Usabilité

1. Scénario : Usabilité 2

2. Scénario : Un utilisateur du système de *helpdesk* souhaite réduire le temps nécessaire pour utiliser certaines fonctionnalités. 4

3. Scénario : L’utilisateur a fait une erreur dans une requête de nouvelle transaction et veut l’annuler. 7

4. Scénario : Panier d’achats 10

5. Scénario : L’utilisateur souhaite consulter les données d’historique de 7 ans pour réaliser un rapport 13

6. Scénario : L’utilisateur désire automatiser ses transactions 16

7. Vue architecturale 16

8. Scénario : Un utilisateur désire acheter des actions de plusieurs entreprises à travers l’application. 17

9. Scénario : Permettre de suspendre et résumer une transaction en bourse. 21

10. Scénario : Permettre à un utilisateur d’utiliser des gabarits de transaction. 23

11. Scénario : L’utilisateur veut créer un compte pour accéder aux fonctionnalités du marché boursier 27

12. Scénario : Un actionnaire de la bourse modifie une transaction en cours 31

13. Scénario 1: Le client veut pouvoir annuler une transaction. 34

# Scénario : Usabilité

| **Objectifs d'affaires** | Augmenter la qualité de l’utilisation du système par les usagers à l’interne qui veulent consulter l’historique des transactions faites par le système. |
| --- | --- |
| **Source** | *The end user (*qui est aussi un développeur) |
| **Stimulus** | L’utilisateur veut utiliser le logiciel de manière plus efficace |
| **Artéfact** | Le module qui “log” les transactions |
| **Environnement** | *Runtime* |
| **Réponse** | Les features pour augmenter l’efficacité et réduire le temps de travail on été implémenté et sont accessible par les utilisateurs. |
| **Mesure de la réponse** | Le taux de satisfaction des utilisateurs (mesuré à l’aide de sondages) |
| **Questions** |  |

## Tactique 1 : Aggregate

**Description**: il est possible d’effectuer la même opération sur plusieur éléments similaires.

**Justification**: Afin d’augmenter la productivité des utilisateurs nous allons permettre d’effe goctuer une seule fois pour plusieurs éléments une opération normalement répétitive.

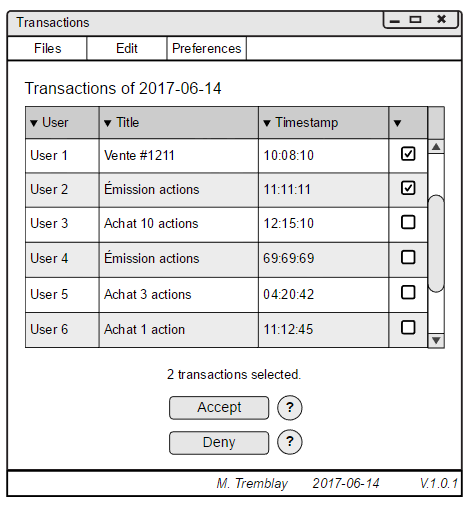
## Tactique 2 : Maintain user model

**Description**: Le modèle offre de l’assistance à l’utilisateur en fonction de ses connaissance.

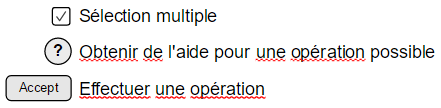
**Justification**: Si un utilisateur est nouveau à ce gros système boursier, il aura des suggestions et des messages d’aide lors de ses premières utilisation. Il peut afficher des fenêtre d’aide s’il le désire les fois suivantes, car il peut être encore perdu face à la complexité du système.

## Vue architecturale (à l’extérieur de la classe)

### Prototype



### Légende



### Texte de description du prototype

L’utilisateur peut voir des transactions qui nécessitent d’être approuvés. Pour éviter des actions répétitives, il peut faire une sélection multiple sur des transactions. S’il est débutant avec l’interface, il a l’option d’appuyer sur les “?” pour avoir de l’assistance.

### Table de description des éléments du diagramme

*Voir légende*

### Texte décrivant la relation entre les éléments et les tactiques

L’option de sélection multiple permet d’intégrer la tactique *Aggregate*, et les bulles d’aides permettent d’intégrer la tactique *Maintain user model*

# Scénario : Un utilisateur du système de *helpdesk* souhaite réduire le temps nécessaire pour utiliser certaines fonctionnalités.

| **objectifs d'affaires** | Augmenter le rendement et l’efficacité des utilisateurs du système de *helpdesk* |
| --- | --- |
| **Source** | User du système de *helpdesk* |
| **Stimulus** | Utiliser le système efficacement |
| **Artéfact** | Le système de *helpdesk* |
| **Environnement** | Runtime (utilisation du système) |
| **Réponse** | Offrir les fonctions nécessaires à une utilisation conviviale et rapide. |
| **Mesure de la réponse** | Mesurer le temps nécessaire pour effectuer une tâche, en milliseconde. |
| **Questions** | 1. Quelles sont les fonctions demandées par l’utilisateur? 2. Quelles sont les tâches répétitives? |

## Tactique 1 : Undo

**Description**: Permettre à l’utilisateur de revenir à un état du système précédent

**Justification**: L’utilisateur n’a pas à corriger une erreur manuellement et peut rapidement revenir en arrière.

## Tactique 2 : Aggregate

**Description**: Agréger plusieurs opérations similaires ou identiques en un groupe unique afin que l’opération y soit appliquée une seule fois.

**Justification**: Réduire le nombre total d’opérations nécessaires et par le même temps, le temps total d’utilisation du système.

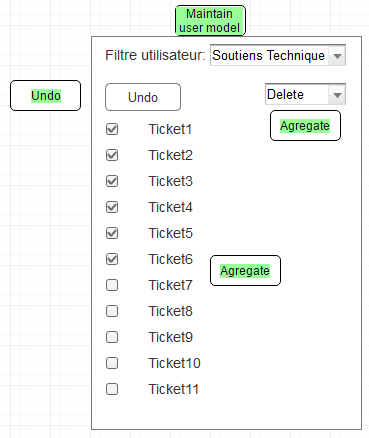
## Tactique 3 : Maintain user model

**Description**: Personnaliser le système aux préférences et habitudes de l’utilisateur pour en faciliter l’utilisation.

**Justification**: Permets d’utiliser le système plus rapidement et efficacement en regroupant les tâches fréquentes à un endroit rapide d’accès.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende

|  | Liste déroulante permettant de faire un choix. |
| --- | --- |
|  | Bouton permettant d'exécuter une action. |
|  | Permets de choisir les éléments pour y effectuer une action groupée. |

### Texte de description du diagramme

Les “checkbox” sont utilisés pour permettre à l’utilisateur de choisir les tickets pour faire une action telle que Delete sur un groupe de ticket et en utilisant la liste déroulante pour choisir l’action désirée sur le groupe de tickets. Le bouton Undo permet de revenir en arrière, par exemple si un ticket est supprimé par erreur. La liste déroulante pour le filtre utilisateur est utilisée pour permettre à l’utilisateur de choisir un profil d’affichage des tickets et leur permettre d’avoir une vue personnalisée.

