Projet GL

FAUCON Léo RULLAC Éloïse

Avril 2025

Sommaire

1 Présentation du Projet

Dans le cadre de ce projet, nous avons préféré choisir notre propre sujet plutôt que celui du gestionnaire de facture et avons donc choisi de concevoir et développer un logiciel de gestion d'agenda et de tâches.
L'objectif principal était de répondre à un besoin concret d'organisation personnelle, en proposant un outil à la fois intuitif, fonctionnel et adaptable aux différents profils d'utilisateurs.
À travers ce développement, nous avons mis en œuvre plusieurs compétences apprises au cours de ce module, tout en nous appliquant sur l'expérience utilisateur et sur la gestion de projet.
Ce rapport retrace les différentes étapes de la conception et de développement de l'application réalisés tout au long du projet.

2 Répartition des tâches

- Groupe : Cahier des charges, axe fonctionnel, statique et dynamique.
- Léo : Développement de la communication client-serveur (back-end)
- Éloïse : Développement IHM, UI et contrôleurs (front-end).

3 Planning du projet

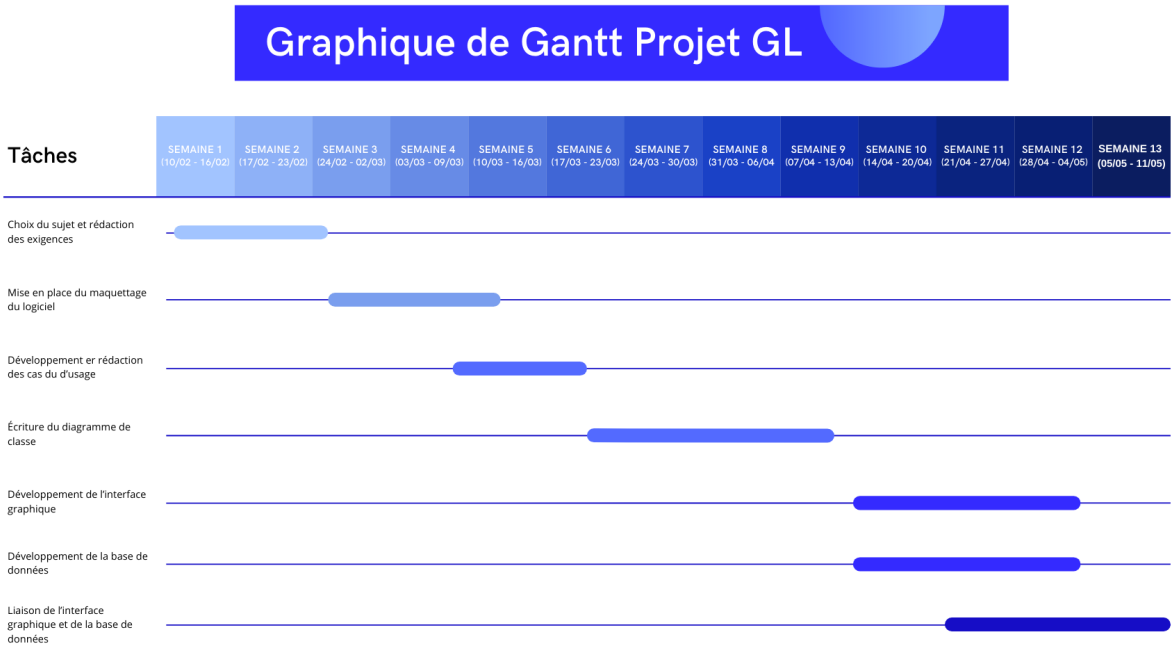

Figure 1: Graphique de Gantt

4 Exigences

SUJET : Agenda

4.1 Exigences

Accès utilisateur :

- AU010 : Le système doit permettre à l'utilisateur de se connecter via une adresse mail et un mot de passe.
- AU020 : Le système doit permettre à l'utilisateur de gérer son compte (adresse mail, mot de passe, nom, prénom).

Personnalisation :

- P010 : Le système doit permettre de paramétrer l'état sombre ou clair, le début du jour de la semaine, le langage, l'heure locale, les éventuelles notifications.
- P020 : Le système doit permettre à l'utilisateur de rechercher un événement (par titre ou description).
- P030 : Le système doit permettre à l'utilisateur de rechercher une tâche (par titre).
- P040 : Le système doit permettre à l'utilisateur de visualiser les détails d'un événement en cliquant dessus.
- P050 : Le système doit permettre à l'utilisateur de passer d'un événement à un autre dans l'ordre chronologique.
- P060 : Le système doit permettre à l'utilisateur de modifier l'affichage (jour, semaine, mois...).

Création et modification :

- CM010 : Le système doit permettre à l'utilisateur de créer des agenda.
- CM020 : Le système doit permettre à l'utilisateur de créer des tâches.
- CM030 : Le système doit permettre à l'utilisateur de modifier des tâches.
- CM040 : Le système doit permettre à l'utilisateur de supprimer des tâches.
- CM050 : Le système doit permettre à l'utilisateur d'annuler des tâches (affiche la tâche barrée).
- CM060 : Le système doit permettre à l'utilisateur de créer des évènements.
- CM070 : Le système doit permettre à l'utilisateur de modifier des événements.
- CM080 : Le système doit permettre à l'utilisateur de supprimer des événements.

- CM090 : Le système doit permettre à l'utilisateur d'annuler des événements (affiche l'événement barré).
- CM100 : Le système doit permettre de dupliquer un événement (affiche la page de création d'un événement avec les détails identique à l'événement dupliqué).
- CM110 : Les agenda doivent être caractérisées par leur couleur, leur visibilité, les personnes y ayant accès (caractérisées par leur adresse mail), les rappels de base
- CM120 : Les évènements doivent être caractérisées par agenda, par heure (de début ou de fin) ou éventuellement toute la journée, de l'heure locale, de la répétition, des gens concernés (référencés par leur mail), d'une localisation, des rappels, d'une couleur, d'une description, d'une visibilité si la catégorie est partagée, par leur état (annulé ou non).

Partage et communication :

- PC010 : Le système doit permettre à l'utilisateur de partager un événement.
- PC020 : Le système doit permettre à l'utilisateur de partager une tâche.
- PC030 : Le système doit permettre à l'utilisateur de partager son agenda (en mode lecture seule.
- PC040 : Le système doit permettre à l'utilisateur d'importer un agenda.
- PC050 : Le système doit envoyer un mail à l'utilisateur lorsqu'un de ses agenda se fait importer.
- PC060 : Le système doit envoyer un mail à l'utilisateur lorsqu'il importe un agenda.
- PC070 : Le système doit permettre à l'utilisateur de transformer un événement en récapitulatif textuel.
- PC080 : Le système doit permettre à l'utilisateur de transformer une tâche en récapitulatif textuel.

4.2 Implémentation des exigences

Exigence	Appliquée	Non appliquée	Partiellement appliquée
Accès utilisateur			
AU010	X		
AU020		X	
Personnalisation			
P010			X
P020		X	
P030		X	
P040		X	
P050		X	
P060	X		
Création et modification			
CM010	X		
CM020	X		
CM030	X		
CM040	X		
CM050	X		
CM060	X		
CM070	X		
CM080	X		
CM090	X		
CM100		X	
CM110			X
CM120			X
Partage et communication			
PC010		X	
PC020		X	
PC030			X
PC040		X	
PC050		X	
PC060		X	
PC070		X	
PC080		X	

