
TB

QUENTIN GIGON

HEIG-VD

Sections and Chapitres

Quentin Gigon

Table des matières

1	Introduction au projet	3
2	Cahier des charges	3
2.1	Contraintes et besoins	3
2.2	Fonctionnalités	3
2.3	Echéancier	4
3	Présentation des technologies utilisées	5
3.1	Framework Play!	5
3.2	EventSource	5
3.3	PostgreSQL	5
3.4	JPA	5
3.5	JWT	5
3.6	HTML5	5
4	Schéma de base de donnée	6
5	Analyse et Architecture	7
5.1	Site	7
5.2	Team	7
5.3	Organisation des flux	7
5.3.1	Flux	7
5.3.2	Schedules	8
5.3.3	Diffuser	8
5.4	Ecrans	9
5.4.1	Affichage	9
5.4.2	Events	9
5.4.3	Authentification	9
5.4.4	Groupe d'écrans	10
6	Protocole de communication écran-serveur	11
6.1	Ecran inconnu du serveur	11
6.2	Ecran connu du serveur	12
6.3	Ecran connu - erreurs de connexion	13
7	Réalisation technique	14
7.1	Organisation des flux	14
7.1.1	Bloc-horaire	14
7.1.2	RunningScheduleThread	15
7.1.3	FluxManager	17
7.2	Contrôleurs	19
7.2.1	EventSource	19
7.2.2	Ecrans	21
7.2.3	Schedules	21
7.2.4	Diffusers	22
7.3	Vues	23
7.3.1	Affichage des flux	23
7.3.2	Echange de données	24

8 Cas d'utilisation	26
8.1 Administrateur :	26
8.2 TeamAdmin :	28
8.3 TeamMember :	29
9 Mockups	31
9.1 Utilisateurs	31
9.2 Ecrans	32
9.3 Teams	33
9.4 Flux	34
9.5 Schedules	35
9.6 Diffusers	36
10 Tests	36
10.1 Tests JUnit	36
10.2 Tests fonctionnels	36
11 Remarques personnelles et commentaires	36

1 Introduction au projet

Le but de ce travail est de pouvoir visualiser et gérer l'affichage de divers flux d'information sur les écrans des différents campus de la HEIG, comme par exemple les horaires de cours, les news de la RTS, les réservations de salles, etc, à travers une interface web.

2 Cahier des charges

A remettre à jour!!

2.1 Contraintes et besoins

Les besoins principaux de cette application sont les suivants :

- Gestion de l'affichage de flux d'information sur des écrans dans la HEIG-VD (smartTV ou ordinateur) par une interface web.
- Protocole concis de communication entre les écrans et le serveur limitant les échanges.
- Affichage de flux "contrôlés" (générés par l'application, par exemple un flux RSS) et "non-contrôlés" (flux de la RTS, horaires des cours, etc).
- Un Schedule qui s'occupe de changer les flux affichés selon un horaire prédéfini.
- Une modélisation générique des flux et une factorisation de ceux venant de sources externes afin de pouvoir les envoyer de la même manière aux écrans
- La possibilité de diffuser plusieurs types de médias (images, vidéos, etc)
- La possibilité de passer outre le Schedule et d'afficher un flux voulu (annonce importante, etc) avec reprise de l'exécution prévue par la suite.

Quelques précisions : Un Schedule est un "horaire" qui gère le changement de flux selon un ordre choisi par l'utilisateur. Il n'y aura pas qu'un seul Schedule mais plutôt la possibilité d'en créer plusieurs qui soient assignables à un écran ou groupe d'écrans.

Les contraintes principales quand à elles sont les suivantes :

- L'affichage des flux doit être fait dans un navigateur supportant le Javascript, HTML5 et CSS3.
- Il doit y avoir une base de données qui enregistre les utilisateurs ainsi que les écrans.
- Il y a plusieurs types d'utilisateurs qui, selon leur emplacement (campus) et/ou leur niveau d'autorisation, peuvent modifier l'affichage des écrans.
- Le système doit être tolérant face aux panne, avec une reprise automatique.
- Le système doit disposer d'une interface simple et être utilisable par des gens du domaine et par des personnes non-initiées.

2.2 Fonctionnalités

Les fonctionnalités nécessaires et principales du programme sont divisées en plusieurs catégories :

— Frontend

1. Interface de login et register sur le site (register dans le cadre du TB)
2. Ecrans
 - (a) Visualisation des écrans actifs et de leur emplacement sur le campus (une "carte" par site)
 - (b) Visualisation des informations d'un écran spécifique (et groupe d'écrans)
 - (c) Modification de la diffusion actuelle sur un écran/groupe d'écrans (flux "hors-runningSchedule", attribution à un Schedule, arrêt de la diffusion, ...)
3. Flux
 - (a) Opérations CRUD sur les flux (interface de création)
 - (b) Visualisation des flux utilisables par le système et infos sur leur contenu
4. Schedules
 - (a) Opérations CRUD sur les Schedules (interface de création)
 - (b) Visualisation des Schedules utilisables par le système et infos sur leur contenu et horaire

— Backend - Play!

1. Ecrans

- (a) Opérations CRUD sur les écrans
 - (b) Emission de token d'authentification pour la connexion des écrans
- 2. Flux
 - (a) Opérations CRUD sur les flux
 - (b) Diffusion de flux aux écrans selon un Schedule
 - (c) Diffusion de flux hors-runningSchedule (annonces, alertes, etc)
 - (d) Formatage et mise en page des flux externes (RTS ou autre)
- 3. Schedules
 - (a) Opérations CRUD sur les Schedules
 - (b) Assignation d'un Schedule à un écran/groupe d'écrans
- 4. Utilisateurs
 - (a) Register
 - (b) Login
 - (c) Niveaux d'autorisation
- **Base de données**
 - 1. Utilisateurs avec différents niveaux d'autorisations
 - 2. Ecrans, avec leurs caractéristiques et emplacement
 - 3. Flux utilisés par le système
 - 4. Schedules de flux

Les fonctionnalités suivantes sont considérées comme secondaires et seront réalisées si le temps le permet :

- **Frontend**
 - 1. Modification en live du contenu d'un flux
- **Backend**
 - 1. Monitoring de l'état des écrans (log du dernier échange avec l'écran)
 - 2. Schedul-ception (Schedule dans un Schedule)

2.3 Echancier

Le travail sera divisé en 4 parties :

- **Analyse et Modélisation** (2 semaines)
Analyse des contraintes et besoins du travail et modélisation d'un système parvenant à y répondre (schémas de DB, modélisation des flux, etc)
- **Architecture** (1 semaine)
Création d'une architecture de code permettant la réalisation d'un programme efficace, modulable et améliorable par la suite
- **Développement et Tests** (7 semaines)
Codage de l'application, en commençant par le serveur et en finissant avec le frontend. Création de la base de données en parallèle. Tests unitaires et fonctionnels (60-70% de coverage visé)
- **Rapport et Documentation** (3 semaines)

Total : env. 13 semaines à partir du 25.02

3 Présentation des technologies utilisées

3.1 Framework Play!

Play! est un framework web open-source qui suit le modèle MVC et qui permet d'écrire rapidement des application web en Java (ou en Scala). A la différence d'autre frameworks Java, Play! est *stateless*, ce qui veut dire qu'il n'y a pas de session JavaEE créée à chaque connexion. Il fourni aussi à ses utilisateurs des frameworks de tests unitaires et fonctionnels, à savoir JUnit et Selenium.

3.2 EventSource

Les EventSource, ou Server-Sent Events (SSE), sont une technologie permettant à un navigateur internet de recevoir des mises à jour automatiques d'un serveur par une connexion HTTP persistante. L'API Javascript (*Server-Sent Events EventSource API*) fut instaurée la première fois dans Opera en 2006 et a été normalisée dans le cadre de HTML5.

Les Events envoyés sont au format *text/event-stream* et sont reçus par le navigateur sous la forme d'Event de type *message*. La connexion reste ouverte tant qu'elle n'a pas été fermée par le navigateur, et contrairement aux WebSockets, les SSE sont uni-directionnels et ne permettent donc pas aux clients de communiquer avec le serveur.

3.3 PostgreSQL

PostgreSQL est un système de gestion de base de données relationnelle et open-source. La différence principale entre PostgreSQL et ses concurrents est la prise en charge de plus de types de donnée que les types traditionnels (entiers, caractères, ...).

3.4 JPA

La Java Persistence API (JPA) est une interface de programmation permettant aux utilisateurs de la plateforme Java (SE et EE) d'organiser facilement et clairement leurs données relationnelles. Elle utilise des annotations pour définir des "objet-métiers" qui serviront d'interface entre la base de données et l'application.

JPA définit aussi le Java Persistence Query Language (JPQL), qui est utilisé pour créer les requêtes SQL dans le cadre de JPA. Les requêtes effectuées dans ce langage ressemblent beaucoup à du SQL classique, sauf que le JPQL fonctionne avec des entités (créées avec des annotations) plutôt que des tables de la base de données.

3.5 JWT

Un JSON Web Token est un standard ouvert permettant l'échange sécurisé de jetons d'accès entre plusieurs parties. Les jetons sont signés à l'aide d'une clé privée (très souvent le serveur) et fournis aux client qui doivent l'intégrer à leurs prochaines requêtes. Un jeton n'est pas associé à un utilisateur mais plutôt à un rôle (p.ex. administrateur) et à une durée de validité définie par le serveur également. Ils sont très souvent utilisés pour gérer des utilisateurs (login).

3.6 HTML5

HTML5 est la dernière version majeure du HTML (octobre 2014). Elle vient avec plein de nouveaux élément, comme la balise *video*, qui permet d'insérer un contenu vidéo en streaming dans un fichier HTML, ou encore *footer*, qui lui permet de facilement afficher du texte en bas de page.

4 Schéma de base de donnée

Une des directives principales du projet était la représentation en tout temps de l'état actuel du programme en base de données. Le schéma a donc été pensé pour répondre à cette demande. Les limitations voulues pour les différents rôles des utilisateurs et pour les services de la HEIG (COM, Secrétariat, Baleinev, etc) sont toutes représentées en BD, comme par exemple un PhysicalScreen est relié à un site (ce qui limite les flux qu'il peut afficher) et il est également relié à une ou plusieurs Teams (autorise uniquement les membres de cette Team à interagir avec cet écran).
TODO : explications du schéma une fois celui-ci définitif (ou presque).