### Table de description des éléments du diagramme

| Aggregate: Checkbox et liste déroulante | Checkbox permet de choisir exactement les tickets pour ensuite choisir l’action à faire dans la liste déroulante qui permet de sauver du temps en effectuant les actions en groupe. |
| --- | --- |
| Undo: Bouton Undo | Bouton permettant de revenir en arrière en utilisant le patron Commande pour pouvoir revenir en arrière. |
| Maintain user model: liste déroulante | Permets à l’utilisateur de modifier l’affichage du système et ensuite garder ce modèle et pouvoir l’utiliser facilement lors de l'ouverture du système de ticket. |

### Texte décrivant la relation entre les éléments et les tactiques

La tactique **Aggregate** est utilisé pour fournir une façon de faire plusieurs fois la même action en un seul clique et est utilisée en conjonction avec la méthode **Undo** pour permettre de de revenir en arrière après avoir fait une opération simple ou groupée. **Maintain user model** est ensuite utilisé pour permettre la modification de la vue et pouvoir afficher seulement certains tickets pour ensuite facilité la modification en groupe de **Aggregate**,

# Scénario : L’utilisateur a fait une erreur dans une requête de nouvelle transaction et veut l’annuler.

| **objectifs d'affaires** | Permettre à l’utilisateur de canceller une requête de nouvelle transaction pas encore traitée. |
| --- | --- |
| **Source** | Externe au système (SAIL et/ou FIX) |
| **Stimulus** | L’utilisateur veut annuler sa requête de transaction erronée. |
| **Artéfact** | Serveur, base de données |
| **Environnement** | Opération normale |
| **Réponse** | La requête de transaction est cancellée |
| **Mesure de la réponse** | * Satisfaction de l’utilisateur * Temps que l’utilisateur a pris pour canceller une requête de transaction |
| **Questions** | 1. Combien de temps l’utilisateur doit-il prendre pour canceller une requête de transaction? 2. Sur une échelle de 1 à 10, quel est le niveau de satisfaction de l’utilisateur concernant la fonctionnalité de cancellation d’une requête de transaction? 3. Est-ce les données de la requête de transaction cancellée sont consultables? 4. Est-ce que le bouton de cancel est bien placé et assez visible? 5. Quel est le nombre d’étapes requis pour effectuer une cancellation? |

## Tactique 1 : Undo

**Description**: Défaire séquentiellement les actions effectuées pour revenir à un état précédent.

**Justification**: Cette stratégie permettra au système de revenir à un état précédent en enlever la requête de transaction de la «Queue de transactions» dans le cas où l’utilisateur a fait une erreur et décide d’annuler sa requête pas encore traitée.

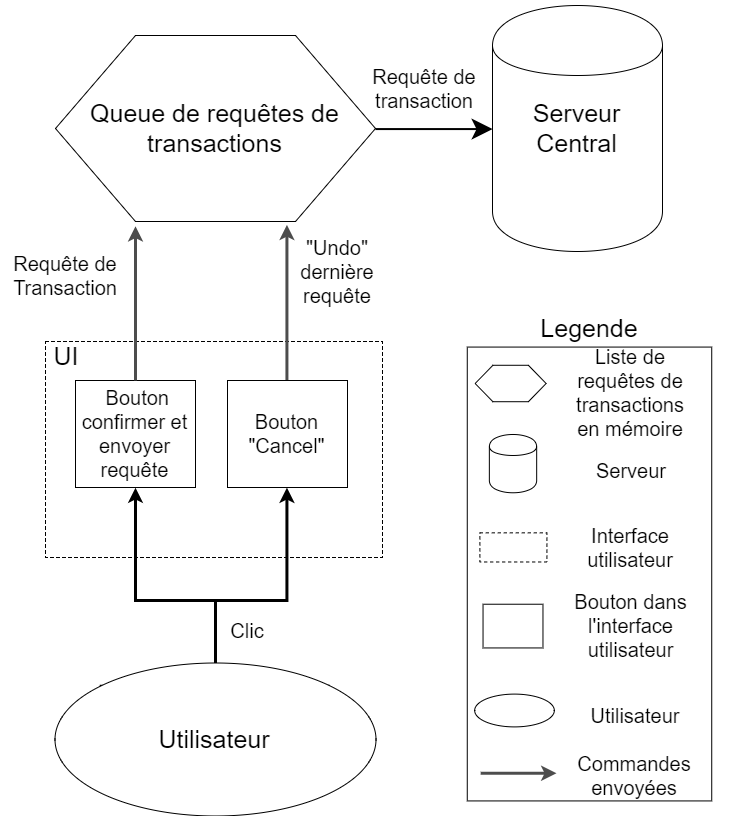
## Tactique 2 : Cancel

**Description**: Ajouter une fonctionnalité «Cancel» pour permettre à l’utilisateur de défaire une action ou une série d’actions effectuée(s) précédemment.

**Justification**: Puisqu’il est possible pour les utilisateurs de changer d’avis ou de faire des erreurs lors de la construction de la requête pour une nouvelle transaction, l’ajout d’un bouton «Cancel Request» à proximité des requêtes pas encore traitées permettrait plus de manoeuvrabilité à l’utilisateur.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Texte de description du diagramme

En temps normal, l’utilisateur effectue ses saisies pour sa requête de transaction et appuie sur le bouton «Confirmer et envoyer requête» pour envoyer la requête de transaction vers la queue de requête de transaction. Cette queue envoie la prochaine requête dans la liste vers le Serveur Central, qui lui s’occupe de faire les communications nécessaires pour compléter la requête de transaction.

Ici, la tactique «Cancel» est modélisée par un bouton «Cancel» dans l’interface utilisateur. L’utilisateur peut appuyer sur le bouton «Cancel» dans l’interface utilisateur pour déclencher l’envoi de la commande «Undo» vers la queue de requêtes de transactions.

### Table de description des éléments du diagramme

| **Élément** | **Description** |
| --- | --- |
| Queue de requêtes de transactions | Liste de requêtes de transactions présentement gardées en mémoires jusqu’à leur complétion. |
| Serveur Central | Serveur physique représentant le système central de communication entre les autres systèmes (base de données, serveurs de traitement de transactions, système boursier, etc.). |
| Interface Utilisateur | Interface avec laquelle l’utilisateur interagit. |
| Boutons dans l’interface utilisateur | Boutons avec lesquels l’utilisateur interagit. |
| Utilisateur | Utilisateur du système boursier. |
| Commandes envoyées | Actions ou appels envoyés aux composants affectés. Par exemple : l’utilisateur effectue des saisies et des clics dans l’interface utilisateur. L’interface utilisateur envoie des requêtes ou des actions de «Undo» par intérim du contrôleur pour supprimer la dernière requête dans la queue, etc.. |

### Texte décrivant la relation entre les éléments et les tactiques

L’utilisateur communique avec le serveur central par intérim du système boursier représenté par le «UI» et les boutons contenus dans celui-ci. Il effectue des opérations génériques de saisies et de clics sur des boutons pour envoyer des actions vers la queue de requêtes de transaction.