Table 1: Implémentation des exigences

5 Maquettage

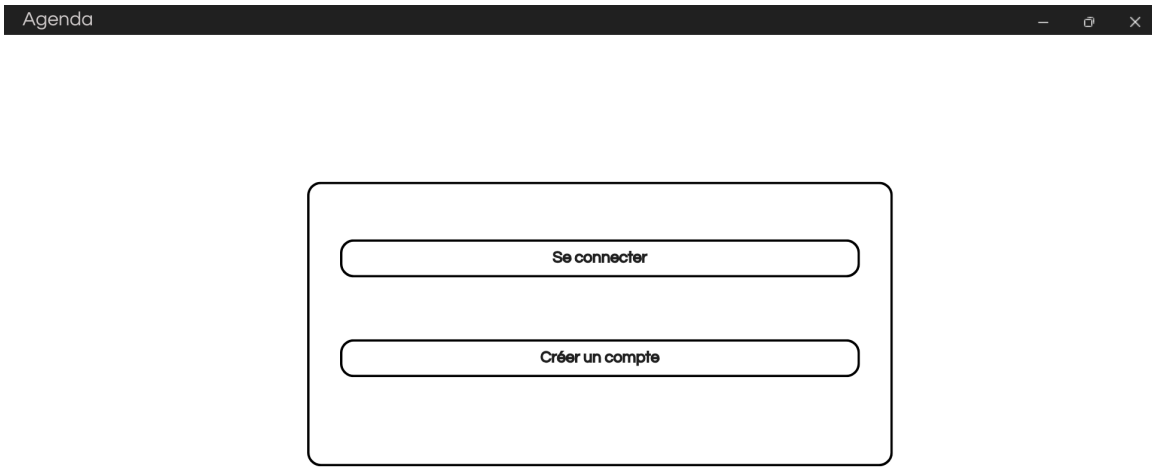

Figure 2: Écran d'accueil

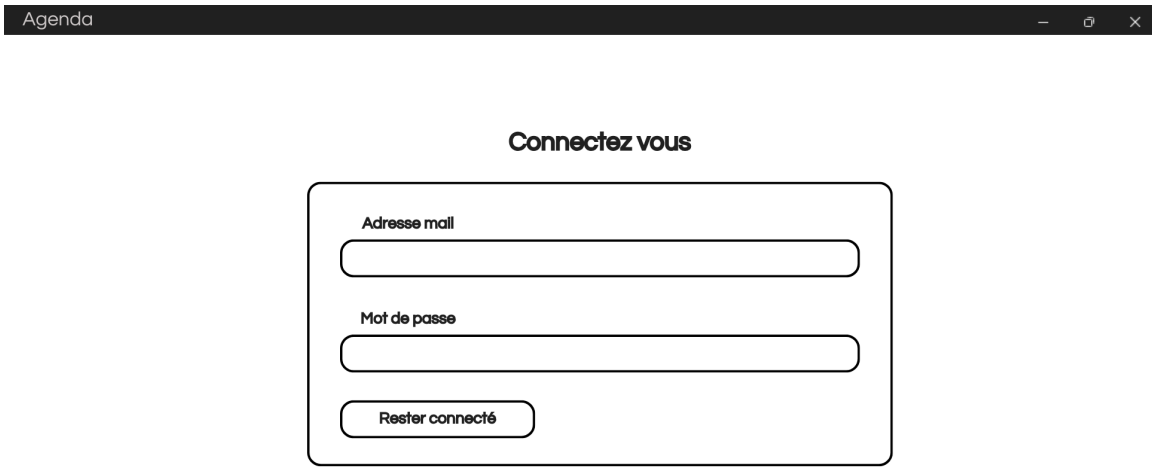

Figure 3: Écran de connexion

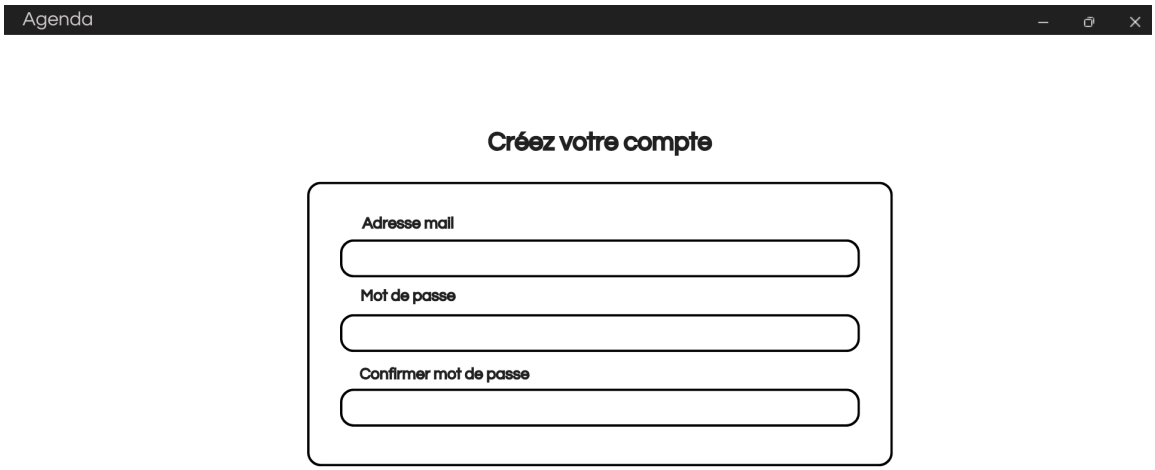

Figure 4: Écran de création de compte

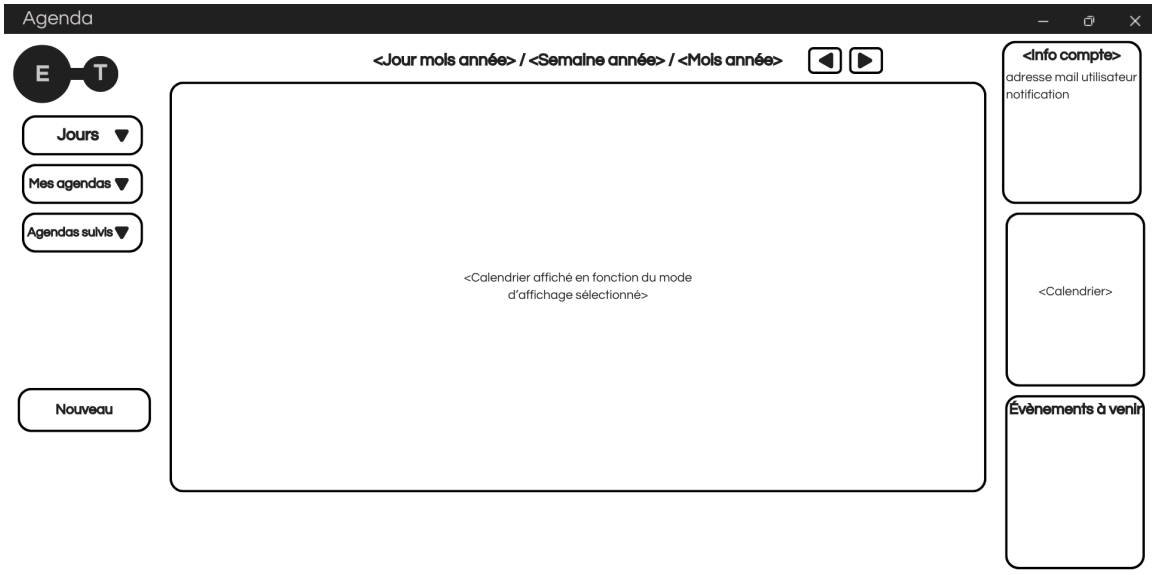

Figure 5: Affichage page événement

Figure 6: Liste des modes d'affichage possibles

Figure 7: Liste des agendas de l'utilisateur

Figure 8: Menu clic droit sur la liste d'agendas

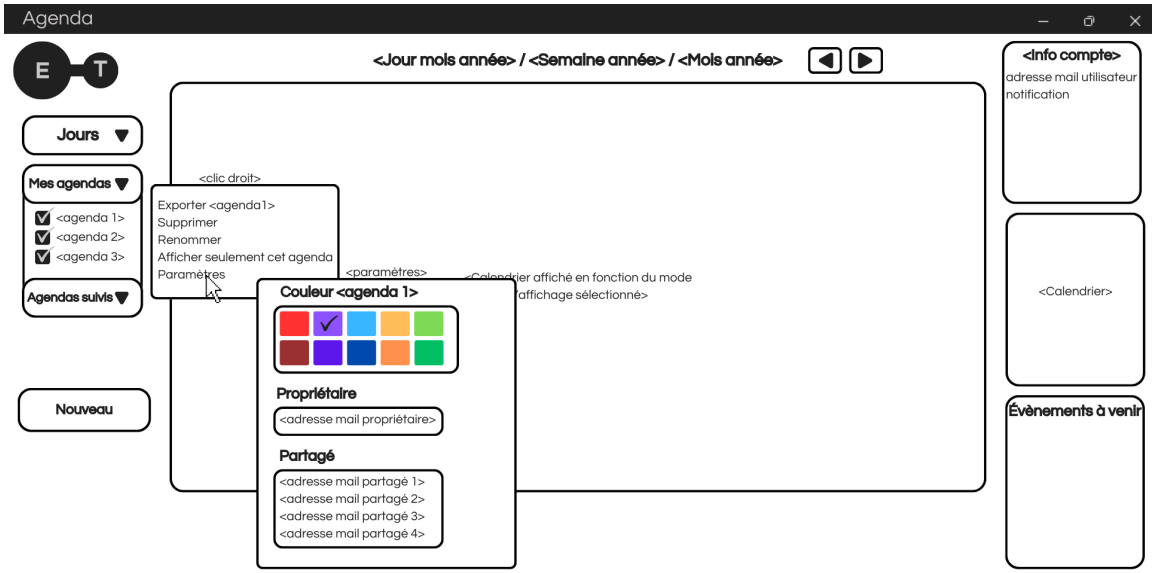

Figure 9: Clic sur les paramètres d'agenda

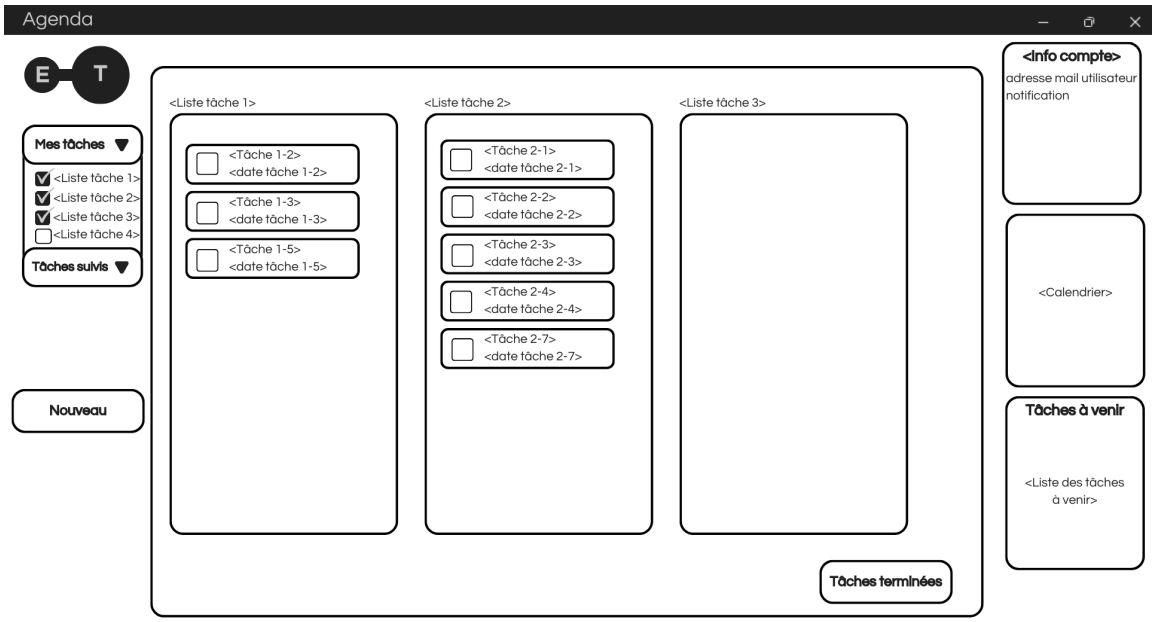

Figure 10: Affichage des tâches en cours

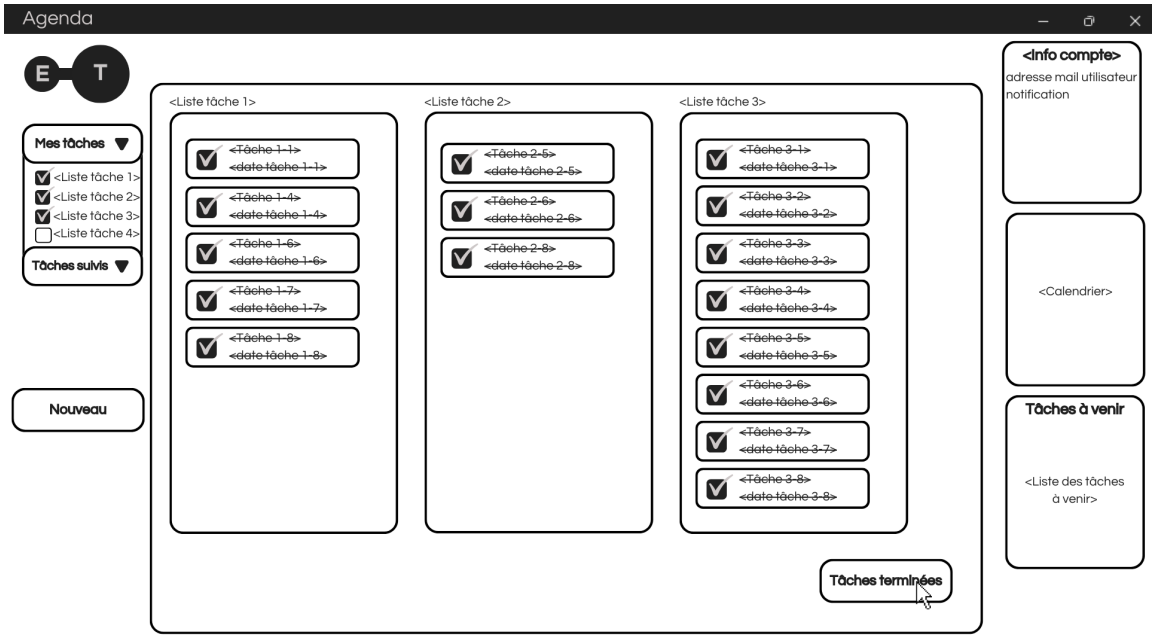

Figure 11: Affichage des tâches terminées

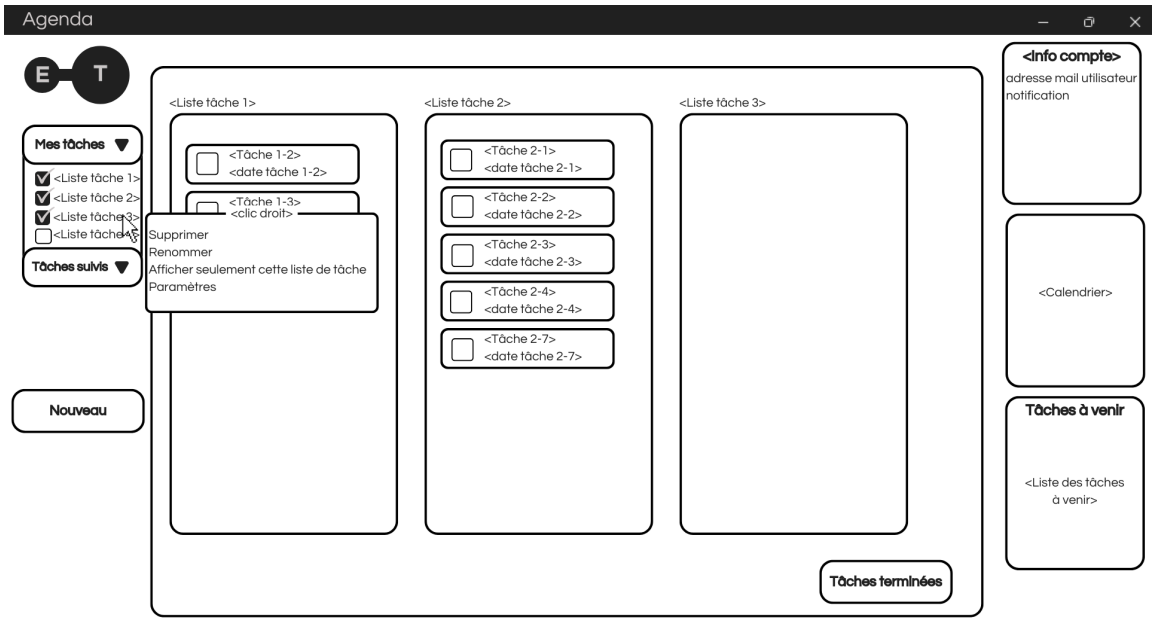

Figure 12: Clic droit sur la liste de tâche

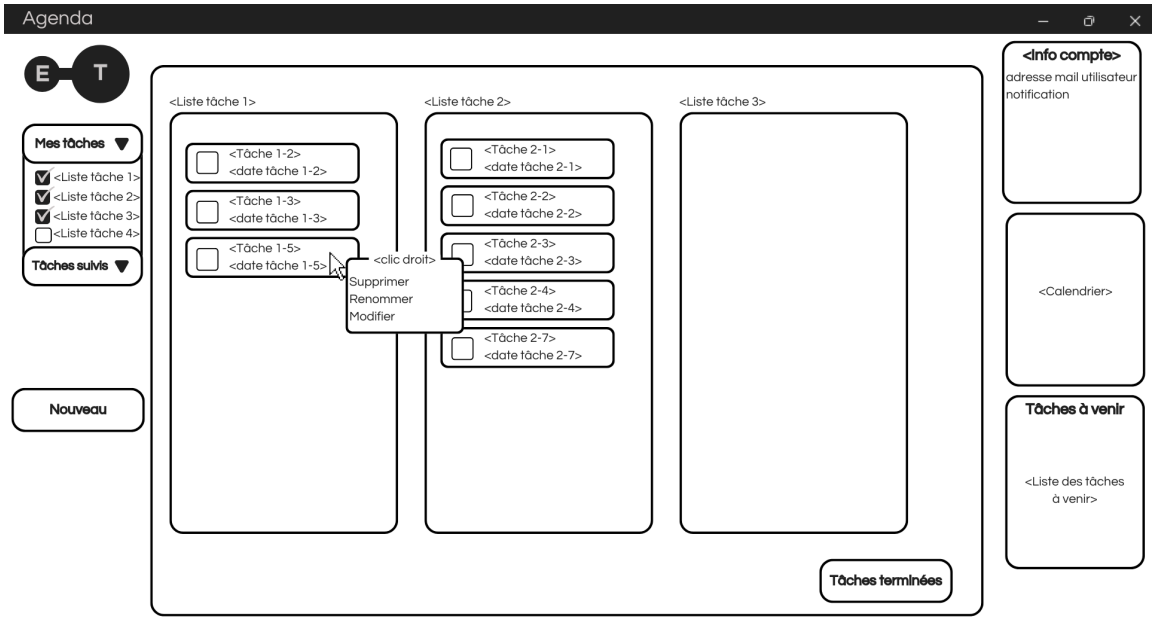

Figure 13: Clic droit sur une tâche

6 Axe Fonctionnel

6.1 Diagrammes des cas d'usages

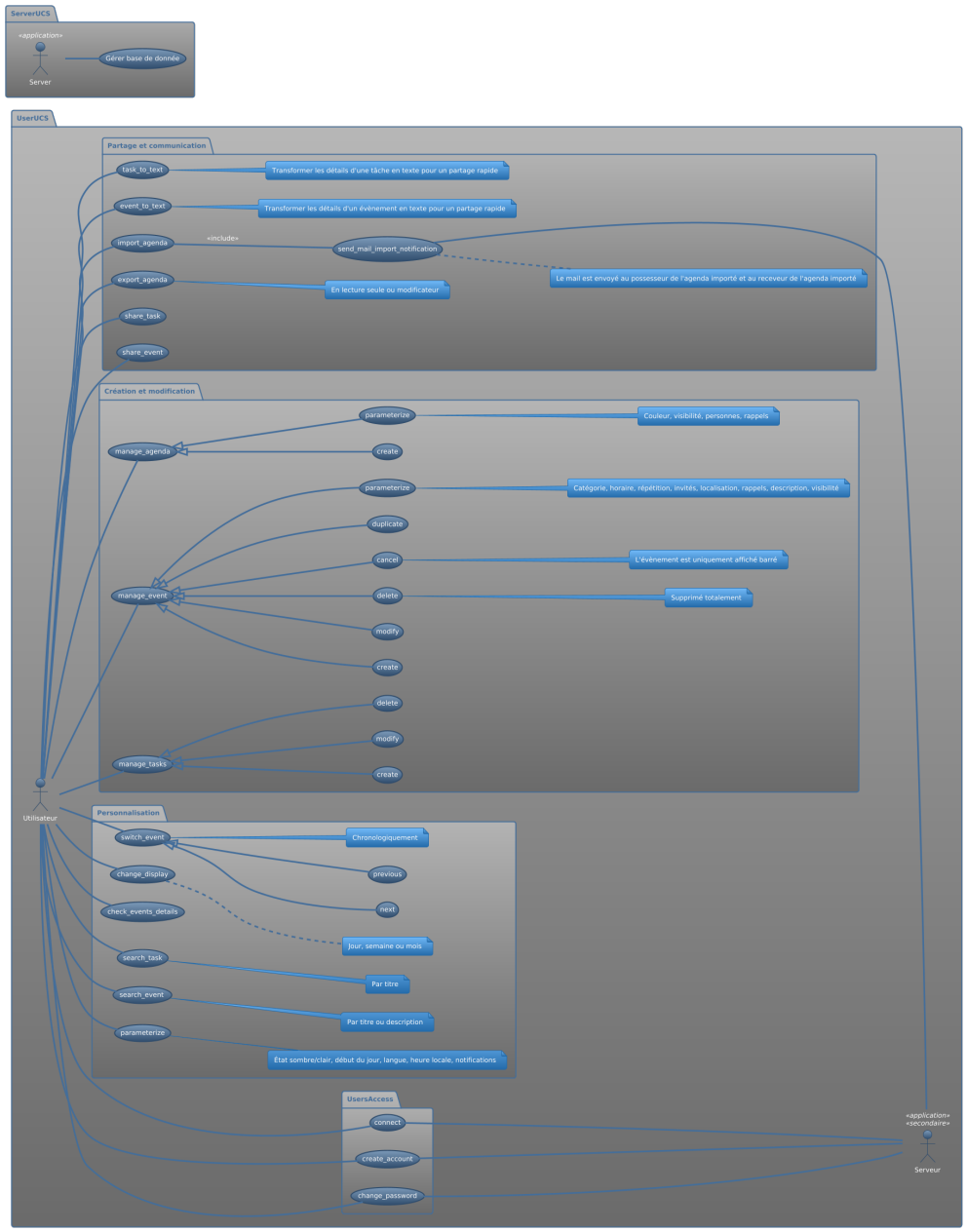

Figure 14: UC Diagram

6.2 Descriptions cas d'usage

6.2.1 Cas : Créer un Evenement

Précondition : L'utilisateur est connecté.
Scénario nominal :
3. L'utilisateur choisit l'agenda.
4. L'utilisateur personnalise la date de l'évènement.
5. Le système valide la date de l'évènement.
6. L'utilisateur personnalise l'heure de l'évènement.
7. Le système valide l'heure de l'évènement.
