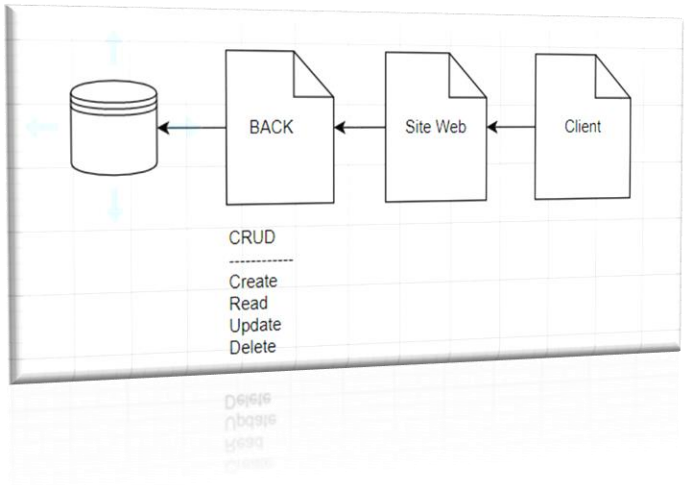
Toujours en suivant cette bonne pratique, nous avons fait le choix d'utiliser des paquets pour structurer au mieux le code. Parmi eux se trouve le package structs dans lequel sont déclarés chaque structure en fonction des tableaux de la base de données. Il s'agit donc d'un type de donnée défini qui permet de combiner des éléments différents types et sont donc utiles pour regrouper des données afin de former des enregistrements personnalisés. Ainsi, la table user dans la base de données possède une colonne correspond à un élément de la structure user : Id, Username, etc… On remplira ses données grâce au fichier bdd.go.

Cookies et Sessions

Les dossiers Cookies et Session gèrent, comme son nom l'indique, la session d'un utilisateur grâce aux cookies. Cookie.go va créer le cookie, modifier sa valeur et le détruire une fois la session expirée ou durant la déconnexion. Quant à Session.go, le fichier va permettre de récupérer un utilisateur grâce à la valeur de son cookie qui possède un identifiant unique (UUID, avec pour nom du cookie : Session). Il a également le rôle de vérifier si un utilisateur existe et si les informations entrées sont correctes. Il comporte aussi la fonction de redirection à la destruction du cookie lors de la déconnexion afin de faciliter l'expérience de l'utilisateur sur le site.

BDD

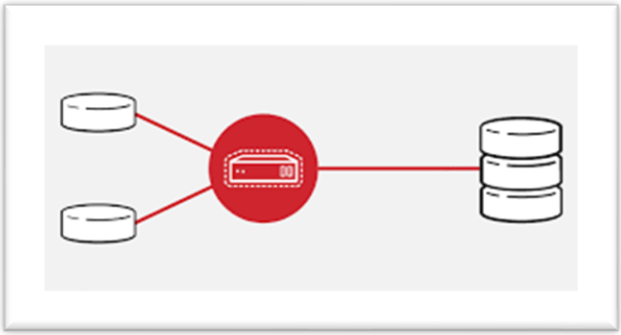
Le fichier de la base de données à un rôle essentiel dans la récupération des informations. C'est ici qu'on se chargera des opérations CRUD (acronyme signifiant Create, Read, Update, Delete).



En effet, chaque fonction fera le pont entre les éléments des tables en permettant leur ajout, modification et suppression tandis que certaines auront la possibilité de remplir les structs. C'est dans ce fichier que se regroupent les requêtes SQL, qui aident à la gestion de la base de données avec la sélection, la mise à jour et la suppression des données. Celles-ci, une fois remplies, seront récupérées par Script.go qui se chargera de les transmettre aux pages HTML.

SCRIPT.GO

Le fichier permet dans un premier temps la création du serveur ainsi que la gestion des pages HTML et leur redirection. Les envois de mails quand nécessaire, la gestion des permissions des utilisateurs à l'aide de middleware qui, comme son nom l'indique, joue le rôle de médiateur entre les accès et le grade de l'utilisateur.



De plus, les erreurs de requête sont toutes gérées par Script.go qui récupère les fonctions de Bdd.go et donc des données. En somme, ce fichier étant à la base de l'arborescence est crucial pour la bonne mise en utilisation du projet.