Si le bouton de confirmation de requête de nouvelle transaction est appuyé, une commande d’ajout de cette requête dans la queue de requête est envoyée. Si le bouton «Cancel» est appuyé, l’action «Undo» est envoyée et la queue supprime la dernière requête de transaction envoyée.

En parallèle, la queue de requête de transactions continue à passer des requêtes de transactions vers le serveur central, qui lui s’occupe de faire la communication nécessaire pour conclure la transaction.

# Scénario : Panier d’achats

| **Objectifs d'affaires** | Créer un panier d’achats |
| --- | --- |
| **Source** | Utilisateur (client) |
| **Stimulus** | ~~Un utilisateur~~ veut gérer les articles de son panier d’achats. |
| **Artéfact** | Site web (Interface de panier d’achats) |
| **Environnement** | Runtime |
| **Réponse** | * Temps d’exécution * Facilité de navigation |
| **Mesure de la \_ réponse** | * Nombre de cliques fait par l’utilisateur * Temps de réponse |
| **Questions** | 1. Combien d’états seront sauvegardés pour la tactique Undo? |

## Tactique 1 : Cancel

## Description: Permettre d’annuler une transaction du panier d’achats à tout moment.

**Justification:** Si pour quelconque raison il y a une erreur de connexion durant la transaction ou que l’utilisateur réalise qu’il ne voulait pas acheter un article en particulier, la tactique d’annulation va restaurer l’état précédent et annuler son achat.

## Tactique 2 : Undo

**Description:** Garder en mémoire les informations du panier d’achats, afin de permettre à l’utilisateur de revenir à l’état précédent s’il le veut.

**Justification:** Avec la tactique de retour en arrière, si l’utilisateur ajoute ou supprime un article de son panier d’achat, il a la possibilité de faire un retour en arrière pour annuler son erreur.

**Tactique 3 : Pause/Resume**

**Description:** Garder en mémoire les informations du panier d’achats, afin de restaurer le dernier état sauvegardé à la réouverture de l’application.

**Justification:** Avec la tactique Pause/Resume, lorsque l’utilisateur va fermer son navigateur, son panier d’achat ne sera pas perdu, l’état de ce dernier sera sauvegardé et lorsqu’il accédera au site de nouveau, le panier d’achat sera restauré.

**Vue architecturale (à l’extérieur de la classe)**

### Diagramme



### Texte de description du diagramme

L’utilisateur du panier d’achat (*Client*), effectue une action (*CommandeConcrète*) sur son panier d’achat, soit ajouter ou retirer un article, en lui passant l’état actuel de celui-ci. Par la suite, *l’Invocateur* va déclencher la *CommandeConcrète* en passant par l’interface *Commande*. Enfin, la *CommandeConcrète* exécutera l’action qui va modifier le *Récepteur*, avec les paramètres reçu par *Client*.

**Table de description des éléments du diagramme**

Client: Il instancie CommandeConcrète avec l’état de son panier d’achat.

Invocateur: Déclenche l'exécution des commandes.

Commande: Classe abstraite qui découple la création et l’exécution des commandes.

CommandeConcrète: Une commande effectuée par l’utilisateur.

Récepteur: L’objet cible qui sera affecté par la commande.

### 

### 

### Texte décrivant la relation entre les éléments et les tactiques

Tout d’abord, on exécutera à nouveau la commande du dernier état sauvegardé, pour la tactique Cancel. Ensuite, pour la tactique Undo, l’Invocateur nous permet d’annuler ou de rétablir des commandes. Enfin, on restaure le dernier état sauvegardé à l’ouverture de l’application et on la sauvegarde à sa fermeture, pour la tactique resume/pause.

# Scénario : L’utilisateur souhaite consulter les données d’historique de 7 ans pour réaliser un rapport

| **objectifs d'affaires** | L’utilisateur doit pouvoir filtrer les données historiques afin d’avoir un rapport contenant seulement les données dont-il a besoin |
| --- | --- |
| **Source** | L’utilisateur souhaitant générer un rapport |
| **Stimulus** | Essait d’utiliser le système efficacement |
| **Artéfact** | Le système d’historique transactionnel |
| **Environnement** | Runtime |
| **Réponse** | Rendre le système le plus intelligent possible en guidant l’utilisateur et en lui permettant de revenir en arrière. |
| **Mesure de la réponse** | Temps pour effectuer une transaction Nombre d’annulations  Satisfaction des utilisateurs |
| **Questions** |  |

## Tactique 1 : Maintenir le modèle du système

**Description**: Toutes les requêtes nécessitant un temps de chargement affichent un estimé de temps restant à l’utilisateur.

**Justification**: En conservant le modèle du comportement du système, nous pouvons également obtenir le niveau d’avancement des différentes opérations. Ceci nous permet de donner un retour visuel à l’utilisateur, qui pourrait autrement penser que le système a planté lors de longs chargements. Il s’agit donc d’une tactique nécessaire afin de permettre des transactions d’envergure.

## Tactique 2 : Maintenir le modèle de la tâche

**Description**: L’interface de recherche pour la personnalisation du rapport fera usage d'auto complétion intelligente dans tous les champs, qui seront tenus à jour par la technologie ajax.

**Justification**: Étant donné la taille des données, il est nécessaire de faciliter au maximum l’utilisation de celles-ci. Ainsi, cette tactique est excellente pour notre scénario, puisqu’elle permet d’orienter l’utilisateur étape par étape dans sa recherche.