8. L'utilisateur personnalise la répétition de l'évènement.
9. Le système propose les répétitions possibles.
10. L'utilisateur ajoute les invités à cet évènement.
11. Le système valide les invités à cet évènement.
12. L'utilisateur indique une localisation à cet évènement.
13. Le système valide la localisation à cet évènement.
14. L'utilisateur indique les rappels à cet évènement.
15. Le système propose les rappels possibles.
16. L'utilisateur choisit ses rappels.
17. L'utilisateur rentre une description.
18. Le système valide la description.
19. L'utilisateur personnalise la visibilité de l'évènement.
20. Le système propose les visibilités possible.
21. L'utilisateur choisit les visibilités possible.

A1. Le système ne valide pas la date de l'évènement.
L'enchaînement A1 commence au point 5. du scénario nominal.
6. Le système indique à l'utilisateur que la date de l'évènement n'est pas conforme.
Le scénario nominal reprend au point 4.

A2. Le système ne valide pas l'heure de l'évènement.
L'enchaînement A2 commence au point 7. du scénario nominal.
8. Le système indique à l'utilisateur que la date de l'évènement n'est pas conforme.
Le scénario nominal reprend au point 6.

A3. Le système ne valide pas les invités de l'évènement.
L'enchaînement A3 commence au point 11. du scénario nominal.
12. Le système indique à l'utilisateur que l'utilisateur saisi n'existe pas dans le serveur.
Le scénario nominal reprend au point 10.