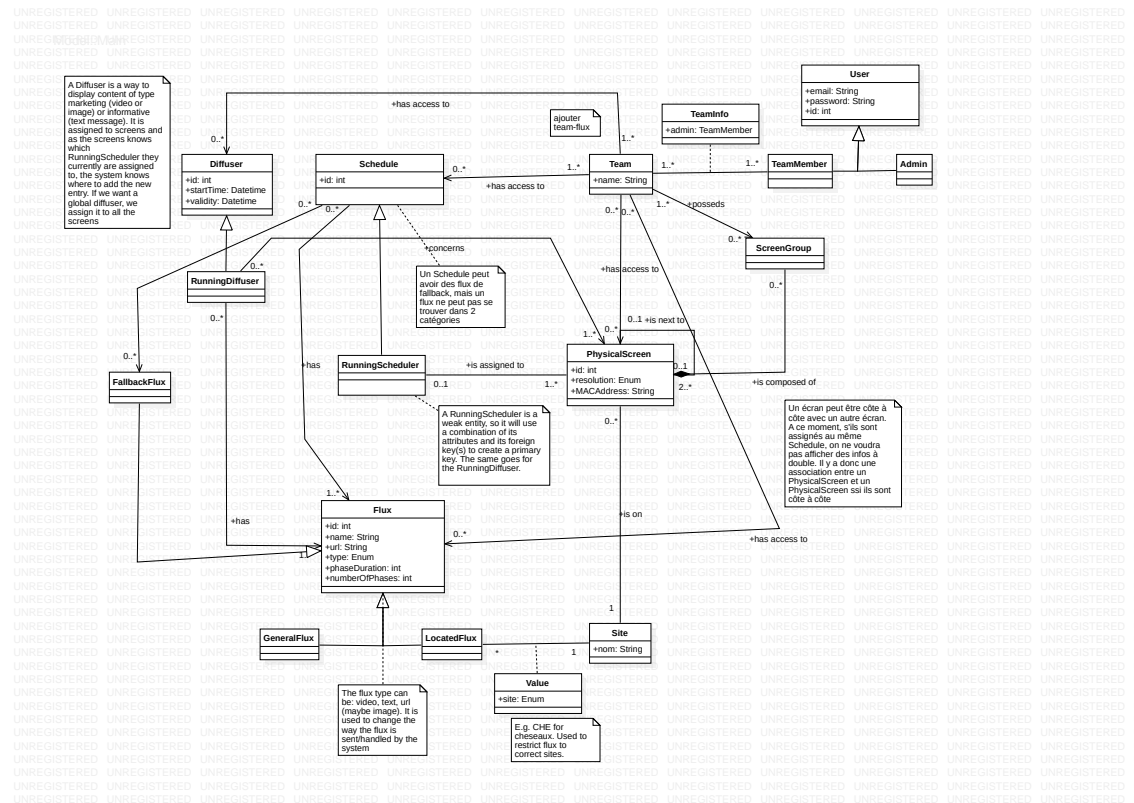


FIGURE 1 – Schéma de base de données

5 Analyse et Architecture

Dans les chapitres suivants seront détaillés l'architecture du programme, ainsi que les différentes idées et versions qui ont été envisagées.

5.1 Site

Un site représente un emplacement physique de l'HEIG-VD, donc les sites de Cheseaux, St-Roch et Y-Park. Ils servent principalement à localiser les écrans et restreindre l'affichage de flux selon le lieu.

5.2 Team

Pendant la phase d'analyse, il a été spécifié qu'un système d'équipe était nécessaire, afin de restreindre les fonctions du programme selon l'appartenance de l'utilisateur à telle ou telle équipe. Cela faisait également sens vu que le programme serait utilisé par les différents départements de la HEIG-VD.

Les **Teams** ont donc une place centrale dans l'architecture du programme dans le sens que les actions proposées à l'utilisateur utilisent uniquement les données accessibles par son équipe. Elle est composée de membres et de chefs d'équipe, qui sont les deux représentés par des **TeamMember**, et qui ont différents niveaux d'accès.

5.3 Organisation des flux

Un flux dans ce programme représente un contenu à afficher sur un écran. Ce contenu peut avoir des origines différentes, par exemple un flux externe à l'application comme le flux de la RTS ou un flux interne, comme les horaires des cours ou les réservations de salle.

Lors de la phase d'analyse, il a été spécifié que le programme devrait pouvoir gérer au moins deux catégories de flux :

- Des flux externes (p.ex. flux de la RTS)
- Des flux internes, avec la possibilité de les interroger pour savoir s'ils ont du contenu à afficher (réservation de salle, élection du CoRe)

J'avais également proposé de fournir un template RSS permettant un ajout facile de nouveau flux RSS, mais cette fonctionnalité a été abandonnée par manque de temps.

En ce qui concerne leur diffusion, il fallait proposer un moyen d'organiser un horaire de flux ainsi que la possibilité d'envoyer immédiatement un flux sur un écran. Deux types d'objets ont été créés pour cela : des Schedules (horaires), qui représentent l'horaire d'une journée et des Diffusers, qui eux permettent d'ajouter une entrée sur l'horaire d'un ou plusieurs écrans.

5.3.1 Flux

Un flux est caractérisé par un nom, type, une durée d'affichage et un contenu. Ce contenu est défini par le type du flux ; un flux 'URL' contiendra une url, tandis un flux de type 'Image' contiendra l'adresse de l'image sur le serveur.

Ils sont regroupés en quatre types :

- URL, ou type standard
- Vidéo
- Image
- Texte

Le traitement de ces flux par le système et la manière par laquelle ils sont affichés sur les écrans changent selon leur type. Par exemple, un flux 'Image' sera rendu dans une balise , tandis qu'un flux 'URL' le sera dans une "iframe".

De plus, il y a trois sortes de flux : des flux généraux, des flux localisés et des flux de fallback. Ces trois groupes sont à voir comme des sous-populations de flux : ils ont une référence vers un flux.

Comme son nom l'indique, un flux général peut être diffusé partout. Un flux localisé est lui uniquement affichable sur le Site correspondant. On peut donc par exemple avoir un seul flux pour les horaires de cours et plusieurs flux localisés qui le référence, en fournissant selon le site où il est envoyé un paramètre de requête différent (-> <http://heig.com/horaires?site=che>). Cette information est stockée dans la base de donnée sous la forme d'une table intermédiaire entre Site et LocatedFlux.

Et enfin un flux peut être un flux de fallback. Ces flux sont spécifiés à la création d'un Schedule et utilisés par celui-ci pour remplacer un flux programmé mais pour lequel le serveur ne détecte aucune

données à afficher. Ces flux peuvent être généraux ou localisés.

5.3.2 Schedules

Un Schedule représente un horaire de flux. Un Schedule n'est pas assigné à des écrans à sa création mais au moment de son activation pour permettre une réutilisation plus facile des Schedules créés. Chaque chef d'équipe peut en créer qui seront utilisables par tous les membres de son équipe.

La première version de cet horaire modélisait un cycle, où les flux choisis par l'utilisateur bouclaient à l'infini. Ce système a été utilisé avec succès pour de la recherche et des tests sur la faisabilité du programme mais fut vite obsolète quand il s'agissait d'avoir plus de contrôle sur l'horaire, par exemple définir une heure de début pour un flux donné.

Il a donc fallu inventer un système permettant à la fois de créer un horaire à respecter pour les Schedules et de modifier cet horaire à la volée pour les Diffusers, tout en garantissant la cohérence du programme.

J'ai choisi de représenter une journée complète d'affichage par des blocs de 1 minute chacun. Cette plage d'affichage est bornée par une heure de départ (inclusive) et une heure de fin (exclusive) qui sont fixes, mais modifiables si besoin. A titre d'exemple, en prenant les valeurs que j'ai défini, soit de 8h à 22h, on obtient une plage horaire de 15h, ce qui est équivalent à 900 blocs. Si on regarde maintenant pas-à-pas la procédure de création puis d'activation d'un Schedule :

Quand un utilisateur crée un Schedule, il peut lui spécifier plusieurs choses :

- Un nom (unique)
- Des flux avec heure de début (ScheduledFlux)
- Des flux sans heure de début
- Des flux de fallback
- Garder l'ordre des flux ou non

Pour chaque flux avec une heure de début associée, une "entrée de calendrier", ou ScheduledFlux, est créée. Cette "entrée" contient des références vers le Flux et le Schedule concerné, ainsi que le numéro de son bloc de départ.

Chaque flux sans heure de début est simplement ajouté au Schedule, pareil pour les flux de fallback. L'option de garder l'ordre des flux permet de garantir que les flux sans heure fixe seront affichés dans l'ordre dans lequel l'utilisateur les a rentrés. Notre Schedule est maintenant créé, il ne reste plus qu'à l'activer.

A l'activation d'un Schedule, on peut choisir parmi les écrans auxquels on a accès ceux qui seront concernés. Un objet RunningSchedule est créé, une entité temporaire qui existe uniquement tant que le Schedule reste actif. Cette entité est ensuite utilisée pour créer un "bloc-horaire", qui associe avec chaque bloc le flux correspondant (en regardant parmi les ScheduledFlux) ou une absence de flux. Un Runnable (RunningScheduleThread) est ensuite lancé avec cet horaire. La manière dont il choisit le prochain flux est détaillée dans la section correspondante.

Ce système permet de répondre à une autre demande du projet, la reprise de l'exécution des Schedules actifs lors du redémarrage du serveur ou après une maintenance. En donnant un moyen de faire correspondre une heure donnée avec un numéro de bloc, on peut très facilement reprendre l'exécution là où elle s'était arrêtée.

5.3.3 Diffuser

Un Diffuser est quand à lui utilisé pour diffuser un flux de catégorie marketing ou un message unique sur une quantité X d'écrans. Ils fonctionnent en changeant les Schedules actifs associés à ces écrans pour ajouter le flux souhaité dans leur horaire. Ce système offre plusieurs possibilités de customisation, comme spécifier la durée du flux ou limiter dans le temps la diffusion de ce flux (p.ex. modifier les Schedules actifs seulement pendant une semaine et revenir à l'état de base après). Au moment de l'ajout du flux dans le Schedule, des vérifications sont faites pour garantir un bon fonctionnement par la suite (on peut imaginer que l'heure choisie pour le nouveau flux soit déjà prise, il faudra alors avertir l'utilisateur qu'il ne peut effectuer cette action ou alors autoriser l'action mais adapter l'heure de démarrage du nouveau flux pour ce Schedule).

Comme pour les Schedules, un Diffuser actif est un RunningDiffuser, lui aussi une entité temporaire qui reste active selon la validité spécifiée à sa création. Un fois que le RunningDiffuser est arrivé en fin de vie, le flux qu'il avait rajouté dans les RunningSchedules est enlevé.

5.4 Ecrans

Lors de mes discussions avec mon mentor, il a été assez vite recommandé et conseillé que j'utilise des Eventsources (voir doc) pour envoyer les ordres d'affichages aux écrans. Les sections suivantes résument et décrivent leur utilisation dans ce programme.