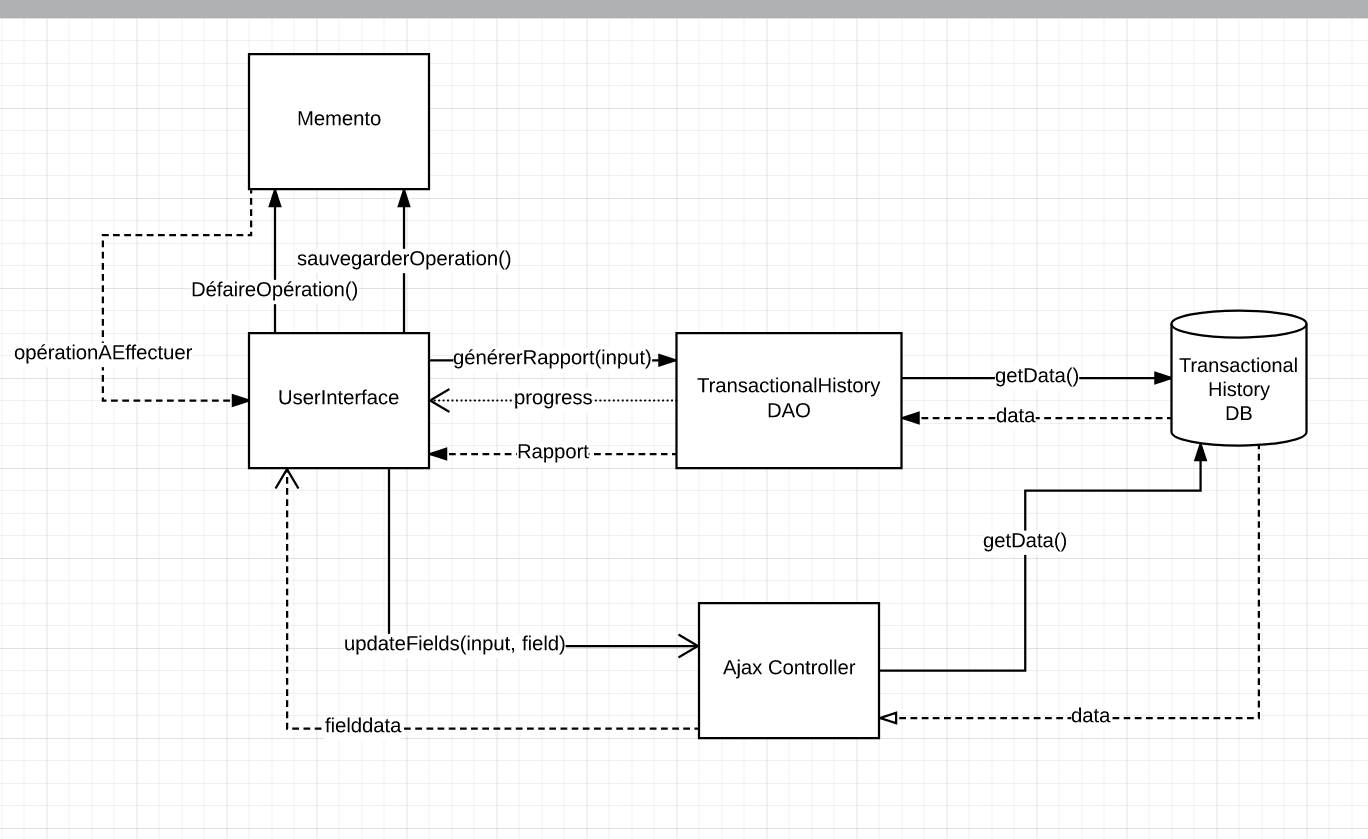
## Tactique 3 : Undo

**Description**: À tout moment lors de la préparation du rapport, l’utilisateur peut annuler la dernière opération afin de revenir à l’état précédent

**Justification**: Lors de toute opération, il est nécessaire d’offrir l’option d’annulation. En effet, une mauvaise manipulation ajoutant une centaine de lignes de données non désirées à un rapport étant possible, l'option d’annulation permet de donner une tranquillité d’esprit à l’utilisateur, qui peut annuler une erreur sans perdre son travail.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende

Les carrés représentent des composants

Les flèches à ligne pleine représentent une opération synchrone.

Les flèches vides à ligne pleine représente une opération asynchrone

Les lignes à point tirés représentent un retour, avec la même convention que pour les opérations

Le cylindre représente une base de données

### Texte de description du diagramme

Le diagramme présente les trois tactiques. La page affichée sur l’interface utilisateur a plusieurs éléments interactifs. Ceux connectés à des requêtes ajax (asynchrones) font directement affaire avec le module ajax, qui gère les requêtes de base de données nécessaires afin d’alimenter les champs de façon intelligente, tel que présenté dans la seconde tactique. La requête de génération de rapport déclenche l’affichage d’une barre de progression sur l’interface utilisateur, montrant à l’utilisateur un estimé du temps restant nécessaire afin de compléter la génération du rapport. Enfin, chaque opération faite sur l’interface principale est sauvegardée à l’aide du patron memento. Ainsi, l’utilisateur peut revenir en arrière aisément si il fait une mauvaise manipulation.

### Table de description des éléments du diagramme

| Élément | Description |
| --- | --- |
| UserInterface | Représente la page de l’utilisateur et son contrôleur |
| Memento | Gère la conservation en mémoire des actions utilisateurs, afin de pouvoir assurer leur annulation |
| DéfaireOpération() | Défait la dernière opération sauvegardée au memento |
| sauvegarderOpération() | Sauvegarde la dernière opération effectuée par l’utilisateur |
| opérationAEffectuer | l’opération que l’interface doit effectuer afin de revenir à son état précédent. |
| TransactionalHistoryDAO | Gère les accès à la base de données d’historique de transactions. |
| générerRapport(input) | L’action de l’interface utilisateur qui lance la génération du rapport avec les paramètres sélectionnés. |
| progress | Le progrès en cours de l’opération de génération de rapport. Est envoyé de façon asynchrone, et est affiché sous forme de barre de progrès. |
| Rapport | Le rapport généré |
| getData() | Fonction quelconque d’accès à la base de données |
| data | Données quelconques provenant de la base de données |
| fielddata | Données formatées par le contrôleur ajax afin d’être consumée par l’interface utilisateur pour la population des champs. |
| AjaxController | Sert à gérer les requêtes ajax faites par l'interface utilisateur. Ce contrôleur recherche le contenu, le formate puis le retourne à l’interface utilisateur. |

### Texte décrivant la relation entre les éléments et les tactiques

UserInterface, Memento et ce qui les relient sont associés à la tactique 3, undo. UserInterface, étant la vue, est aussi présent avec TransactionalHistoryDAO, générerRapport(), progress afin d’expliquer la tactique 1, soit le maintien du modèle système. Enfin, Le Ajax Controller et ce qui le relie à la base de donnée et à l’interface utilisateur présente la tactique 2, soit le maintien du modèle de la tâche.