7 Axe Dynamique

7.1 Diagrammes de séquence

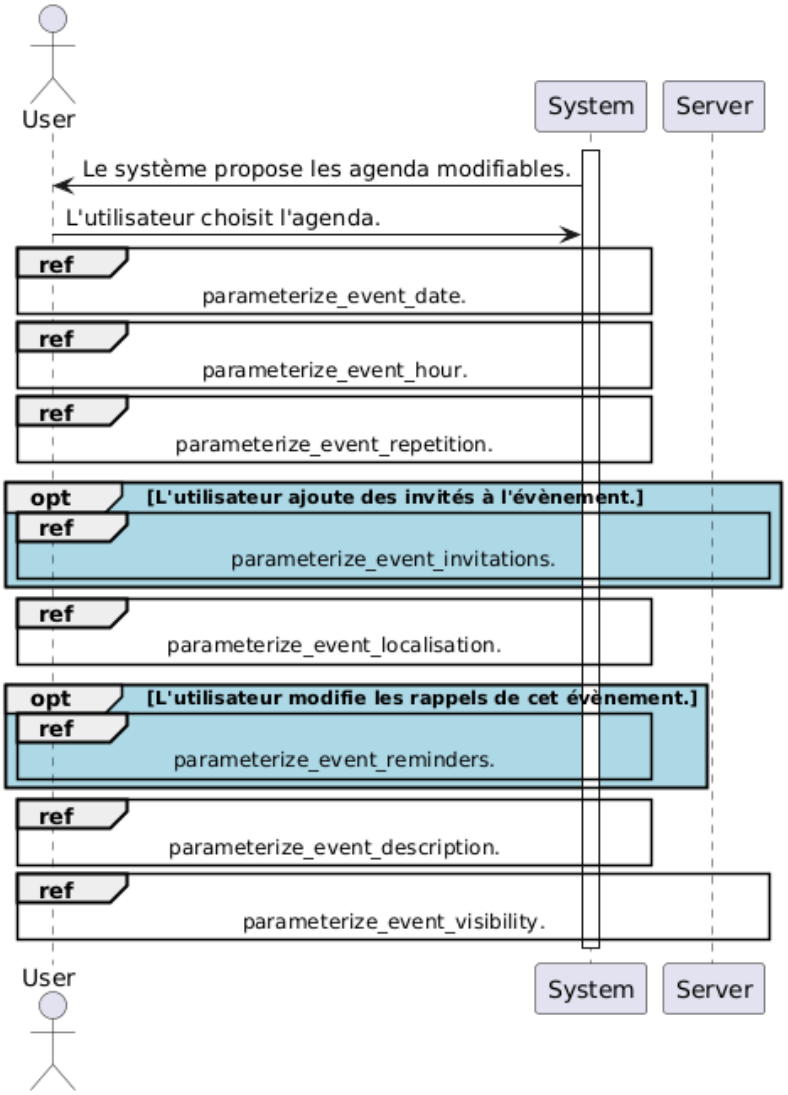

Figure 15: Create event seq

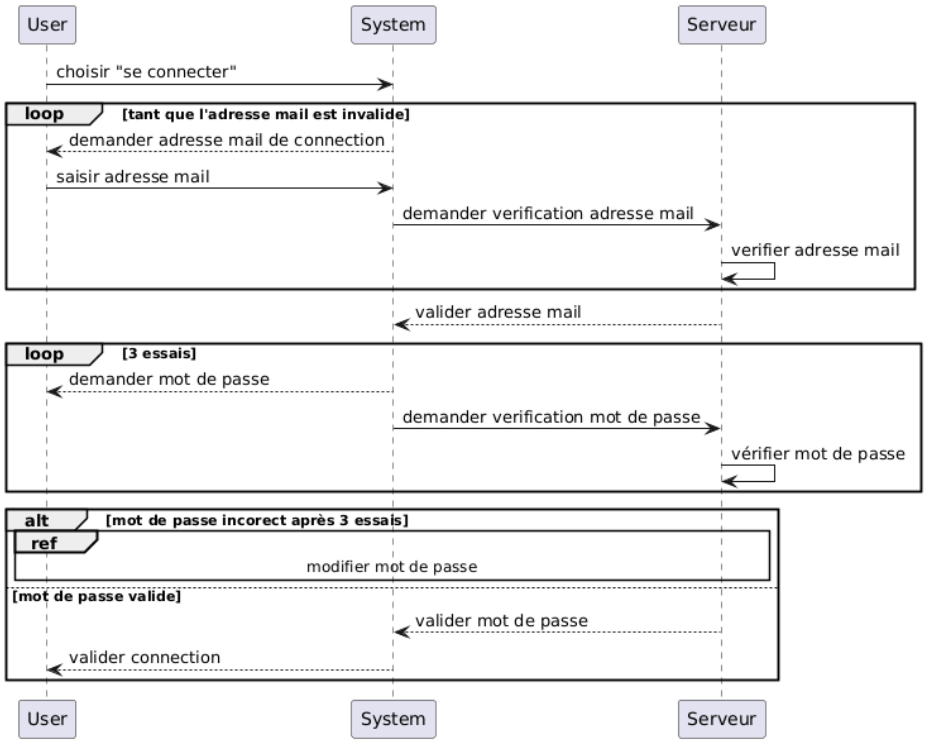

Figure 16: Connect seq

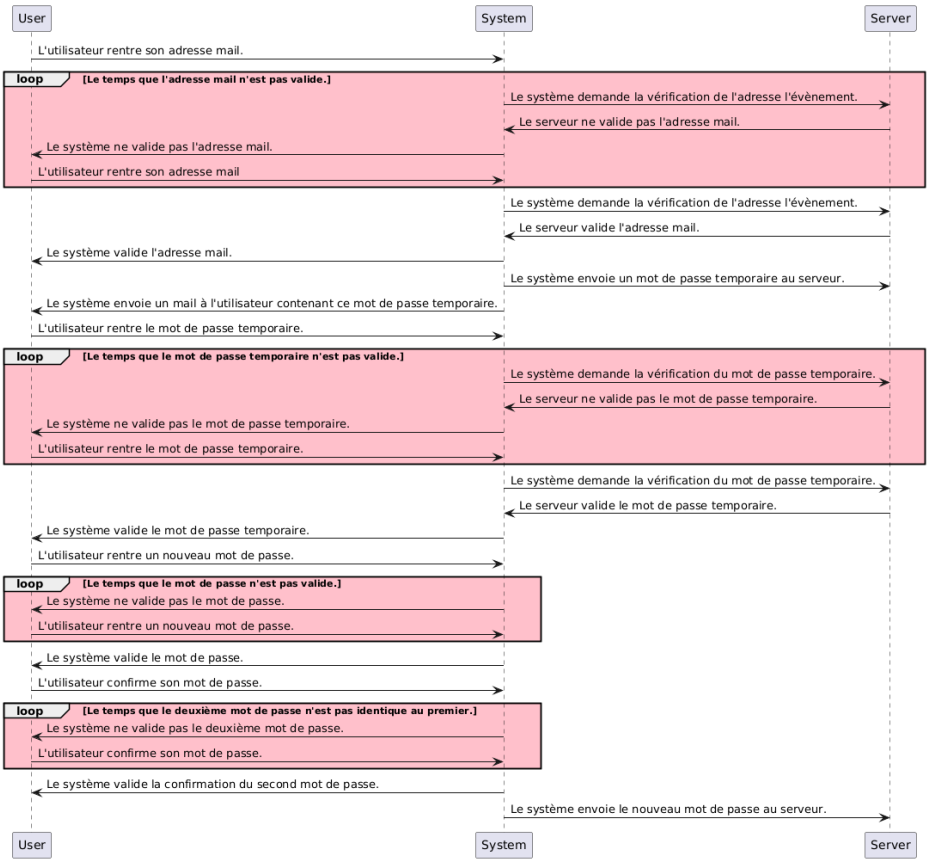

Figure 17: Change password seq

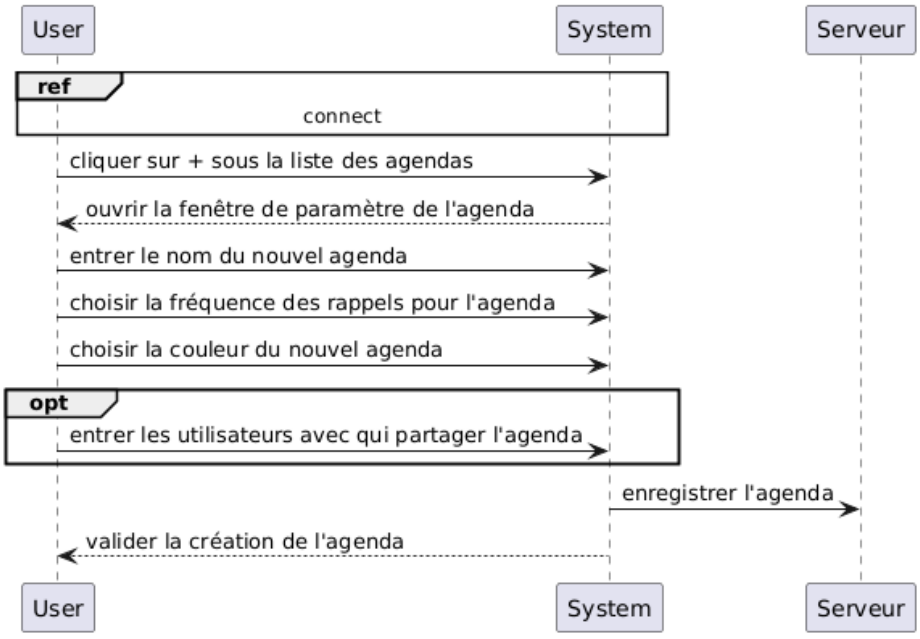

Figure 18: Create agenda seq

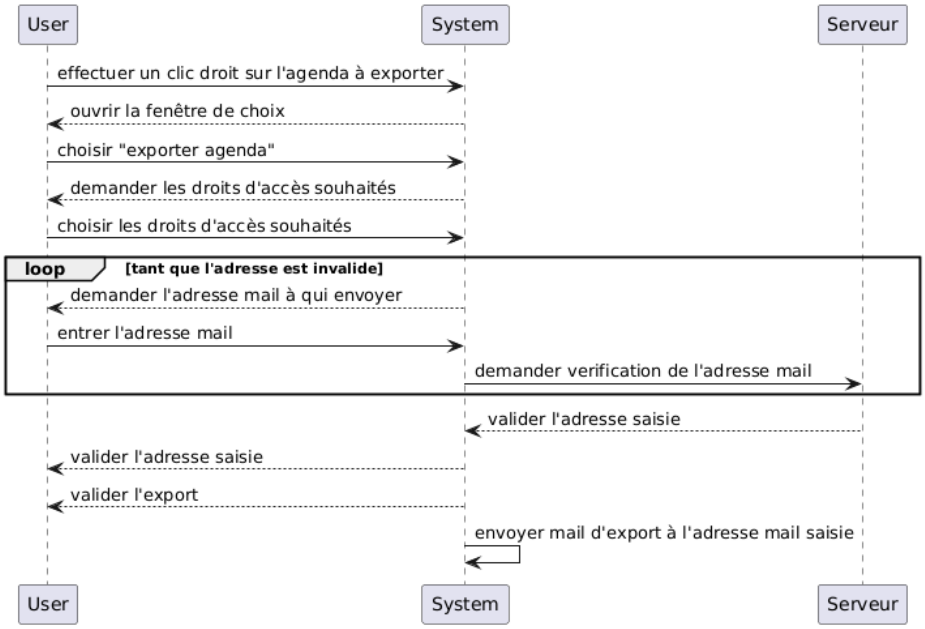

Figure 19: Export agenda seq

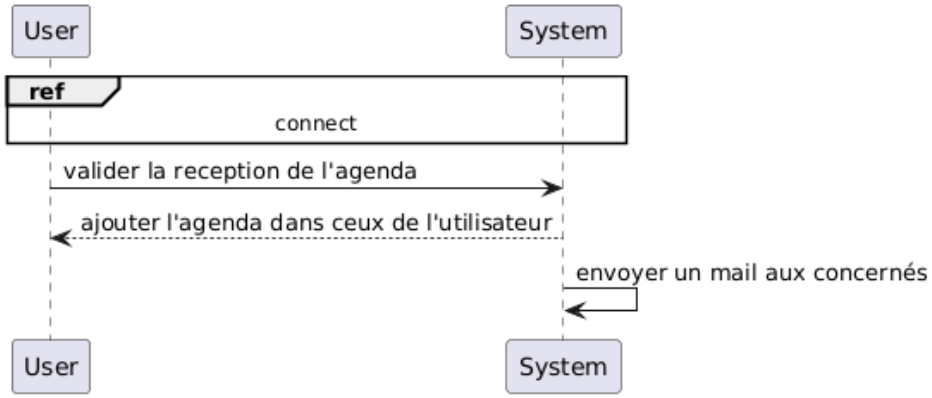

Figure 20: Import agenda seq

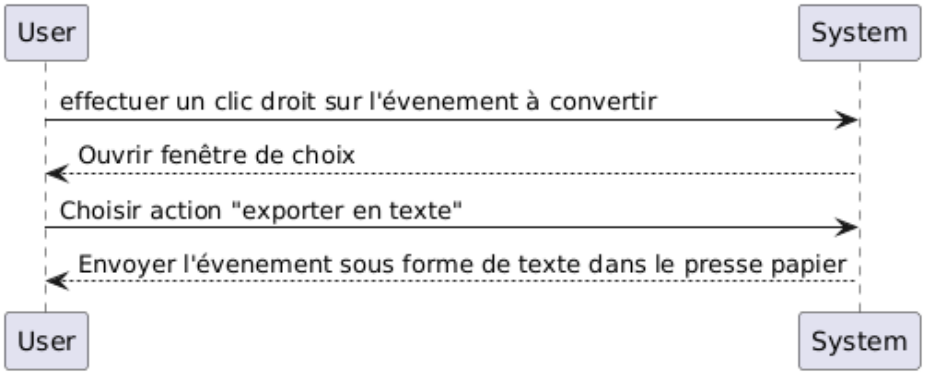

Figure 21: Event to text seq

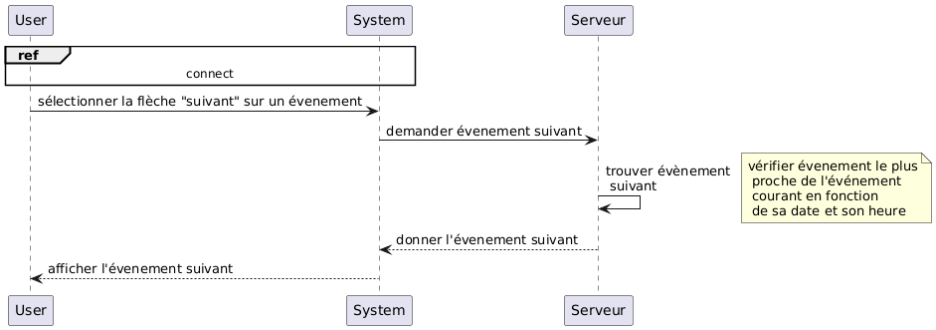

Figure 22: Switch event seq

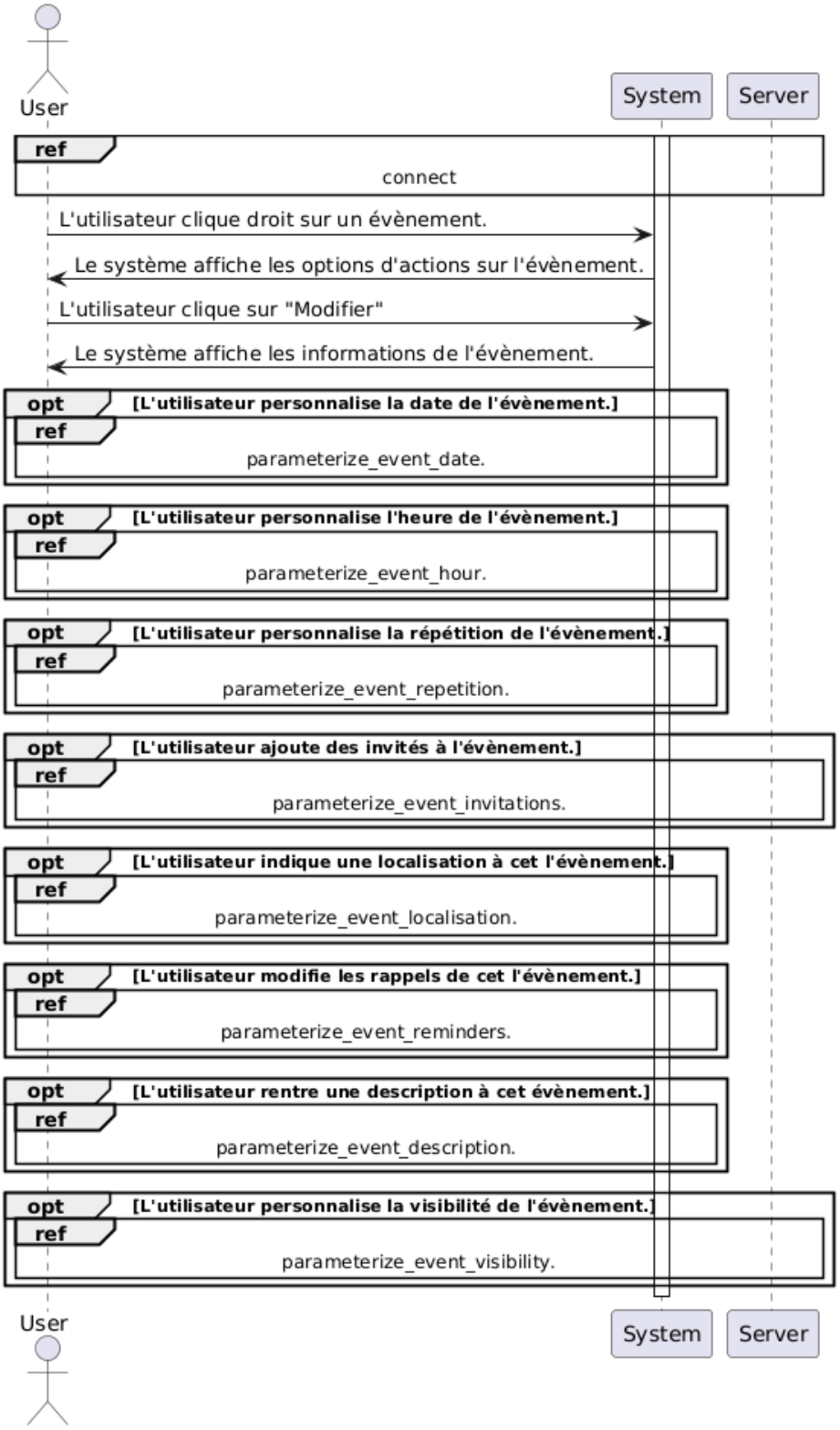

Figure 23: Parameterize event seq

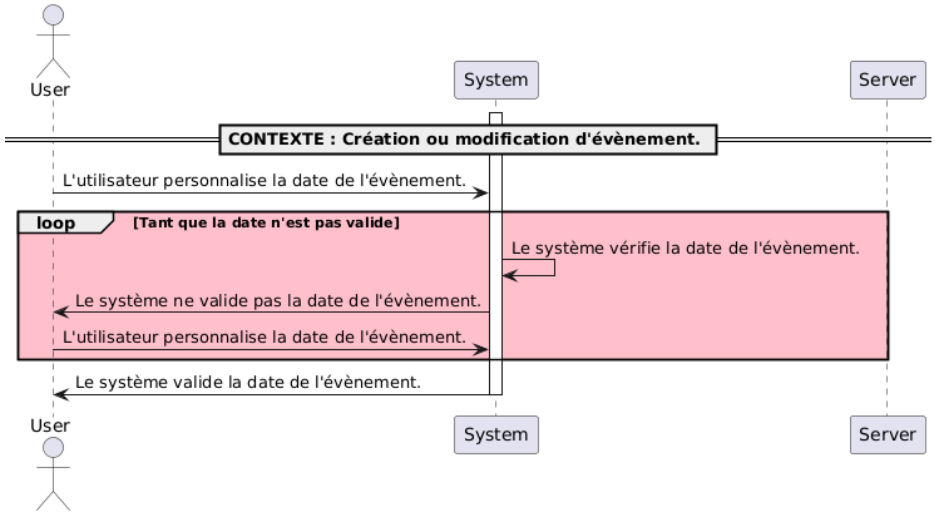

Figure 24: Parameterize date event seq

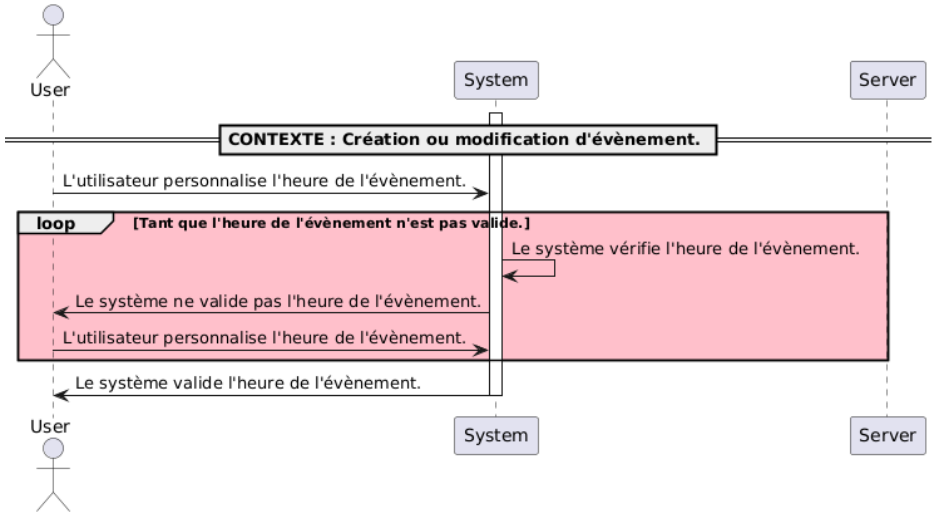

Figure 25: Parameterize hour event seq

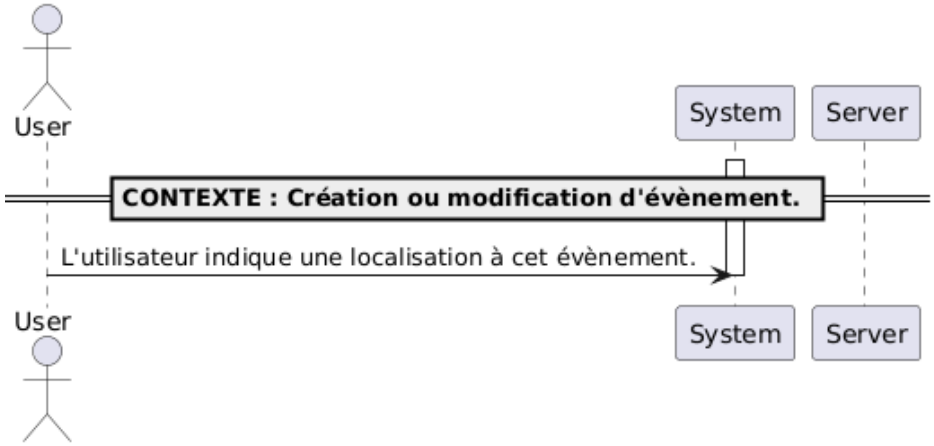

Figure 26: Parameterize location event seq

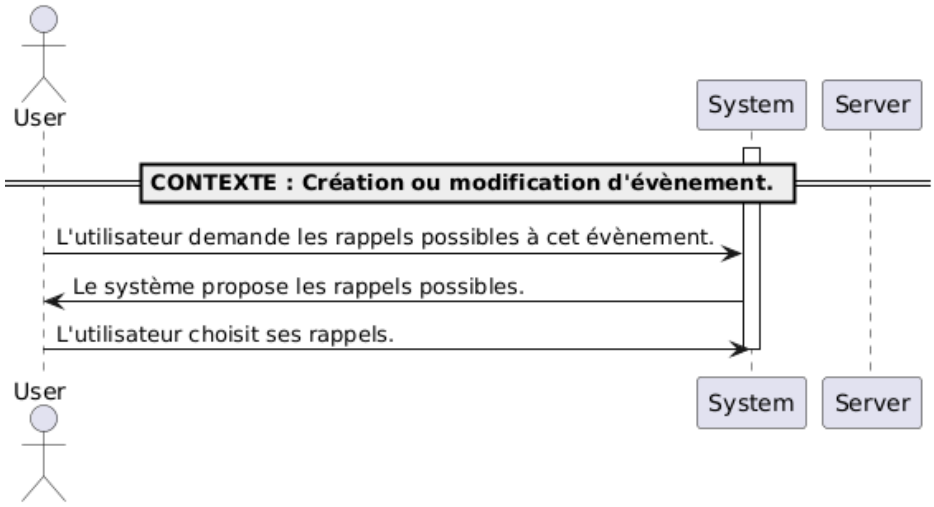

Figure 27: Parameterize reminder event seq

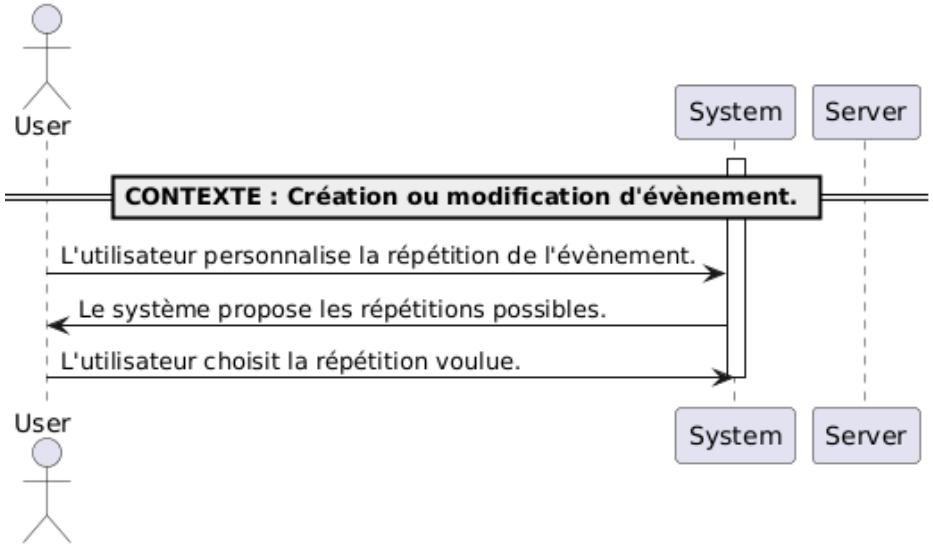

Figure 28: Parameterize repetition event seq

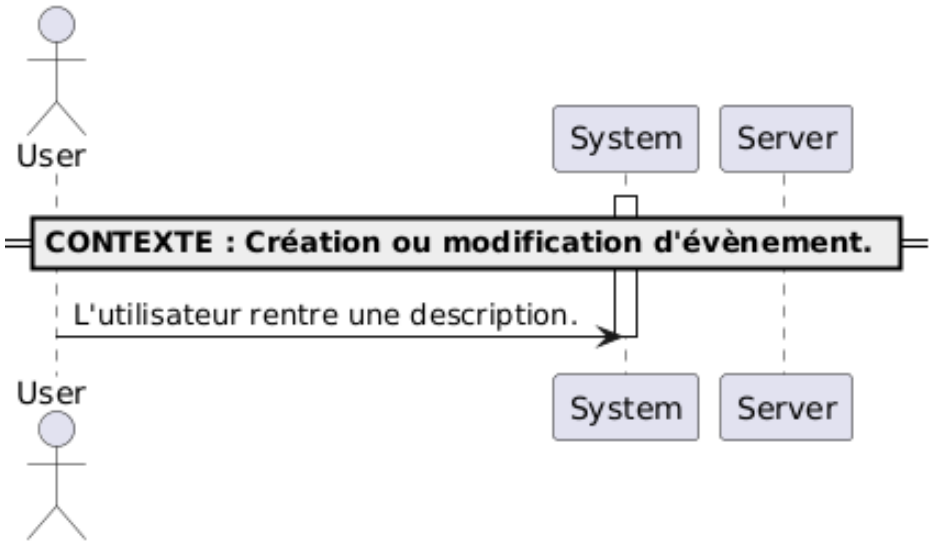

Figure 29: Parameterize description event seq

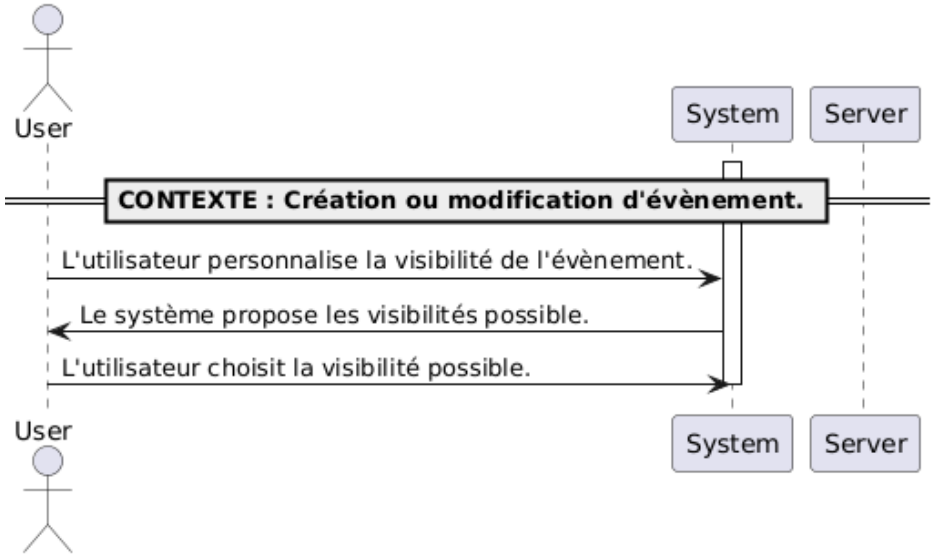

Figure 30: Parameterize visibility event seq

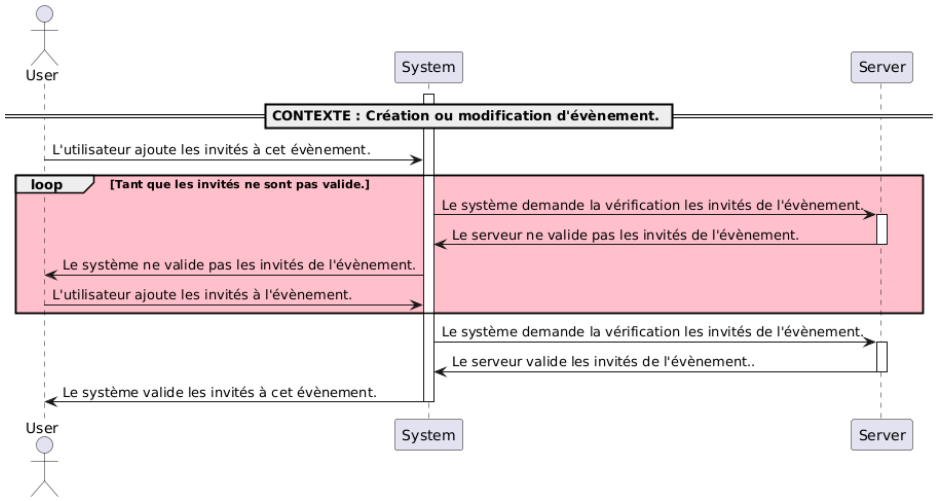

Figure 31: Parameterize invitation event seq

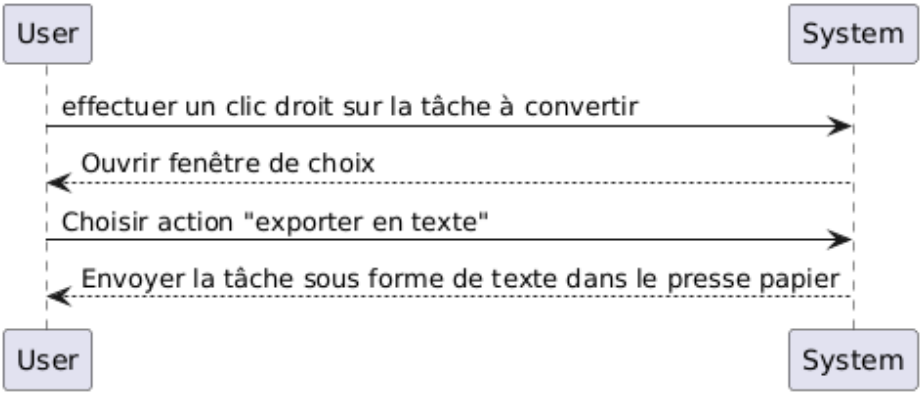

Figure 32: Task to text seq

8 Axe Statique

8.1 Diagramme de classes