5.4.1 Affichage

Pour l'affichage des flux sur les écrans, les technologies autorisées étaient le HTML5, CSS3 et Javascript pur (sans frameworks). La demande initiale était de pouvoir afficher le contenu associé à une URL grâce aux iframes. Il fallait donc un moyen pour les écrans de recevoir une URL sous forme texte. En partant de ce constat, j'ai remarqué que si un Event contenait une URL en texte, il pouvait contenir d'autres données. J'ai donc proposé d'implémenter des flux de type image et texte. Les écrans sont donc capable de déterminer le type du flux reçu et de l'afficher de la manière adéquate. Les différents moyens utilisés l'affichage en question sont décrit dans la section Réalisation.

5.4.2 Events

Un des problèmes à résoudre était la question d'envoyer les bons Events aux bons écrans. Il y avait deux manières principales de faire, soit envoyer tous les Events à tous les écrans avec la liste des écrans concernés, soit générer dynamiquement des endpoints pour chaque Schedule. Pour des raisons de simplicité et parce que le nombre d'écrans restera raisonnable, la première possibilité a été choisie.

Les Events générés par les Schedules sont donc envoyés à tous les écrans s'étant connecté auprès du serveur. Chaque Event contient les adresses MAC des écrans concernés et c'est à l'écran de vérifier s'il est concerné par l'Event qu'il vient de recevoir.

Un Event est construit de la manière suivante : il contient son type, ses "données" (url ou texte) et les écrans concernés. Ci-dessous un exemple pour chaque type :

- URL :
`url?https://heig-vd.ch|mac_address1,mac_address2`
- Image :
`image?/assets/image1|mac_address2, mac_address3`
- Video :
`video?https://www.youtube.com/embed/dQw4w9WgXcQ|mac_address1, mac_address3`
- Texte :
`text?Hello World|mac_address2`

5.4.3 Authentification

Un écran ne peut recevoir d'Event qu'une fois authentifié, pour des raisons évidentes de sécurité. Il fallait donc trouver un moyen d'identifier de manière unique chaque écran. La première piste envisagée fut d'utiliser les hostnames des écrans car cette information était facilement récupérable en Java, et spécialement avec Play. Mais en raison de l'architecture réseau de la HEIG et de la volonté d'intégrer le protocole WakeOnLan au programme, il a été décidé d'utiliser les adresses MAC à la place.

La difficulté inhérente à ceci était de récupérer ces adresses depuis Java. Pour remédier à ce problème et en même temps fournir une couche de sécurité au niveau des écrans, il a été choisi que lors de l'ajout d'un écran au système, son adresse MAC devrait être précisée.

L'accès des écrans serveur se fait en deux étapes. Il faut d'abord que l'écran se connecte à la route d'authentification des écrans, tout en spécifiant comme paramètre de requête son adresse MAC. Exemple :

```
http://server/screens/auth?mac=1234
```

Là, si l'adresse est inconnue par le serveur, il nous renvoie un code permettant l'ajout de l'écran dans le système depuis le site web. Si l'adresse est connue, il ajoute ajoute dans les cookies l'adresse MAC de l'écran ainsi que sa résolutions (utile pour la gestion de l'affichage), passe l'écran comme actif et le redirige sur le endpoint

5.4.4 Groupe d'écrans

Les groupes d'écrans sont créés par des chefs d'équipe et donc uniquement disponible à l'intérieur d'une Team (on pourrait aussi imaginer des groupes communs à toutes les Teams). Ils servent à regrouper les écrans selon leur utilité, pour faciliter leur assignement à un Schedule. Par exemple, un groupe d'écran possible serait celui des écrans du hall de Cheseaux.

Un écran peut faire partie de plusieurs groupes, mais comme précisé plus haut, il ne peut être associé qu'à un RunningSchedule à la fois. Il y aura donc des vérifications qui seront faites au moment d'activer un Schedule sur un groupe d'écran pour empêcher ceci.

6 Protocole de communication écran-serveur

TODO : refaire les images.

Le protocole ci-dessous explique et détaille la communication entre les écrans et le serveur, ainsi que les erreurs potentielles et leur prise en charge.

6.1 Ecran inconnu du serveur

Dans ce cas de figure, l'écran n'a pas encore été ajouté au système par l'administrateur, et donc le serveur ne reconnaît pas son adresse MAC lors de la connexion. A ce moment, il lui renvoie une page HTML qui affiche un code (propre à chaque écran) à utiliser pour introduire l'écran dans le système.

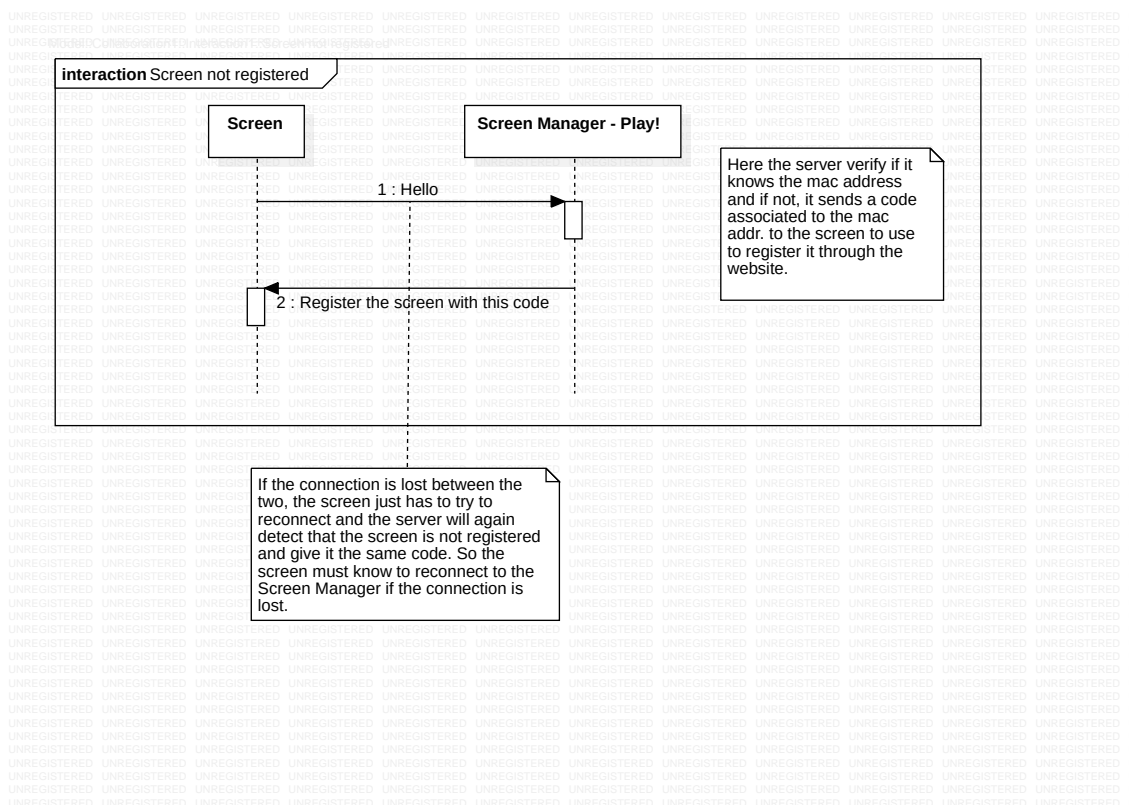


FIGURE 2 – Ecran inconnu

Le fait que l'écran perde sa connexion Internet pendant cet échange ne devrait poser aucun problème car le code sera simplement ré-envoyé à l'écran lors de sa prochaine connexion (en effet l'adresse MAC sera toujours inconnue du serveur).

6.2 Ecran connu du serveur

La figure suivante décrit le cas idéal, où l'écran est connu par le serveur et il n'y a pas de perte de connexion. L'adresse MAC de l'écran est cette fois connue par le serveur, donc il marque l'écran comme

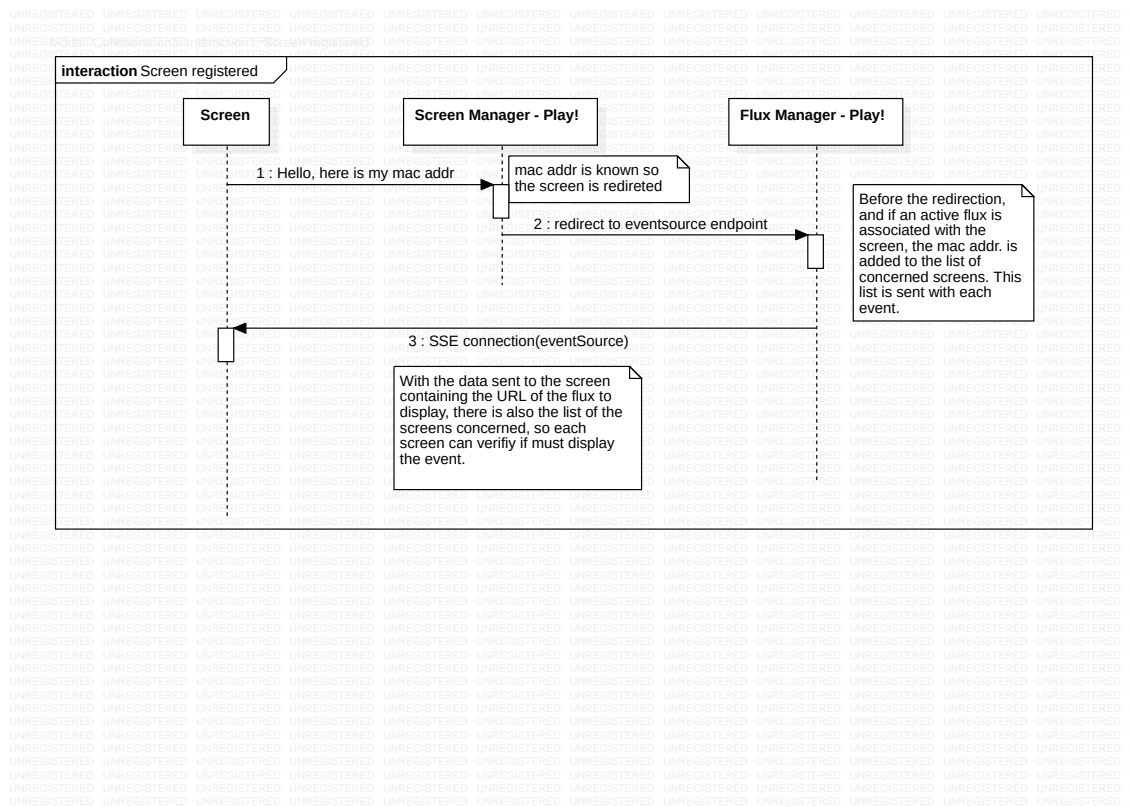


FIGURE 3 – Ecran connu

actif (logged in) et vérifie si l'écran est associé avec un `RunningSchedule`. Si c'est le cas, il ajoute son adresse MAC dans la liste des écrans concernés par ce `Schedule` et le redirige vers le endpoint d'envoi d'Events. Dans le cas contraire, l'écran affiche simplement un message générique (p.ex. "Ecran non-assigné...").