# Scénario : L’utilisateur désire automatiser ses transactions

| **Objectifs d'affaires** | L’utilisateur désire entrer un montant et dire au système de déclencher une transaction lorsque l’action atteint le prix indiqué |
| --- | --- |
| **Source** | L’utilisateur utilisant le système de bourse |
| **Stimulus** | ~~L’utilisateur~~ veut configurer le système à faire des transactions automatiquement. |
| **Artéfact** | Session courante (implique les données et services du système) |
| **Environnement** | Le système est en ligne et sert le client authentifié |
| **Réponse** | Un “trigger” sur le montant d’une action est mise en place. |
| **Mesure de la réponse** | L’utilisateur est en mesure d’automatiser ses transactions. |
| **Questions** | 1. Est-ce qu’il est possible d’automatiser plusieurs transactions ? 2. Est-ce qu’un message avertira l’usager que la transaction a été effectuée lorsque l’action a atteint le prix indiqué ? 3. Faudrait-il instaurer un système de sécurité pour s’assurer que la transaction est vraiment voulu, avant de l’effectuer ? |

## Tactique 1 : Maintain a model of the task

**Description** : Principalement, cette tactique permet au système de posséder un certain automatisme qui permet de rouler des tâches lorsque désiré.

**Justification** : Puisque nous voulons aider le client à effectuer des tâches plus rapidement et efficacement, avec cette automatisation de transaction, nous lui permettons de planifier des transactions en fonction d’un cap qu’une action doit atteindre avant d’effectuer cette transaction. Ceci permet donc à l’utilisateur de profiter de meilleurs transactions à de meilleurs moments.

# Vue architecturale

Afin de détailler cette tactique architecture, voici les étapes que devra faire le système pour effectuer les transactions automatiquement.

1- Itérer à travers la liste d’action

2- Est-ce que l’action est surveillé par des “tasker”

3- Si oui, vérifier si l’action a passé le seuil d’achat de l’action (établi selon l’utilisateur)

4- Si oui, effectuer la transaction

5- Garder des traces des transactions effectuées

## Liens entre les vues et les tactiques

La vue présenté ci-dessus représente textuellement la tactique de maintenance d’un modèle de tâche. Puisqu’il est dur de représenter cette tactique visuellement, une simple liste de tâche permet de faciliter la compréhension de la tactique. Cette liste de tâche représente en fait ce que le système effectuera comme vérification afin d’appliquer la tactique voulu, qui est de maintenir une certaine automatisation dans la transaction d’action.

# Scénario : Un utilisateur désire acheter des actions de plusieurs entreprises à travers l’application.

| **objectifs d'affaires** | Permettre à un acheteur d’acheter des actions de façon à minimiser le taux d’erreur éventuelle |
| --- | --- |
| **Source** | L’utilisateur |
| **Stimulus** | ~~L’utilisateu~~r désire acheter des actions. |
| **Artéfact** | La plateforme d’achat et gestion d’action en ligne |
| **Environnement** | En opération normale au runtime |
| **Réponse** | Le système fournit à l’utilisateur une fonctionnalité permettant d’annuler l’achat d’actions avant le paiement. Le système permet à un utilisateur d’acheter plusieurs actions en même temps. Le système permet à l’utilisateur de décider s’il veut que sa carte de crédit soit utilisé pour les achats subséquents. |
| **Mesure de la réponse** | Le nombre d’erreurs effectuées par l’utilisateur.  La satisfaction de l’utilisateur |
| **Questions** | 1. Quelles sont les erreurs commises le plus fréquemment par l’utilisateur ? 2. Combien de temps la tâche d’acheter des actions prend-elle en moyenne ? 3. Comment optimiser l’expérience de l’utilisateur ? |

## Tactique 1 : Cancel

**Description**: Annule une commande initiée par l’utilisateur.

**Justification**: Quand l’utilisateur sélectionne une ou plusieurs actions à acheter, il peut décider d’annuler l’achat de ces actions avant que le paiement ne soit effectué.

## Tactique 2 : Aggregate

**Description**: Permet d’effectuer une commande sur plusieurs éléments en même temps.

**Justification**: Le système permet d’acheter plusieurs actions en même temps afin d’éviter la tâche longue et répétitive qui consisterait à les acheter séparément, ainsi que pour réduire les risques d’erreurs lorsqu’on entre les données en relation avec le paiement.

## Tactique 3 : Maintain user model

**Description**: Garde en mémoire des éléments relatifs à l’utilisateur pour faciliter ses utilisations subséquentes.

**Justification**: À la suite d’un achat, le système demande à l’utilisateur s’il désire que le système conserve ses informations de paiement pour ses achats subséquents. Si l’utilisateur accepte alors l’utilisateur sera en mesure d'acheter d’autres actions sans avoir à ré-entrer toutes ses informations à nouveau.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme (Vue utilisateurs)