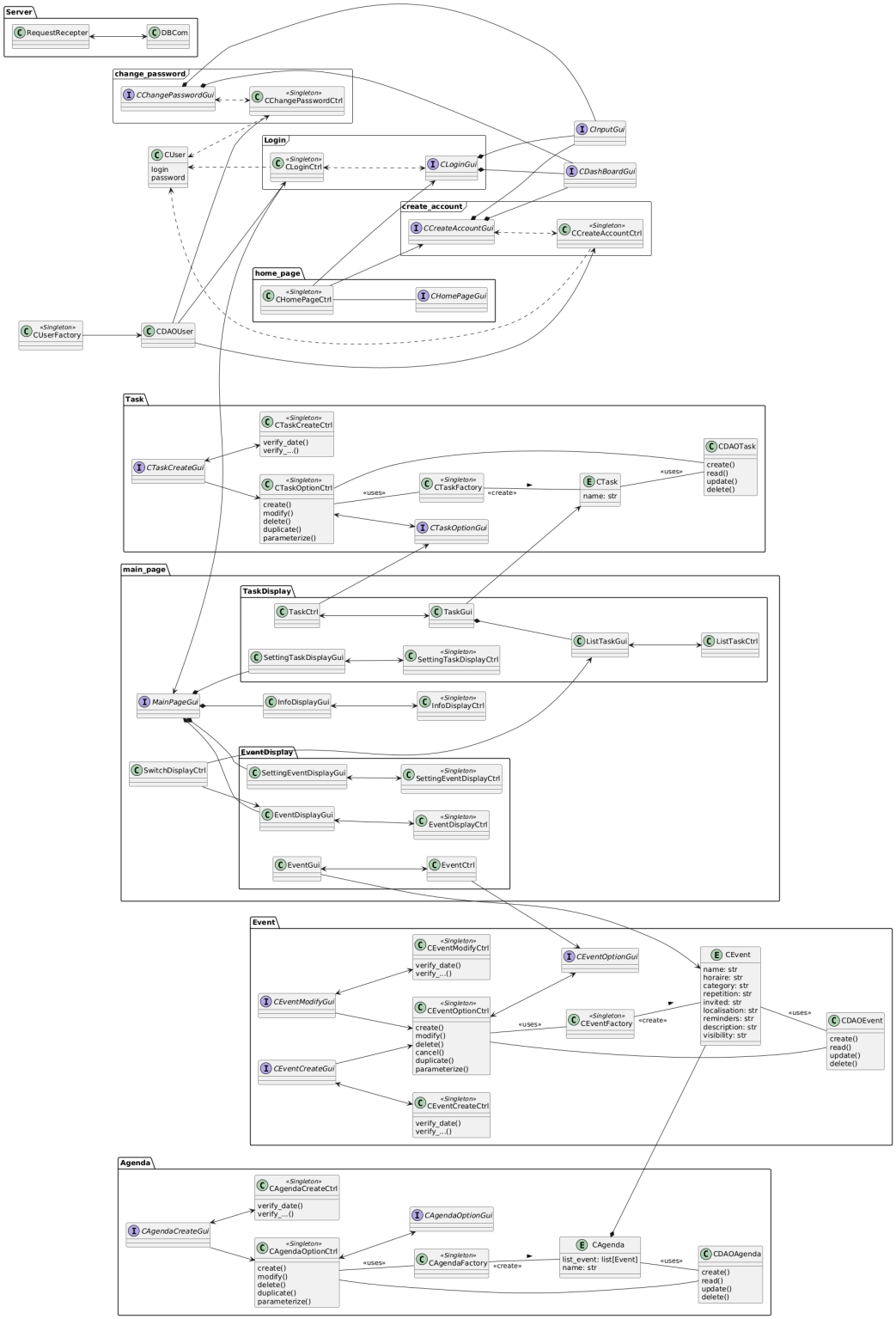

Figure 33: Class diagram

9 Phase de tests

Lors du développement du projet nous avons effectué nos test à chaque fois qu'une des fonctionnalité listées dans les exigence du logiciel était programmée.
D'abord le bon fonctionnement des actions sur l'interface graphique (callback sur les boutons, récupération des données dans les input), puis la gestion des erreur par rapport aux entrées possibles par l'utilisateur ainsi que leur signalement à celui-ci.
Nous avons ensuite associé la base de données à l'interface graphique puis testé le bon fonctionnement de cette association, en testant la communication entre les deux.

10 Interface graphique

10.1 Interface

Pour la conception de l'interface visuel nous avons utilisé le module PyQT et le logiciel QtDesigner.
Le fichier "software_ui.py" a été généré par QtDesigner et réadapté sur quelque points par nos soins.
Notamment l'ajout d'une méthode pour initialiser l'affichage des calendrier affichés par jours ou par semaines et le rajout d'un élément graphique non crée par QtDesigner pour afficher les tâches en cours ou terminées (Les classes "TaskOngoingDisplay" et "TaskFinishedDisplay" servent à créer ces éléments graphiques).

10.2 Contrôleurs

Pour ce qui est des contrôleurs, toutes les autres classes dans les dossiers "pages" et "menu" leur sont dédiées.
Les classes de "pages" servent à contrôler tous les éléments graphiques par page du logiciel : écran d'accueil, écran de connexion, écran de création de compte, affichage et gestion des événements, affichage et gestion des tâches.