6.3 Ecran connu - erreurs de connexion

Ici sont listées les actions effectuées par le serveur ou le client en cas d'erreurs lors la phase d'authentification.

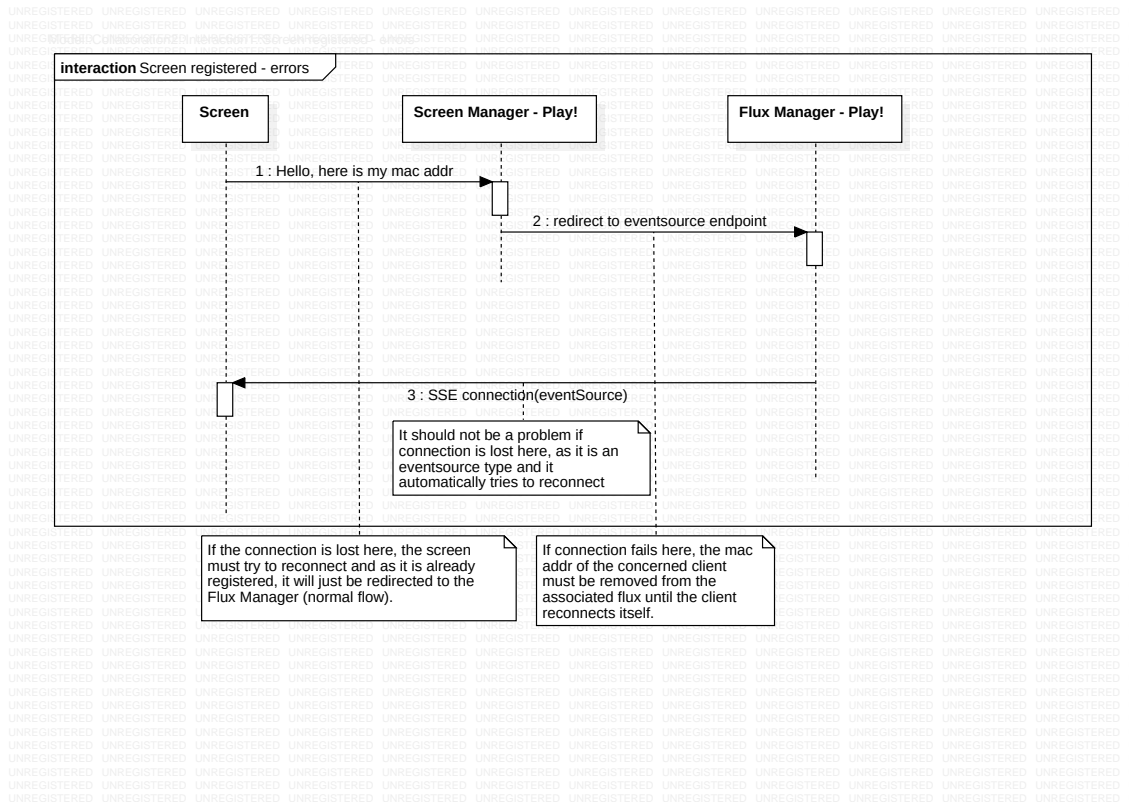


FIGURE 4 – Ecran connu - erreurs

Comme précédemment, le fait qu'un écran perde sa connexion au serveur lorsqu'il essaie de s'authentifier ne devrait pas poser de problèmes, car il lui suffit de ré-essayer. Par contre, si l'écran perd sa connexion une fois qu'il s'est identifié auprès du serveur, il faudra le retirer de la liste des écrans concernés par son RunningSchedule jusqu'à ce qu'il se re-connecte.

7 Réalisation technique

7.1 Organisation des flux

Cette section se concentre sur la manière dont la diffusion de flux est gérée par le système, de l'analyse des Schedules à l'envoi d'Events aux écrans. Ce traitement se fait principalement à l'aide de trois objets, dont voici un schéma :

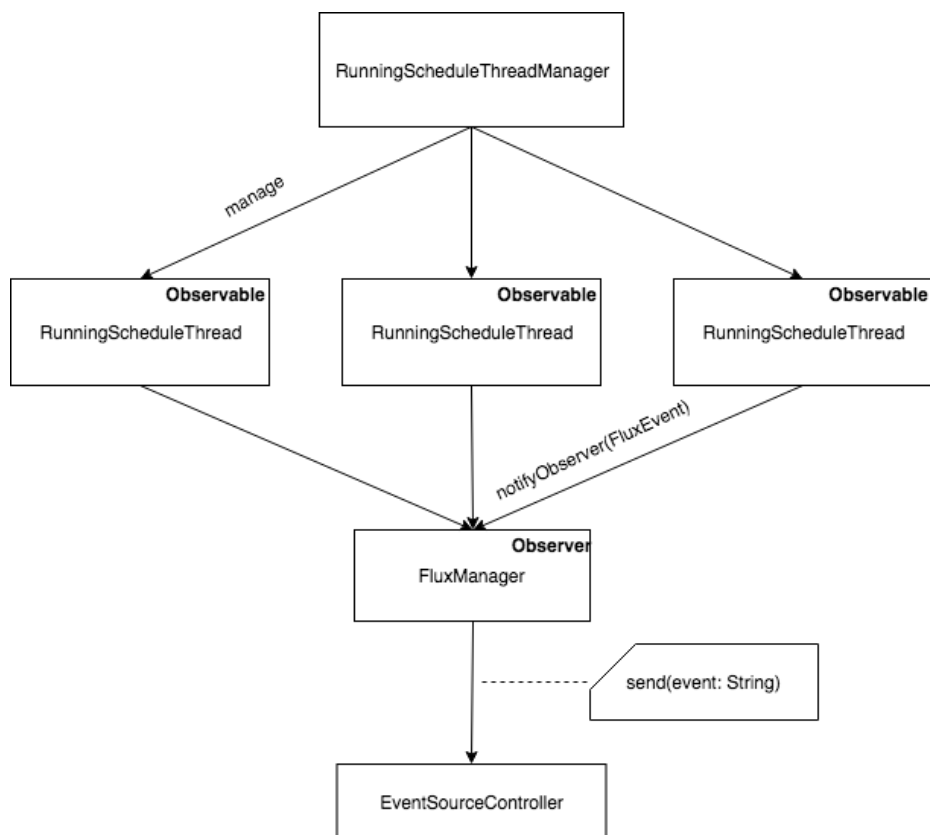


FIGURE 5 – Organisation des flux

En élément central, nous avons les **RunningSchedulesThreads**. Comme leur nom l'indique, c'est eux qui représentent l'exécution d'un **Schedule** actif. Ils sont gérés par un manager, qui permet de les activer ou désactiver ou simplement d'y avoir accès. Ils génèrent des événements selon les flux de leur **Schedule** et les transmettent au **FluxManager** via le patron de conception **Observer**.

Le **FluxManager** est un **Singleton** dont les seules tâches sont de récupérer les **Events** générés par les différents **RunningSchedulesThreads**, construire le message de l'événement à envoyer (donc le type, l'url, les écrans concernés, etc) et de les transmettre aux écrans. Il utilise pour cela la méthode `send(event : String)` fournie par l'**EventSourceController**.

Comme précisé dans le chapitre **Analyse et Architecture**, un système de "bloc-minute" est utilisé pour organiser la diffusion de flux en créant un "bloc-horaire" qui est utilisé par les **RunningSchedulesThreads** pour déterminer quels flux doivent-ils envoyer et quand.

7.1.1 Bloc-horaire

La première implémentation (un peu naïve) que j'ai faites de ces horaires utilisait une simple liste de flux qui, si elle me permettait de vérifier le bon fonctionnement du programme n'offraient que peu de souplesse pour les différentes fonctionnalités à implémenter. Il a donc été choisi de représenter une plage d'affichage comme un ensemble de bloc et j'ai choisi comme valeur temporelle une minute afin de simplifier le processus et d'offrir une plus grande granularité à l'utilisateur.

Concrètement, un "bloc-horaire" est une `Map<Integer, Integer>` qui associe à chaque index de bloc l'id du flux programmé à cette heure-ci ou -1 afin de représenter une absence de flux. Il est construit à chaque fois qu'un **Schedule** est activé à partir de ses données puis fourni au **RunningSchedulesThread** associé. On peut voir dans la figure suivante la manière dont cet horaire est généré :

```

1 public HashMap<Integer, Integer> getTimeTable(Schedule schedule) {
2     List<ScheduledFlux> scheduledFluxes = scheduleRepository.
3     getAllScheduledFluxesByScheduleId(schedule.getId());
4     Flux lastFlux = new Flux();
5     long lastFluxDuration = 0;
6     boolean noFluxSent;
7
8     HashMap<Integer, Integer> timetable = new HashMap<>();
9     for (int i = 0; i < blockNumber; i++) {
10         noFluxSent = true;
11
12         if (lastFluxDuration != 0) {
13             lastFluxDuration--;
14             timetable.put(i, lastFlux.getId());
15         }
16         else {
17             for (ScheduledFlux sf : scheduledFluxes) {
18                 // si un flux doit commencer a ce bloc
19                 if (sf.getStartBlock().equals(i)) {
20                     Flux flux = fluxRepository.getById(sf.getFluxId());
21                     lastFlux = flux;
22                     lastFluxDuration = flux.getTotalDuration() - 1;
23                     timetable.put(i, flux.getId());
24                     noFluxSent = false;
25                     scheduledFluxes.remove(sf);
26                     break;
27                 }
28             }
29             if (noFluxSent) {
30                 // aucun flux ne commence a ce bloc
31                 timetable.put(i, -1);
32             }
33         }
34     }
35     return timetable;
36 }

```

Listing 1 – Création du bloc-horaire

On commence par récupérer les flux avec heure de début du Schedule concerné, puis on remplit l'horaire bloc par bloc en vérifiant à chaque fois si un flux est prévu pour le bloc courant.

Ce système offre principalement deux avantages :

- Il est très facile de modifier le "bloc-horaire" d'un Schedule actif pour en modifier le comportement (changer de flux, en rajouter ou même en supprimer) sans devoir arrêter le programme.
- La reprise automatique de l'exécution des Schedules actifs est elle aussi très simple : le système inspecte la base de données au démarrage et si elle contient des RunningSchedules, il lance les threads associés.