Diagramme A

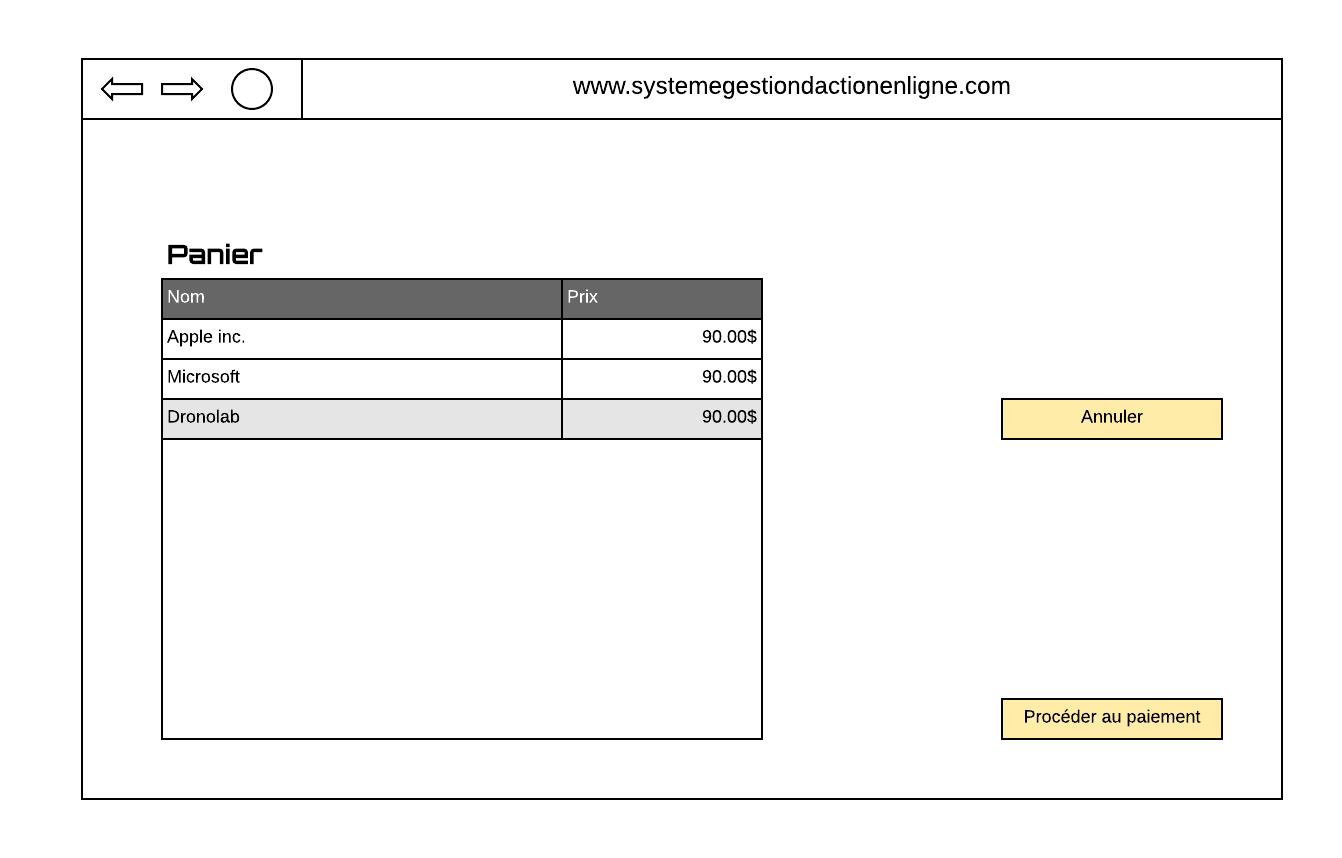


Diagramme B

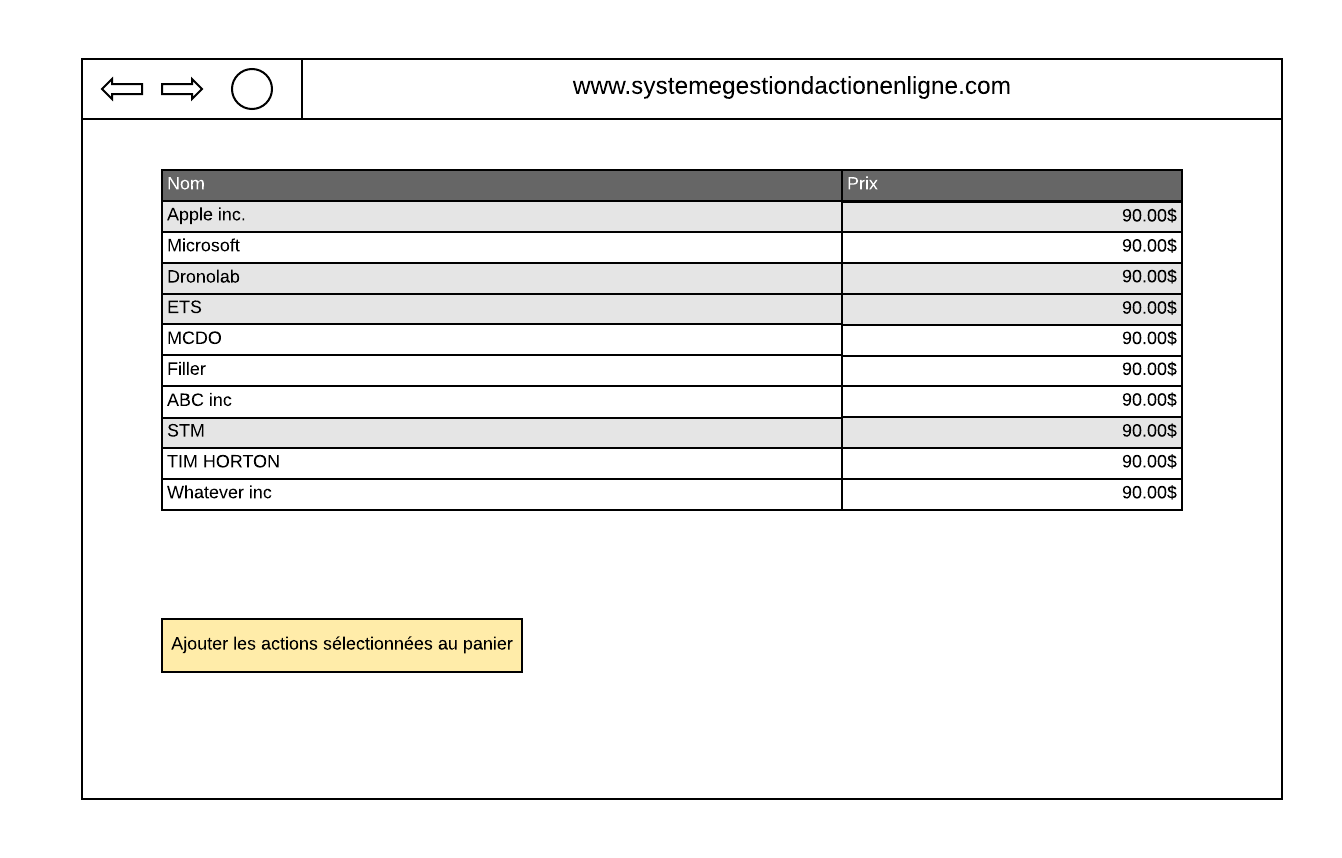
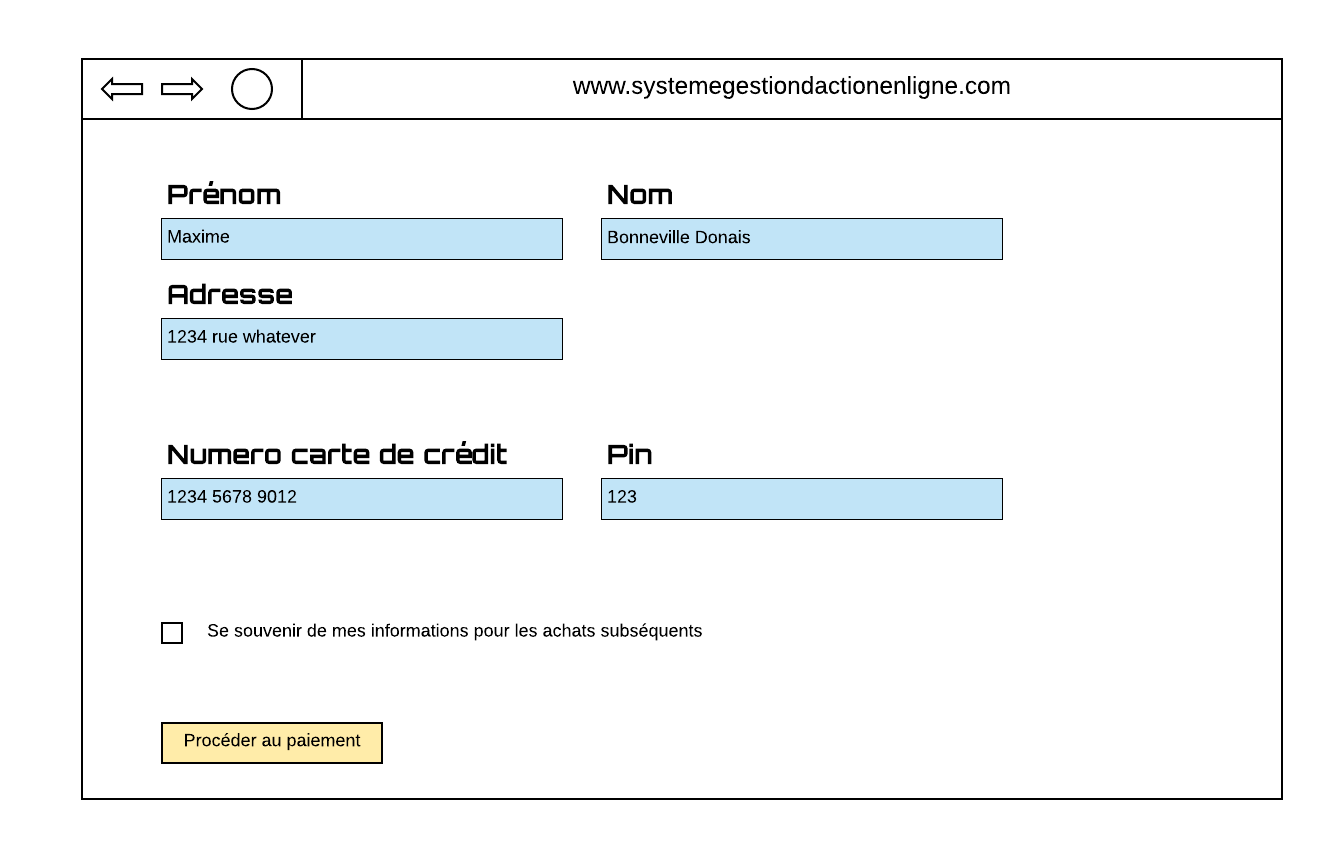
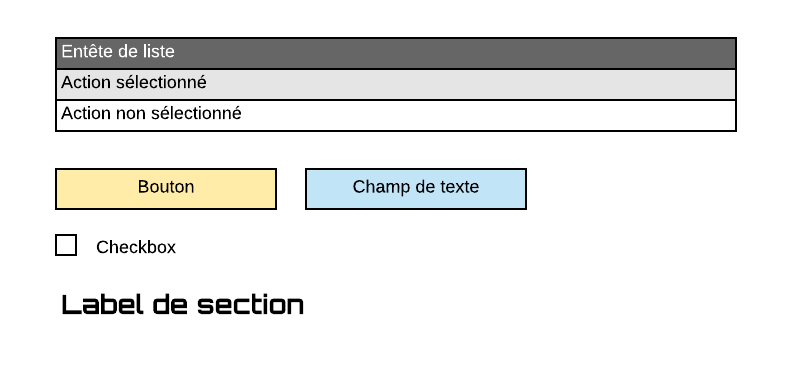