Les classes dans "menu" servent à gérer les menus affichés lors du clic droit sur des éléments des pages, ou encore les pages de paramétrage des événements et des tâches, ainsi que leurs modifications.

11 Communication client-serveur

11.1 Serveur

Pour la communication entre le client et le serveur, nous avons utilisé le module socket. Pour la communication entre le serveur et la base de données, nous avons utilisé le module sqlite3. Le serveur est simplement en attente de connexion d'un client, et lance une boucle qui attend des messages de celui-ci après connexion. Les messages sont formatés en JSON, et le serveur les traite en fonction de leur type (appelé *op*).

Il y a 5 types d'opérations :

- **1**: création de compte
- **2**: connexion à un compte
- **3**: requête du client avec comme spécification la catégorie "requestType"
- **4**: retour serveur positif
- **5**: retour serveur négatif avec message d'erreur dans la catégorie "errmsg"

Il existe plusieurs types de requêtes comme :

- **"createAgenda"**: création d'un agenda
- **"updateAgenda"**: mise à jour d'un agenda
- **"getAgendaList"**: récupération de la liste des agendas pour un utilisateur
- **"deleteAgenda"**: suppression d'un agenda
- et bien d'autres...

Nous avons mis toutes ces requêtes sous la forme d'un dictionnaire, Dans celui-ci, la clef sera le nom de la requête et la valeur sera une Request.