CONCLUSION

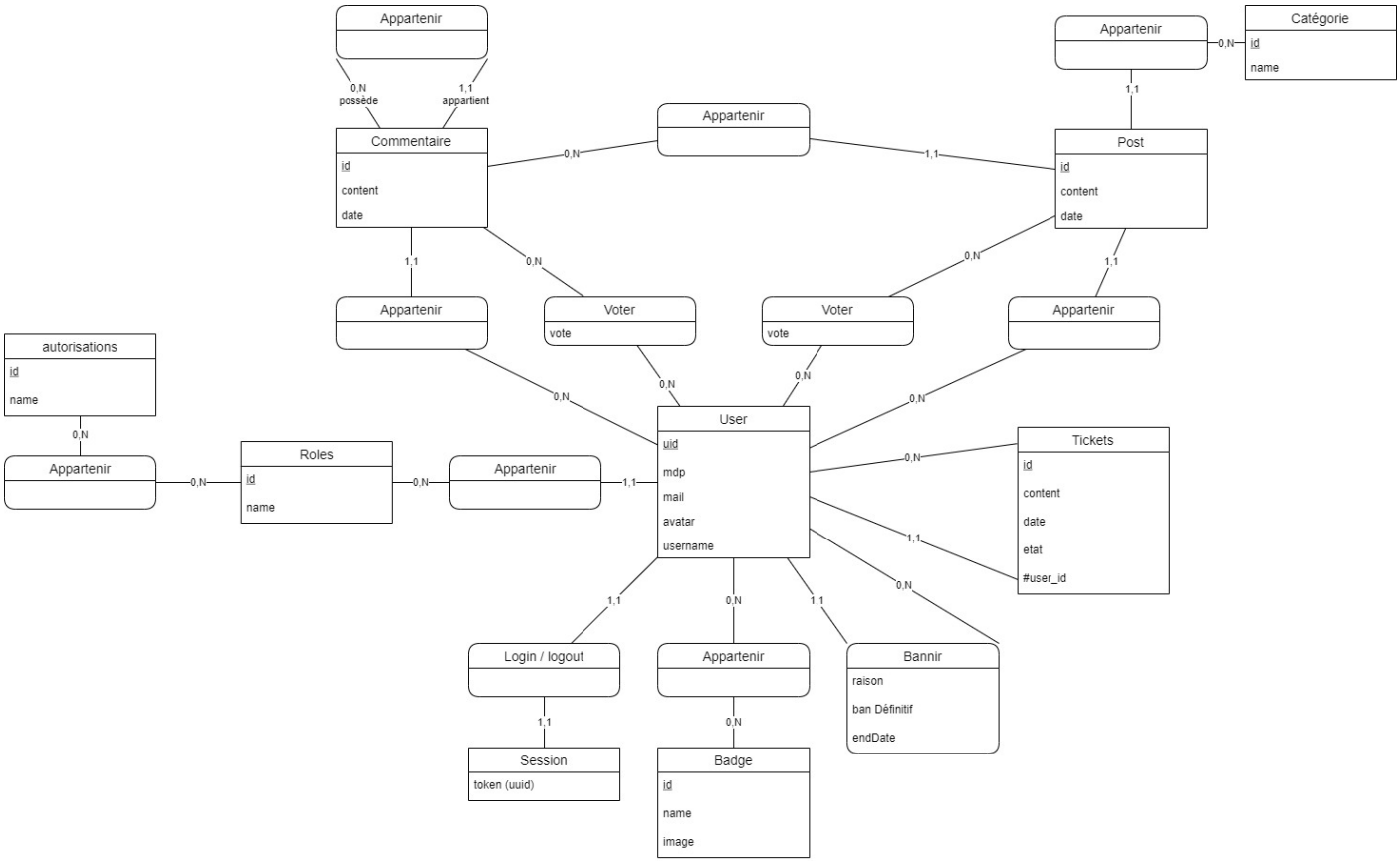
Ce travail de groupe sur un projet ayant un but bien défini a été réalisé de manière efficiente et sérieuse malgré le retard pris. De l'avis général, nous avons consolidé nos connaissances et appris à gérer le back-end, chose encore inédite pour certains d'entre nous. Nous sommes globalement satisfaits de ce que nous avons réalisé, malgré les nombreuses difficultés rencontrées et notre volonté de faire davantage.

Par ailleurs, les temps impartis à la réalisation du projet furent bref et il a fallu faire preuve de flexibilité et de persévérance pour respecter les délais mais également pour respecter les critères imposés par le projet. Somme toute, nous avons retrouvé lors de ces cinq semaines, les compétences, les contraintes mais aussi l'excitation d'un projet d'entreprise.

Enfin, au niveau de la gestion du projet en équipe, nous avons réussi à bien nous répartir les tâches afin de réaliser la plupart de nos objectifs fixés.

ANNEXE

MCD – Modèle Conceptuel de Données, soit la **représentation logique de l'organisation des informations et de leurs relations entre elles.**



MLD - Modèle Logique des Données, soit la représentation textuelle du schéma du MCD.

ROLES(id, name)
id : Clé primaire de la table ROLES

AUTORISATIONS(id, name)
id : Clé primaire de la table AUTORISATIONS

ROLEAUTH(#role_id, #autorisations_id)
#role_id, #autorisations_id : Clé primaire composée de la table ROLEAUTH

BAN(id, endDate, #bannedBy, raison, banDef, #user_id)
id : Clé primaire de la table BAN
#bannedBy : Clé étrangère qui lie "id" de la table USERS

BADGES(id, name, image)
id : Clé primaire de la table BADGES

USERS(id, username, mail, mdp, avatar, sessionToken, #roles_id)
id : Clé primaire de la table USERS

BADGEUSER(#user_id, #badge_id)
#user_id, #badge_id : Clé primaire composée de la table BADGEUSER

COMMENTAIRES(id, content, date, #user_id, #posts_id, #commentaires_id)
id : Clé primaire de la table COMMENTAIRE

POSTS(id, content, date, #user_id, #categories_id)
id : Clé primaire de la table POSTS

CATEGORIES(id, name)
id : Clé primaire de la table CATEGORIES

POSTLIKE(#user_id, #post_id, vote)
#user_id, #post_id : Clé primaire composée de la table POSTLIKE

COMMENTLIKE(#user_id, #commentaire_id, vote)
#user_id, #commentaire_id : Clé primaire composée de la table COMMENTLIKE

TICKETS(id, content, date, etat, #user_id)
id: Clé primaire de la table TICKETS
#user_id: Clé primaire composée de la table USERS