7.1.2 RunningScheduleThread

C'est vraiment dans cette classe que la logique de "scheduling" est implémentée. Voici une version très simplifiée de sa méthode *run()* mais qui garde la même logique fondamentale :

```

1 public void run() {
2
3     while (running) {
4
5         DateTime dt = new DateTime();
6         int hours = dt.getHourOfDay();
7         int minutes = dt.getMinuteOfHour();
8
9         // si on est dans la plage horaire active
10        if (hours >= beginningHour && hours < endHour) {
11            int blockIndex = getBlockNumberOfTime(hours, minutes);
12

```



```

13 do {
14     Flux currentFlux = fluxRepository.getById(timetable.get(blockIndex++));
15
16     // si un flux est prévu pour ce bloc
17     if (currentFlux != null) {
18         sendFluxEventAsGeneralOrLocated(currentFlux);
19     }
20     // sinon on choisi un flux sans heure de debut
21     else {
22         int freeBlocksN = getNumberOfBlocksToNextScheduledFlux(blockIndex);
23         boolean sent = false;
24         int fluxId;
25
26         do {
27             // s'il a ete specifie que l'ordre des flux devait etre respecte
28             if (keepOrder) {
29                 // cycle
30                 unscheduledFluxIds.add(unscheduledFluxIds.get(0));
31                 fluxId = unscheduledFluxIds.remove(0);
32             }
33             // sinon on en prend un au hasard
34             else {
35                 Collections.shuffle(unscheduledFluxIds);
36                 fluxId = unscheduledFluxIds.get(0);
37             }
38             Flux unscheduledFlux = fluxRepository.getById(fluxId);
39
40             // si la place est suffisante
41             if (unscheduledFlux.getTotalDuration() <= freeBlocksN) {
42                 // mise a jour de l'horaire
43                 scheduleFlux(unscheduledFlux, blockIndex);
44                 // envoi de l'event au FluxManager
45                 sendFluxEventAsGeneralOrLocated(unscheduledFlux);
46                 sent = true;
47             }
48         } while (!sent);
49     }
50     // sleep pendant 1 minutes
51     try {
52         if (Thread.currentThread().isInterrupted()) {
53             throw new InterruptedException("Thread interrupted");
54         }
55         Thread.sleep((long) blockDuration * 60000);
56     } catch (InterruptedException e) {
57         e.printStackTrace();
58     }
59
60 } while (blockIndex < blockNumber && running);
61 }
62 }
63 }

```

Listing 2 – Eventsource Java

Lorsqu'il débute, le thread récupère l'heure actuelle et vérifie qu'il est dans les bornes de la plage horaire. Si oui, il récupère l'indice du bloc courant puis commence son exécution. Dans le code ci-dessus ne figure pas toutes les opérations effectuées normalement mais il donne une assez bonne idée de son fonctionnement (l'extrait de code serait trop important sinon). Je préciserais quand même qu'on peut avoir une boucle infinie (lignes 26 à 48) dans le cas où aucun flux sans heure de début ne peut être casé. Ce problème est résolu mais pour des raison de clarté j'ai décidé d'enlever quelques parties de la fonction présentée.

On peut résumer l'algorithme de choix de flux ainsi :

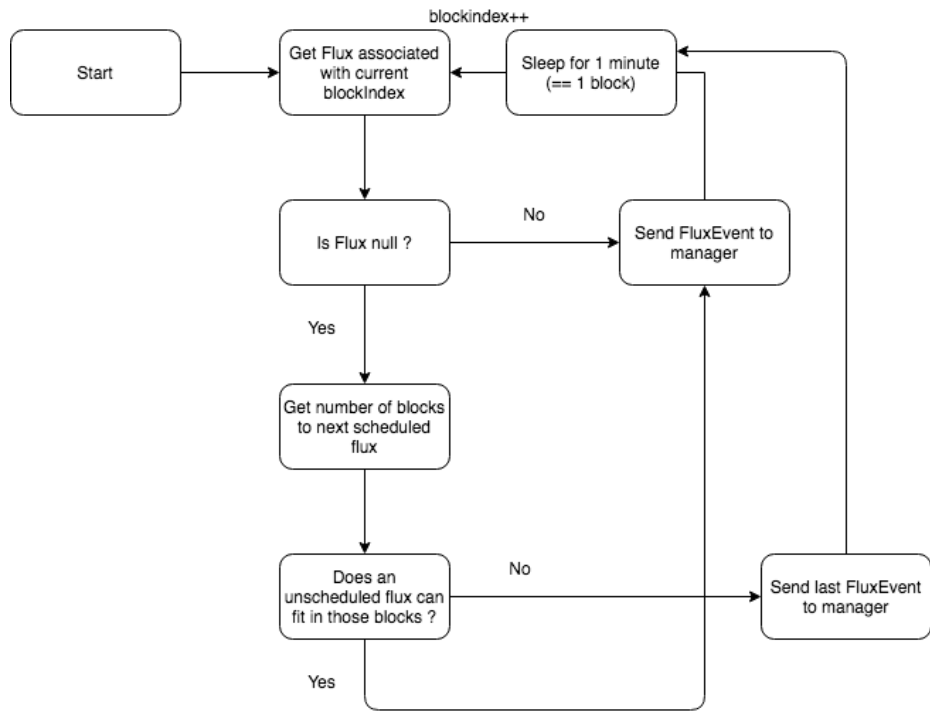


FIGURE 6 – Algorithme de scheduling des flux

7.1.3 FluxManager

Le FluxManager est l'entité responsable de regrouper tout les Events générés par les Schedule pour les transmettre au contrôleur chargé de les envoyer aux écrans. C'est un Runnable Singleton qui est créé par injection de dépendance au démarrage de l'application et dont les contrôleurs ou autres objets peuvent obtenir une référence, par injection de dépendance à nouveau. Pour représenter l'événement d'un flux, j'utilise un objet FluxEvent, qui est composé d'un flux et de la liste des adresses MAC des écrans concernés par cet événement.

C'est également le FluxManager qui construit le contenu de l'Event qui sera envoyé aux écrans (voir Chapitre Analyse et Architecture, section Events). Présenté ci-dessous, sa fonction *run()* simplifiée :

```

1
2 public void run() {
3
4     while (running) {
5         // il y a des event a envoyer
6         if (!fluxEvents.isEmpty()) {
7             FluxEvent currentFlux = fluxEvents.remove(0);
8
9             boolean run = true;
10
11             do {
12                 eventController.send(
13                     currentFlux.getFlux().getType().toLowerCase() +
14                     "?" +
15                     currentFlux.getFlux().getUrl() +
16                     "|" +
17                     String.join(",", currentFlux.getMacs())
18                 );
19
20                 if (!fluxEvents.isEmpty()) {
21                     currentFlux = fluxEvents.remove(0);
22                     if (fluxEvents.isEmpty()) {
23                         run = false;
24                     }
25                 }
26             } else {

```

```

27         run = false;
28     }
29     } while (run);
30 }
31 // attend un peu avant de recommencer
32 else {
33     try {
34         Thread.sleep(2000);
35     } catch (InterruptedException e) {
36         e.printStackTrace();
37     }
38 }
39 }
40 }

```

Listing 3 – FluxManager

La logique de cette classe est elle aussi plutôt simple ; le FluxManager itère sur sa liste de FluxEvents et pour chacun d’entre eux il donne l’ordre au contrôleur d’envoyer un Event aux écrans. Dans le cas où il n’a pas de FluxEvent à traiter, il *sleep* pendant quelques instants avant de recommencer la procédure.

7.2 Contrôleurs

La plupart des contrôleurs de l'application servent uniquement à exécuter des opérations CRUD, mais certains offrent des fonctionnalités plus poussées qui sont décrites dans les sections suivantes.

7.2.1 EventSource

Lors des discussions préalables avec mon mentor, j'avais été prévenu que dans la version 2.7 de Play Java, les Eventources ne fonctionnaient pas car les informations de session étaient perdues lors d'un mapping. Travaillant avec la version 2.7.1 (sensée avoir résolu ce problème), je suis donc parti sur une implémentation en Java. J'ai d'abord pensé avoir réussi, car j'observais un comportement normal, mais en analysant de manière plus approfondie les échanges clients-serveurs lors d'une séance avec mon mentor, nous nous sommes aperçu que la connexion Eventsource était recréée toute les 3 secondes, quand le client essayait de se reconnecter au serveur. Il a donc été nécessaire de passer à une version en Scala.

Ci-dessous une version simplifiée de l'ancien code Java :

```
1 @Singleton
2 public class EventSourceController extends Controller implements Observer {
3
4     private static Source<String, ?> source;
5
6     @Override
7     public synchronized void update(Observable o, Object arg) {
8
9         if (source == null) {
10             source = Source.tick(Duration.ZERO, Duration.ofSeconds(5), "tick");
11         }
12         else {
13             List<String> list = new ArrayList<>();
14             list.add((String) arg);
15             Source<String, ?> s = Source.from(list);
16             source.merge(s);
17         }
18     }
19
20     public Result events() {
21
22         final Source<EventSource.Event, ?> eventSource;
23
24         return ok().chunked(source
25             .map(EventSource.Event::event)
26             .via(EventSource.flow()))
27             .as(Http.MimeTypes.EVENT_STREAM);
28     }
29 }
30 }
```

Listing 4 – Eventsource Java

L'idée de ce contrôleur était de mettre à jour une source à chaque fois qu'un Event devait être envoyé grâce au patron de conception *Observer*.

Après un peu de recherche sur les différents moyen d'implémenter une utilisation des Eventsource en Scala, j'ai choisi une solution basée sur les Akka Actor. Elle semblait la plus simple et la plus adaptée à mon problème, car elle permet de représenter chaque écran par un acteur. Le contrôleur équivalent en Scala :

```

1 @Singleton
2 class EventSourceController @Inject() (system: ActorSystem,
3                                     cc: ControllerComponents)
4                                     (implicit executionContext: ExecutionContext)
5 extends AbstractController(cc) {
6
7
8     private[this] val manager = system.actorOf(EventActorManager.props)
9
10    implicit def pair[E]: EventNameExtractor[E] = EventNameExtractor[E](_ => Some("test1"))
11
12    def send(event: String) = {
13        print("Send event to screens : " + event)
14        manager ! SendMessage(event)
15        Ok
16    }
17
18    def index = Action {
19        Ok(eventsource.render())
20    }
21    def events = Action {
22
23        val source =
24            Source
25                .actorRef[String](32, OverflowStrategy.dropHead)
26                .watchTermination() { case (actorRef, terminate) =>
27                    manager ! Register(actorRef)
28                    terminate.onComplete(_ => manager ! UnRegister(actorRef))
29                    actorRef
30                }
31
32        val eventSource = Source.fromGraph(source.map(EventSource.Event.event))
33
34
35        Ok.chunked(eventSource via EventSource.flow).as(ContentType.EVENT_STREAM)
36    }
37 }
38

```

Listing 5 – Eventsource Scala

On peut voir que ce contrôleur est au final très simple. Il offre deux actions principales : la possibilité pour un écran de s'enregistrer auprès du manager grâce à la méthode *events()* et un moyen pour le système d'envoyer un Event aux écrans avec la méthode *send(event : String)*. Il utilise pour ce faire un ActorSystem, construit à l'aide d'une sous-classe d'Actor, créée dans le cadre du projet. En voici l'implémentation :

```

1 class EventActor extends Actor {
2     private[this] val actors = mutable.Set.empty[ActorRef]
3
4     def receive = {
5         case Register(actorRef)    => actors += actorRef
6         case UnRegister(actorRef) => actors -= actorRef
7         case SendMessage(message) => actors.foreach(_ ! message)
8     }
9 }
10
11 object EventActorManager {
12     def props: Props = Props[EventActor]
13 }

```

```

14 case class SendMessage(message: String)
15 case class Register(actorRef: ActorRef)
16 case class UnRegister(actorRef: ActorRef)
17 }

```

Listing 6 – Akka Actor Scala

7.2.2 Ecrans

C'est par ce contrôleur que passe les écrans souhaitant s'authentifier auprès du système et ainsi recevoir des flux. La logique du comportement étant décrite dans le chapitre **Protocole**, seul l'implémentation sera évoquée ici.