```
1 class Request:
2     def __init__(self, query: str, func: Callable[[sqlite3.Cursor], str]):
3         self.query = query
4         self.func = func
5 
6     def send(self, *args) -> bytes:
7         return self.func(con.execute(self.query, args)).encode()
```

Nous pouvons donc voir que le constructeur de la class Request prend en paramètres

- **query**: la requête SQL à exécuter.
- **func**: la fonction qui va formater le résultat de la requête en une réponse JSON à envoyer au client.

Pour un exemple :

```
def createAgenda(cur: sqlite3.Cursor) -> str:
    return "{\"data\":{\"agenda_id\":" + str(cur.lastrowid) + "},\"op\":4}"

requestType_to_query: dict[str, Request] = {
    "createAgenda": Request(
            "insert into agenda (user_id, name) values (?, ?);",
            createAgenda
        )
}
```

Toutes les fonctions qui formatent les retours de requêtes sont dans *request_format.py*.
Pour l'utilisation de ce dictionnaire il suffit de faire :

```
# Si l'opération est la numéro 3
try:
    data: bytes =
        requestType_to_query[m_json["requestType"]].send(*m_json["data"].values())
except sqlite3.Error as e:
    conn.sendall(("{\"errmsg\":\"" + str(e) + "\",\"op\":5}").encode())
else:
    conn.sendall(data)
```

L'affreuse ligne

$$requestType\_to\_query[m\_json[\text{"}requestType\text{"}]].send(*m\_json[\text{"}data\text{"}].values())$$

est centrale dans l'automatisation des requêtes, elle permet d'en ajouter sans rajouter un nouveau elif.