Diagramme C



### Légende



### Texte de description du diagramme

Dans le diagramme A nous pouvons voir que l’utilisateur utilise un panier pour stocker les actions qu’il désire acheter. De ce panier il peut sélectionner une (ou plusieurs) actions et les annuler, Ceci permet à l’utilisateur d’annuler des actions ajoutées par accident à son panier avant de procéder à l’achat.

Dans le diagramme B nous pouvons voir que l’utilisateur peut sélectionner plusieurs actions (en gris) et les ajouter à son panier.

Dans le diagramme C nous pouvons voir que l’utilisateur peut cocher la case ‘se souvenir de mes informations pour les achats subséquents’ ce qui permettra au système de remplir les champs du formulaire automatiquement lors des prochains achats réalisés par cet utilisateur..

### Table de description des éléments du diagramme

Non applicable.

### Texte décrivant la relation entre les éléments et les tactiques

Le bouton Annuler du diagramme A est en lien direct avec la tactique ‘Cancel’. La possibilité de choisir plusieurs actions dans le diagramme B pour toutes les ajouter au panier en même temps est en lien direct avec la tactique ‘Aggregate’. Le fait que le système puisse se souvenir des informations de paiement de l’utilisateurs dans le diagramme C est en lien avec la tactique ‘Maintain user model’.

# Scénario : Permettre de suspendre et résumer une transaction en bourse.

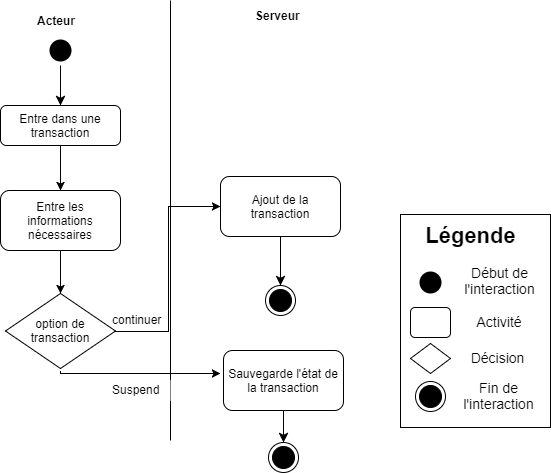
| **Objectifs d’affaires** | Permettre de suspendre une transaction et de revenir plus tard en gardant les informations en mémoire. |
| --- | --- |
| **Source** | un “Trader” |
| **Stimulus** | Mettre une transaction en attente |
| **Artéfact** | Système de transactions |
| **Environnement** | Runtime |
| **Réponse** | La transaction suspend la transaction et offre la possibilité de résumer celle-ci |
| **Mesure de la réponse** | * Le temps que le trader a pour suspendre la transaction avant qu’elle soit complété. * Le trader a la possibilité d’annuler la transaction. |
| **Questions** | * Quels modifications sur la transaction peut faire le trader pendant que la cette dernière est en arrêt. * Dans quels circonstances ne peut-on suspendre une transaction? * Pendant combien de temps une transaction suspendue est gardée en mémoire? |

## Tactique 1 : Pause/Resume

**Description**: Permet de suspendre un état pour pouvoir résumer celui-ci à un moment opportun.

**Justification**: L’utilisateur peut suspendre une transaction s’il désire attendre le bon moment pour résumer la transaction à un meilleur prix.

**Vue Architecturale**



### Texte de description du diagramme

Un acteur, probablement un “trader” entre dans une transaction (Une transaction déjà existante ou non). Il entre les informations nécessaires pour compléter une transaction puis le système lui offre la possibilité de continuer la transaction ou bien la suspendre. Si l’acteur décide de la suspendre, elle le système sauvegarde l’état de la transaction (les informations entrées par l’utilisateur sont sauvegardées) et l’acteur peut y revenir plus tard. Si l’acteur choisi de continuer, la transaction est ajoutée par le système de transaction.