Voici la version simplifiée de cette fonction :

```

1 public Result authentication(Http.Request request) {
2
3     String macAdr = request.queryString().get("mac")[0];
4     Screen screen = getScreenByMacAddress(macAdr);
5
6     // Ecran inconnu du systeme
7     if (screen == null) {
8
9         // L'ecran a deja essayer de s'identifier
10        if (getWaitingScreenByMacAddress(macAdr) != null) {
11            return ok(screen_code.render(getWaitingScreenByMacAddress(macAdr).getCode()
12        ));
13        }
14
15        String code = screenRegisterCodeGenerator();
16        add(new WaitingScreen(code, macAdr));
17
18        // Envoi du code
19        return ok(screen_code.render(code));
20    }
21    // Ecran connu du systeme
22    else {
23        // Pas de Schedule actif pour cet ecran
24        if (getRunningScheduleIdByScreenId(screen.getId()) == null) {
25            return redirect(routes.ErrorPageController.noScheduleView());
26        }
27
28        // Ecran actif
29        if (!screen.isLogged()) {
30            screen.setLogged(true);
31        }
32        return ok(eventsource.render()).withCookies(
33            Http.Cookie.builder("mac", macAdr)
34                .withHttpOnly(false)
35                .build(),
36            Http.Cookie.builder("resolution", screen.getResolution())
37                .withHttpOnly(false)
38                .build());
39    }
40 }

```

Listing 7 – Authentification des écrans

7.2.3 Schedules

Ce contrôleur permet d'activer et de désactiver des Schedules.

7.2.3.1 Activation

7.2.3.2 Désactivation

7.2.4 Diffusers

Ce contrôleur permet d'activer et de désactiver des Diffusers.

7.2.4.1 Activation

7.2.4.2 Désactivation

7.3 Vues

7.3.1 Affichage des flux

L'affichage des Events reçus par les écrans se fait à l'aide d'une page HTML simple, qui définit les balises nécessaires à l'affichage des différents types de flux. A part un peu de CSS, rien d'autre ne s'y passe. En voici le body :

```
1 <body>
2   <div class="frame">
3     <iframe id="frame0" class="frame" frameborder="0"></iframe>
4   </div>
5
6   <div>
7     <img id="image0">
8   </div>
9
10  <div id="footer0" class="footer"></div>
11 </body>
```

Listing 8 – Eventsource HTML

Il est couplé d'un script Javascript qui crée la connexion SSE avec le serveur et qui traite les Events qu'il reçoit par la suite. Selon le type de flux, il appelle des fonctions d'affichage différentes, qui agissent avec JQuery sur les balises présentées précédemment pour en modifier le contenu ou les cacher. Ci-dessous est présentée la fonction principale de ce script, avec l'analyse des Events reçus afin de déterminer le traitement adéquat :

```
1 $(document).ready(function () {
2   var evtSource = new EventSource("http://localhost:9000/eventsource", {withCredentials:
3     true});
4   var macAdress = getCookie("mac");
5   var resolution = getCookie("resolution");
6
7   evtSource.addEventListener('message', function(e) {
8     var type = e.data.substr(0, e.data.indexOf("?"));
9     var data = e.data.substr(e.data.indexOf("?") + 1, e.data.indexOf("|") - type.length
10     - 1);
11     var macs = e.data.substr(e.data.indexOf("|") + 1).split(",");
12
13     if (macs.includes(macAdress)) {
14       if (type === "url") {
15         displayFlux(data);
16       }
17       else if (type === "image") {
18         if (resolution === "1080") {
19           displayImage(data, 1900, 1080);
20         }
21         else if (resolution === "720") {
22           displayImage(data, 1280, 720);
23         }
24       }
25       else if (type === "text") {
26         displayFooterText(data);
27       }
28       else if (type === "video") {
29         displayVideo(data);
30       }
31     }
32   });
33 });
```

Listing 9 – Eventsource JS

7.3.2 Echange de données

Pour échanger des informations entre le serveur et le client, j'ai choisi d'utiliser les fonctionnalités offertes par Play!, donc avoir des entités représentant mes données (par exemple un Flux devient un FluxData) et les utiliser avec les templates Scala, Play permettant de facilement transférer des objets Java des contrôleurs aux vues. Ces templates sont des blocs de texte contenant du code Scala qui est par la suite compilé en HTML. Il devient alors facile de combiner cela à Bootstrap pour l'affichage ou l'envoi de données depuis le client.

Exemple d'entité :

```
1 public class UserData {
2
3     private String email;
4
5     private String password;
6
7     public UserData() {
8     }
9     // getters and setters...
10 }
```

Listing 10 – UserData.java

7.3.2.1 Serveur -> client

```
1 // Serveur
2 public Result index() {
3     return ok(team_page.render(dataUtils.getAllTeams(), null));
4 }
5
6
7
8 // Client: team_page.scala.html
9 @(teams: util.List[TeamData], error: String)
10
11 @if(teams != null) {
12     @for(team <- teams) {
13         <tr>
14             <td>@team.getName()</td>
15             <td>@if(team.getMembers() != null) {
16                 team.getMembers().length
17             }</td>
18         </tr>
19     }
20 }
```

Listing 11 – Exemple échange serveur-client

Ici, datautils.getAllTeams() renvoie une liste de **TeamData** qui sont récupérés et utilisés ensuite par le fichier html. On peut observer ici que l'intégration d'une boucle for avec une Table Bootstrap.

7.3.2.2 Client -> serveur

```
1 // Client
2 @helper.form(routes.UserController.register()) {
3     @helper.CSRF.formField
4     <div class="form-group">
5         <input name="email" type="email" class="form-control">
6     </div>
7     <div class="form-group">
8         <input name="password" type="password" class="form-control">
9     </div>
```

```

10     <button type="submit" class="btn btn-primary">Register</button>
11 }
12
13
14 // Serveur: UserController.java
15 public Result register(Http.Request request) {
16     final Form<UserData> boundForm = form.bindFromRequest(request);
17     UserData data = boundForm.get();
18     User newUser = new User(data.getEmail(), data.getPassword());
19
20     // email is not unique
21     if (userRepository.getByEmail(newUser.getEmail()) != null) {
22         return registerViewWithErrorMessage("Email is already used");
23     }
24     else
25         ...
26 }

```

Listing 12 – Exemple échange client-serveur

Dans cet extrait de code, on a un exemple des fonctionnalités offertes par Play sous la forme de helpers servant à faciliter la création de formulaires Bootstrap. L'action effectuée par le bouton du formulaire est directement liée à la méthode de `UserController.java`. Il faut choisir comme valeur pour l'attribut `name` des balises `input` les noms des attributs correspondant dans le modèle correspondant (`UserData` en l'occurrence).

Le serveur est par la suite capable de reconstruire un objet du même type en récupérant le formulaire depuis la requête.

8 Cas d'utilisation

Les cas d'utilisation suivants sont regroupés par catégorie d'utilisateurs. Il y en a trois : les administrateurs, les chefs d'équipe (TeamAdmin) et les simples membres d'une équipe (TeamMember). Les admins ne sont associés à aucun écrans tandis que les deux autres sont restreints à certains écrans. Toute action possible pour une catégorie l'est également pour celles en dessus.

8.1 Administrateur :

L'admins peut effectuer toutes les actions et est le seul à pouvoir ajouter ou supprimer des écrans ou des utilisateurs au système. L'écran est allumé et connecté dans tous les cas suivants.

- **Scénario 1 : Ajout d'un écran**

Déroulement : L'admins rentre l'URL d'authentification des écrans dans le navigateur en spécifiant l'adresse MAC de l'écran comme paramètre de requête. Le serveur ne reconnaît pas l'adresse MAC envoyée et renvoie donc un code servant à enregistrer l'écran dans le système. L'admins passe donc par le site pour ajouter l'écran en spécifiant entre autres son adresse MAC, son emplacement et le code fourni précédemment.

Résultat : L'écran sera maintenant reconnu par le serveur et correctement redirigé à la prochaine tentative.

Erreurs potentielles : Si la connexion est perdue entre l'écran et le backend à n'importe quel moment du scénario, les mêmes opérations seront effectuées à la re-connexion de l'écran (envoi du code).

- **Scénario 2 : Mise à jour des infos d'un écran**

Pré-requis : l'écran est déjà connu par le système.

Déroulement : L'admins se connecte au site et utilise l'interface fournie pour mettre à jour les infos souhaitées (nécessite potentiellement que l'écran ne soit pas actif).

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération.

- **Scénario 3 : Suppression d'un écran**

Pré-requis : l'écran est déjà connu par le système.

Déroulement : L'admins se connecte au site et supprime l'écran du système en utilisant l'interface.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et l'adresse MAC de l'écran est supprimée du système.

- **Scénario 4 : Ajout d'un utilisateur**

Déroulement : L'admins se connecte au site et ajoute l'utilisateur en utilisant l'interface fournie. Lors de l'ajout, il spécifie les écrans auxquels l'utilisateur pourra assigner des Schedules.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et l'utilisateur est ajouté à la base de donnée.

- **Scénario 5 : Modération : désactivation de Schedule**

Pré-requis : Le Schedule est activé.

Déroulement : L'admins se connecte au site et va sur la page des Schedules. Dans la liste des actifs, il sélectionne celui qu'il veut désactiver et confirme son choix.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et le Schedule est désactivé.

- **Scénario 6 : Modération : désactivation de Diffuser**

Pré-requis : Le Diffuser est activé.

Déroulement : L'admins se connecte au site et va sur la page des Diffusers. Dans la liste des actifs, il sélectionne celui qu'il veut désactiver et confirme son choix.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération, le Diffuser est désactivé et son flux est retiré des Schedules correspondants.