Elle récupère les données dans "data", ne récupère que les valeurs via *.values*(), transforme cela via $*$ en suite d'arguments (donc ce n'est plus une liste), et envoie tout dans la fonction *send* de la classe Request.
Il est donc un peu contreintuitif, mais obligatoire à ce que les requêtes côté client envoient bien les données dans l'ordre des ? des requêtes dans *requestType_to_query*.

11.2 Client

Du côté du client il y a simplement une classe qui gère la communication avec le serveur (DBCom) et plusieurs DAO qui transcrivent les types requêtes en méthode. Voici deux exemples, un assesseur et un mutateur:

```
class AgendaDAO:
    def __init__(self, s: DBCom):
        self.dbcom = s

    def delete(self, agenda: Agenda):
        self.dbcom.sendall({
            "data":{
                "agenda_id": agenda.id
            },
            "requestType": "deleteAgenda",
            "op": 3
        })
        return self.dbcom.recv()

    def get_list(self, user: User) -> list[Agenda]:
        self.dbcom.sendall({
            "data": {
                "user_id": user.id
            },
            "requestType": "getAgendaList",
            "op": 3
        })
        r: list[Agenda] = []
        data: dict = self.dbcom.recv()
        for agenda in data["data"]["agendaList"]:
            r.append(Agenda(agenda["id"], agenda["name"]))

        return r
```

Voici le modèle entité association de notre base de donnée :

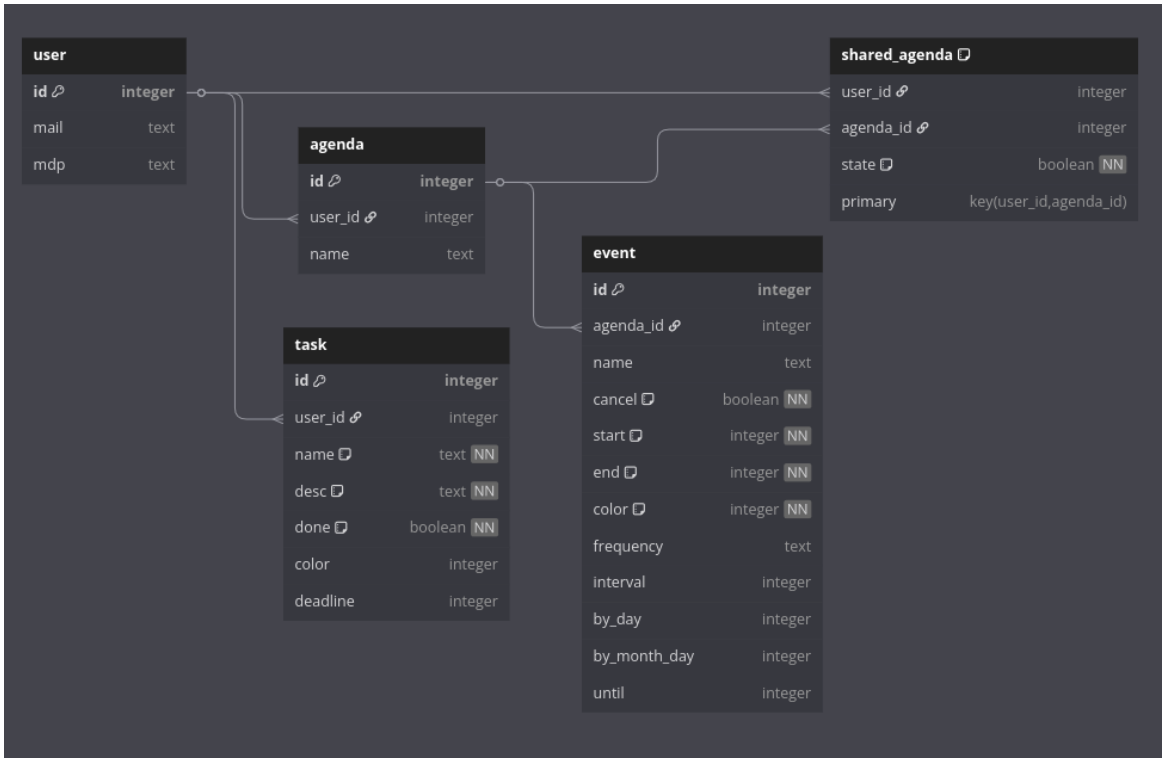

Figure 34: MEA