- **Scénario 7 : Création d'une Team**

Déroulement : L'admins se connecte au site et va sur la page des Teams. Il utilise l'interface fournie pour créer une nouvelle Team. Il doit spécifier à la création le nom de la Team et les

écrans accessibles par ses membres.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et la Team est créée et ajoutée en BD.

Erreurs potentielles :

— Si le nom choisi pour la Team existe déjà, une erreur sera lancée et l'admins devra en choisir un autre.

— **Scénario 8 :** Modification d'une Team

Pré-requis : La Team existe.

Déroulement : L'admins se connecte au site et va sur la page des Teams. Il utilise la même interface que pour la création pour mettre à jour les infos souhaitées (nom, membres, admins).

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et la Team est modifiée.

Erreurs potentielles :

— Si le nom choisi pour la Team existe déjà, une erreur sera lancée et l'admins devra en choisir un autre.

— **Scénario 9 :** Suppression d'une Team

Pré-requis : La Team existe.

Déroulement : L'admins se connecte au site et va sur la page des Teams. Il sélectionne dans la liste celle qu'il souhaite supprimer et utilise l'interface fournie pour le faire.

Résultat : L'admins est informé du succès (ou de l'échec) de l'opération et la Team est supprimée. Les entités associés avec cette équipe sont également supprimées (Schedules et Diffuser).

8.2 TeamAdmin :

Un TeamAdmin ne peut ajouter d'écrans mais a la permission d'activer Schedules et Diffusers.

— **Scénario 1 :** Activation d'un Schedule

Pré-requis : Le Schedule existe et les écrans choisis ne sont pas déjà assignés à un autre Schedule.

Déroulement : Le TeamAdmin se connecte au site et va sur la page des Schedules. Il choisit dans la liste affichée celui qu'il veut activer et utilise l'interface pour assigner des écrans ou groupes d'écrans à ce Schedule. Il peut ensuite activer son Schedule.

Résultat : Le TeamAdmin est informé du succès (ou de l'échec) de l'opération.

Erreurs potentielles :

- Si le Schedule contient un flux restreint à un site et que l'on l'assigne à un écran sur un autre site, le système nous empêchera de le faire. Par contre, assigner un groupe d'écran avec un sous-ensemble de ce groupe d'un site différent sera possible (un flux de backup sera diffusé sur cet écran à la place).
- Si, parmi les écrans choisis, un ou plusieurs sont déjà assignés à un Schedule, le TeamAdmin en est prévenu et doit changer sa sélection.

— **Scénario 2 :** Activation d'un Diffuser

Pré-requis : Le Diffuser existe et les écrans choisis sont assignés à un Schedule.

Déroulement : Le TeamAdmin se connecte au site et va sur la page des Diffusers. Il choisit dans la liste affichée celui qu'il veut activer et utilise l'interface pour assigner des écrans ou groupes d'écrans à ce Diffuser. Il peut ensuite l'activer.

Résultat : Le TeamAdmin est informé du succès (ou de l'échec) de l'opération.

Erreurs potentielles :

- L'heure de début prévue pour le flux du Diffuser est identique (ou à peine après) à l'heure de début d'un flux du Schedule. Il y a plusieurs manières de traiter ce cas : checker la durée du nouveau flux et reprendre l'exécution de l'ancien une fois fini, repousser un des deux flux (pas top je pense).

— **Scénario 3 :** Création de groupe d'écrans

Déroulement : Le TeamAdmin se connecte au site et va sur la page des écrans. Il choisit les écrans (au moins 2) dans la liste pour son groupe et confirme son choix. (Les écrans peuvent appartenir à plusieurs groupes)

Résultat : Le TeamAdmin est informé du succès (ou de l'échec) de l'opération et le groupe est créé en BD.

Erreurs potentielles :

- Moins de 2 écrans sont choisis, le TeamAdmin est informé et doit choisir plus d'écrans.

— **Scénario 4 :** Modification de groupe d'écrans

Pré-requis : Le groupe existe.

Déroulement : Le TeamAdmin se connecte au site et va sur la page des écrans/groupes. Il choisit dans la liste des groupes celui ou ceux qu'il désire modifier et utilise pour ce faire la même interface que pour la création de groupe.

Résultat : Le TeamAdmin est informé du succès (ou de l'échec) de l'opération et le groupe est modifié en BD.

Erreurs potentielles :

- Si le groupe est actuellement assigné à un RunningSchedule, la modification est empêchée et le TeamAdmin en est informé.
- Si le groupe modifié contient moins de deux écrans, la modification est empêchée et le TeamAdmin en est informé.

— **Scénario 5 :** Suppression de groupe d'écrans

Pré-requis : Le groupe existe.

Déroulement : Le TeamAdmin se connecte au site et va sur la page des écrans/groupes. Il choisit dans la liste des groupes celui ou ceux qu'il désire supprimer et utilise l'interface fournie pour le faire.

Résultat : Le TeamAdmin est informé du succès (ou de l'échec) de l'opération et le groupe est supprimé en BD.

Erreurs potentielles :

- Si le groupe est actuellement assigné à un RunningSchedule, la suppression est empêchée et le TeamAdmin en est informé.

8.3 TeamMember :

Un TeamMember est assigné à une Team, qui elle à accès à des écrans, Schedules et Diffusers. Il peut créer et modifier des Schedules et Diffuser non-actifs mais ne peut pas les activer. Comme tous les autres types d'utilisateurs, il peut créer des flux.

- **Scénario 1 :** Création d'un flux

Déroulement : Le TeamMember se connecte au site et va sur la page des flux. Il entre les paramètres de son flux (à définir)

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération et le flux est ajouté à la liste des flux disponibles.

- **Scénario 2 :** Modification d'un flux

Pré-requis : Le flux existe.

Déroulement : Le TeamMember se connecte au site et va sur la page des flux. Il choisit le flux à modifier dans la liste et utilise la même interface que pour la création pour la mise à jour.

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération.

- **Scénario 3 :** Suppression d'un flux

Pré-requis : Le flux existe.

Déroulement : Le TeamMember se connecte au site et va sur la page des flux. Il choisit le flux à supprimer et confirme son choix.

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération et le flux est retiré de la liste des flux disponibles.

- **Scénario 4 :** Création d'un Schedule

Pré-requis : Des flux ont préalablement été créés.

Déroulement : Le TeamMember se connecte au site et va sur la page de création de Schedules. Il choisit les heures de début des flux en associant le flux voulu. Il peut encore spécifier le nom du Schedule ou un commentaire sur son utilité. Il confirme son choix.

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération et le Schedule est ajouté à la liste des Schedules disponibles.

Erreurs potentielles : Les heures de début de flux ne sont pas cohérentes (confirmation alors impossible).

- **Scénario 5 :** Modification d'un Schedule

Pré-requis : Le Schedule existe.

Déroulement : Le TeamMember se connecte au site et va sur la page des Schedules. Il choisit le Schedule à modifier dans la liste et utilise la même interface que pour la création pour la mise à jour.

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération.

Erreurs potentielles : Les heures de début des nouveaux flux ne sont pas cohérentes (confirmation alors impossible).

- **Scénario 6 :** Création d'un Diffuser

Pré-requis : Des flux ont préalablement été créés.

Déroulement : Le TeamMember se connecte au site et va sur la page de création de Diffuser. Il choisit les heures de début du flux voulu et précise sa durée de validité (en jours?, semaines?). Il peut encore spécifier le nom du Diffuser ou un commentaire sur son utilité. Il confirme son choix.

Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération et le Diffuser est ajouté à la liste des Diffusers disponibles.

Erreurs potentielles : Les heures de début de flux ne sont pas cohérentes (confirmation alors impossible).

— **Scénario 7 : Modification d'un Diffuser**

Pré-requis : Le Diffuser existe.

Déroulement : Le TeamMember se connecte au site et va sur la page des Diffuser. Il choisit le Diffuser à modifier dans la liste et utilise la même interface que pour la création pour la mise à jour.

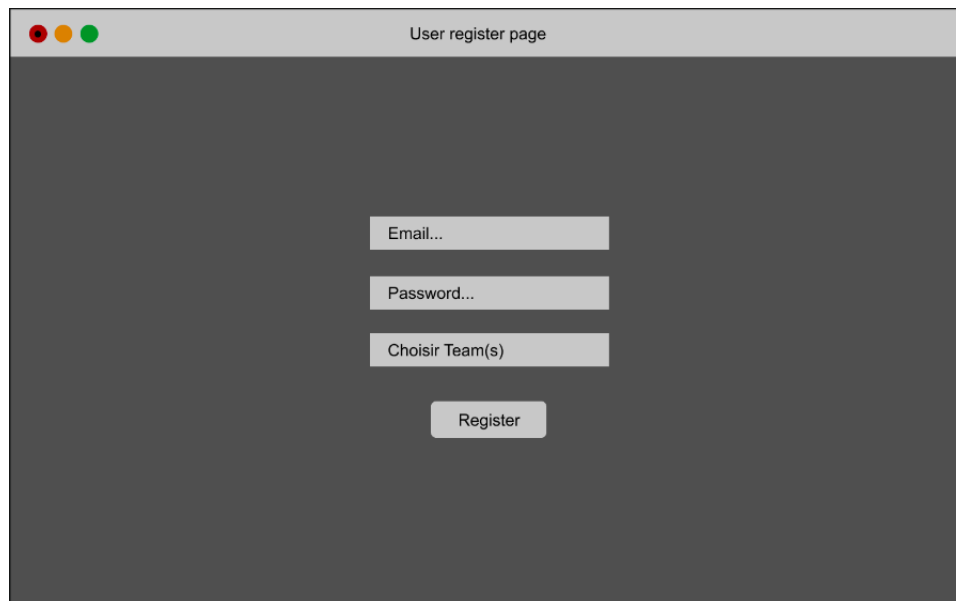
Résultat : Le TeamMember est informé du succès (ou de l'échec) de l'opération.

Erreurs potentielles : Les heures de début des nouveaux flux ne sont pas cohérentes (confirmation alors impossible).

9 Mockups

Les mockups suivant ne sont pas représentatifs de l'aspect final de l'application mais plutôt des fonctionnalités offertes.

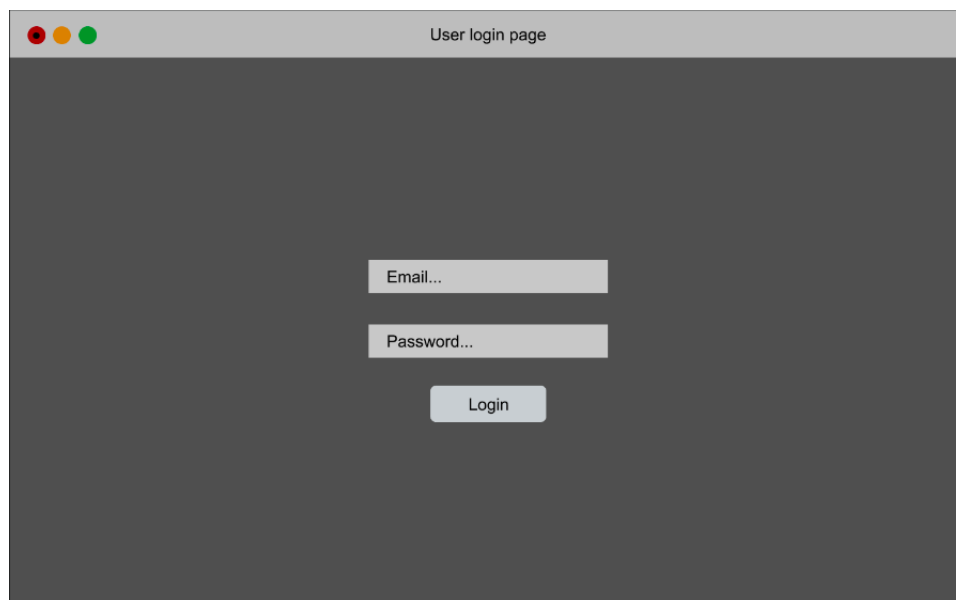
9.1 Utilisateurs



The mockup shows a window titled "User register page". It features a dark gray background with three light gray input fields stacked vertically: "Email...", "Password...", and "Choisir Team(s)". Below these fields is a light gray button labeled "Register". The window has a standard macOS-style title bar with red, yellow, and green window control buttons on the left.

FIGURE 7 – Interface d'ajout de nouveaux utilisateurs

Cette page en Figure 7 sera uniquement accessible aux administrateurs (dans le cadre de mon TB en tout cas).



The mockup shows a window titled "User login page". It features a dark gray background with two light gray input fields stacked vertically: "Email..." and "Password...". Below these fields is a light gray button labeled "Login". The window has a standard macOS-style title bar with red, yellow, and green window control buttons on the left.

FIGURE 8 – Interface de connexion des utilisateurs

9.2 Ecrans

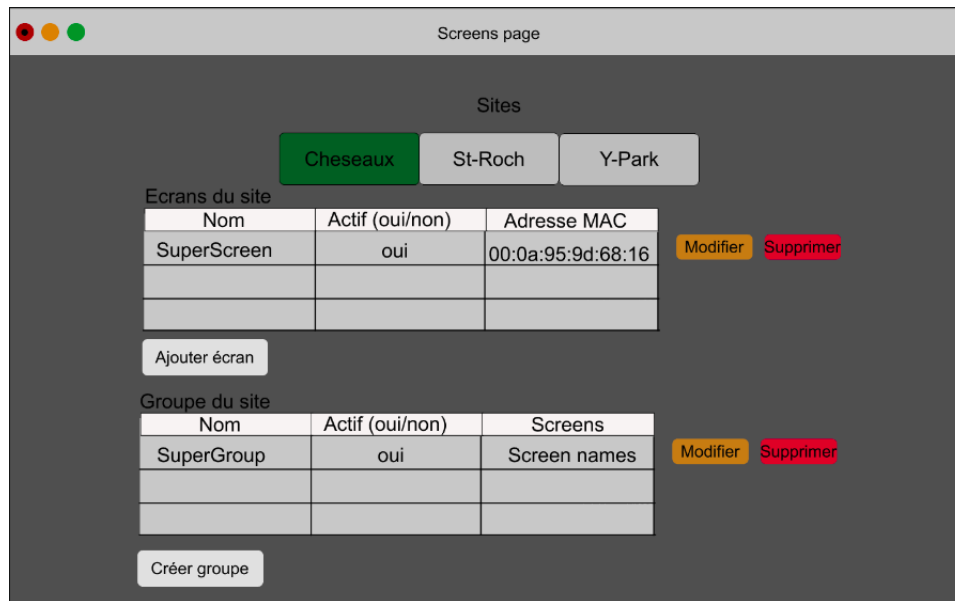


FIGURE 9 – Page principale des écrans

Cette page affiche les écrans accessibles par l'utilisateur actuel, regroupés par site. On a également accès aux groupes d'écrans.



FIGURE 10 – Interface d'ajout de nouvel écran

L'administrateur peut ajouter un nouvel écran au système par le biais de cette interface. Il doit préciser quelques paramètres et surtout donner le code fourni précédemment par le serveur.

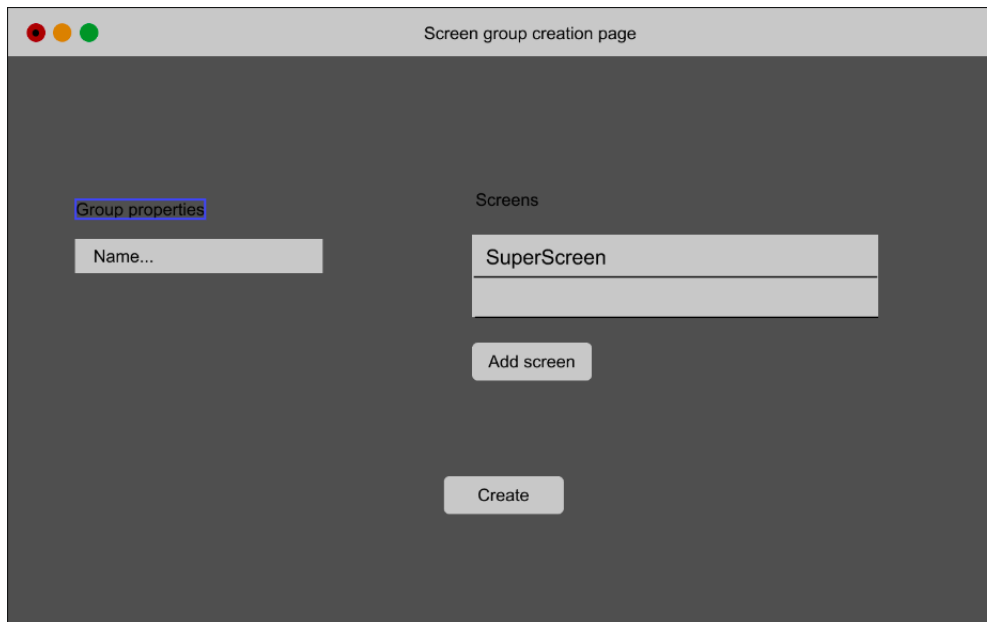


FIGURE 11 – Interface d'ajout de nouveau groupe d'écran

9.3 Teams

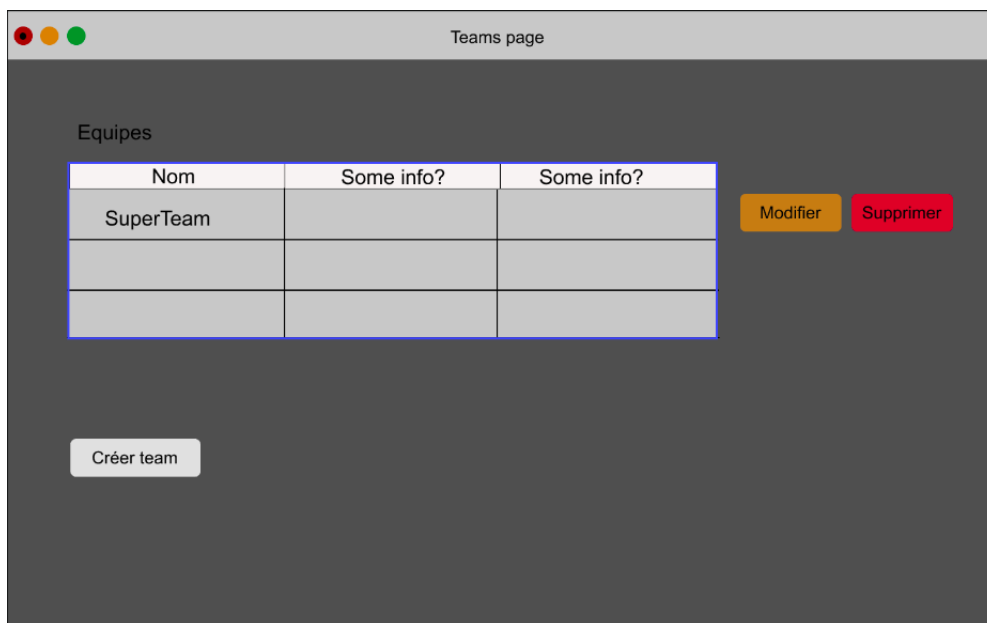


FIGURE 12 – Page principale des teams

Team creation page

Team properties

Nom de team...

Membres

SuperMember

Add member

Team access

Flux

SuperFlux

Add flux

Schedules

SuperSchedule

Add schedule

Créer

FIGURE 13 – Interface de création de team

9.4 Flux

Flux page

Flux localisés

Flux généraux

Cheseaux

St-Roch

Y-Park

Flux disponibles

Nom	Type	Durée
SuperFlux	video	5 min

Modifier Supprimer

Ajouter flux

FIGURE 14 – Page principale des flux

Flux creation page

Flux properties

Name...

URL...

Choose type from list

Description...

Flux duration

Number of phases...

Duration per phase...

Create

FIGURE 15 – Interface de création de flux

9.5 Schedules

Schedules page

Schedules

	Nom	Actif (oui/non)	Some info
✕	SuperSchedule	oui	
✕			

Modifier Supprimer

Ajouter schedule Activer/Désactiver sélection

FIGURE 16 – Page principale des runningSchedules

On choisit les flux principaux qui vont constituer le cycle du Schedule. On peut également définir des flux de fallback qui prendront le relais si le flux actif n'a aucune information à afficher.

Schedule creation page

Schedule properties

Name...

Create

Cycle des flux

Horaire des cours

Réservation des salles

RTS

Add flux

Flux de fallback

Baleinev

Chill-out

Add fallback flux

FIGURE 17 – Interface de création de runningSchedule

9.6 Diffusers

Diffusers page

Diffusers

	Nom	Actif (oui/non)	Valide jusqu'au	
✕	SuperDiffuser	oui	12/04/2019	Modifier Supprimer
✕				

Ajouter Diffuser Activer/Désactiver sélection

FIGURE 18 – Page principale des diffusers

On spécifie la durée de validité du Diffuser en choisissant une date limite.

10 Tests

10.1 Tests JUnit

10.2 Tests fonctionnels

11 Remarques personnelles et commentaires

Diffuser creation page

Diffuser properties

Name...

Valid until... (choose date)

Flux envoyé

Flux: Annonce parking fermé

Add flux

Heure d'envoi...

Create

FIGURE 19 – Interface de création de diffuser