### Table de description des éléments du diagramme

| **Acteur** | Entité voulant ajouter une transaction qui pourra être suspendue ou résumée |
| --- | --- |
| **Serveur** | Reçoit la demande d’ajout de transaction et traite l’événement de suspension de celle-ci. |

### Texte décrivant la relation entre les éléments et les tactiques

L’acteur est celui qui peut déclencher un événement tel que la suspension d’une transaction ainsi que de la résumer. Le serveur est celui qui gère l’événement déclenchée par l’acteur. Le serveur s’occupe donc de traiter la suspension d’une transaction en la gardant en mémoire. De plus, à la demande de l’acteur une transaction peut être résumée et c’est le serveur qui entame l’ajout de la transaction.

# Scénario : Permettre à un utilisateur d’utiliser des gabarits de transaction.

| **Objectif d'affaires** | Accélérer la complétion d’une transaction |
| --- | --- |
| **Source** | Client |
| **Stimulus** | Utiliser le système efficacement |
| **Artéfact** | Le formulaire de création des transactions |
| **Environnement** | Temps d’exécution |
| **Réponse** | Le client n’a pas à remplir tous les champs du formulaire pour chaque transaction. |
| **Mesure de la réponse** | Le nombre de champs à remplir pour compléter une transaction. |
| **Questions** | 1. Combien de champs comporte un formulaire de création de transaction? 2. Quelles informations sont redondantes dans une transaction? |

## Tactique 1 : Undo

**Description** : Permet à un utilisateur de défaire les modifications liées à la sélection d’un gabarit.

**Justification** : Dans le cas d’une transaction, si un utilisateur fait une erreur lors de la sélection d’un gabarit ou si celui-ci ne désire pas utiliser de gabarit, il doit être en mesure de défaire ou modifier les ajouts automatiques apportés par un gabarit.

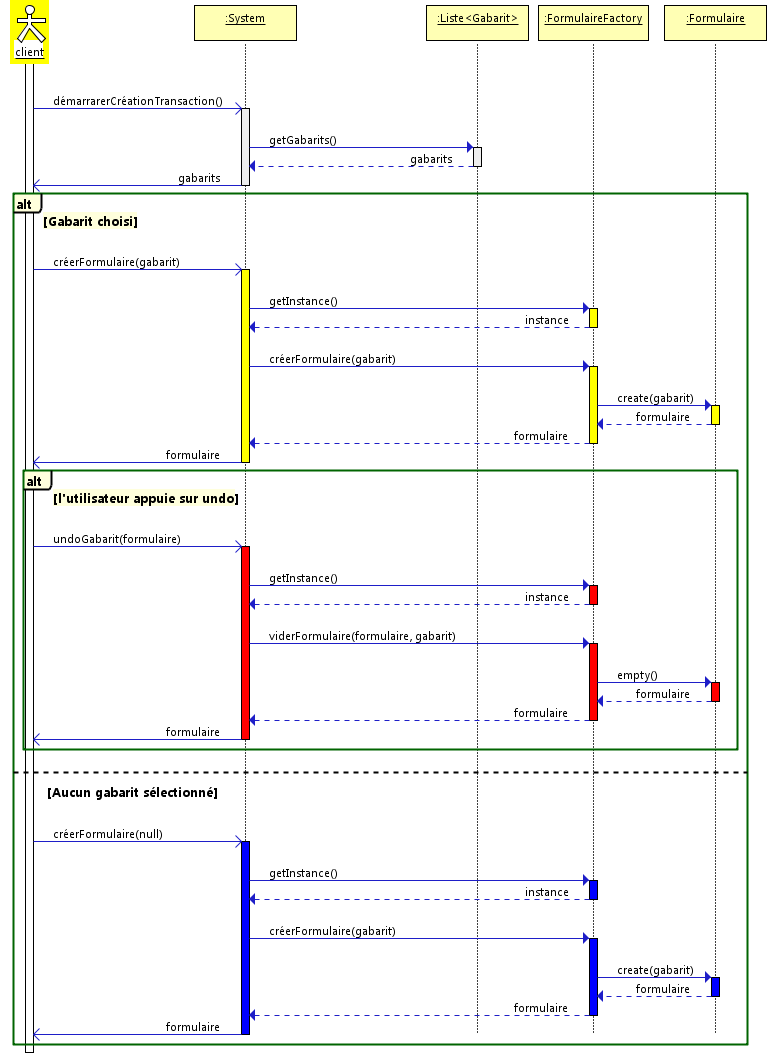
## Tactique 2 : Maintain User Model

**Description** : Permet à un utilisateur d’éviter de répéter certaines actions courantes en laissant le système enregistrer ces actions et les répéter au bon moment.

**Justification** : L’utilisateur pourra créer un gabarit de transaction contenant des informations récurrentes comme un nom d’entreprise. En appliquant un gabarit sur un nouveau formulaire, certains champs seront remplis automatiquement pour accélérer la complétion de la transaction.

## Vue architecturale

### Diagramme

\*\*Aucune légende n’est fournie puisqu’il s’agit d’un diagramme de séquence UML.\*\*

### Texte de description du diagramme

Le diagramme présente l’application d’un gabarit à un nouveau formulaire et sa « désapplication » par la commande *undo*.

### Table de description des éléments du diagramme

| **Élément du diagramme** | **Description** |
| --- | --- |
| Client | Utilisateur du **système**. |
| Système | Système boursier. |
| Liste<Gabarit> | Liste logicielle des gabarits du **client**. |
| Formulaire | Ensemble des champs de texte à remplir pour compléter une **transaction**. |
| Transaction | Transaction boursière complétée par le **client** à l’aide du **formulaire**. |
| FormulaireFactory | Permet de créer et de modifier des **formulaires**. |

### Texte décrivant la relation entre les éléments et les tactiques

La **liste de gabarits** permet à un **client** de sélectionner l’un de ceux-ci pour l’appliquer à un **formulaire** de **transaction**, ce qui maintient un modèle utilisateur et réduit le nombre de champs à remplir pour compléter une **transaction**. À tout moment, le **client** peut décider d’effacer ou modifier les champs affectés par un **gabarit**, ce qui permet à l’utilisateur de revenir sur ses pas en cas d’erreur, un exemple typique de la tactique *undo*.

# Scénario : L’utilisateur veut créer un compte pour accéder aux fonctionnalités du marché boursier

| Objectifs d'affaires | On veut faciliter l’accès aux fonctionnalités du marché boursier en conservant les informations de l’utilisateur de manière à éviter la redondance lors de l’utilisation. |
| --- | --- |
| Source | Un utilisateur quelconque. |
| Stimulus | Le système s’adapte avec les informations et les paramètres de l’usager. |
| Artéfact | Le système du marché boursier. |
| Environnement | Runtime. |
| Réponse | L’anticipation des besoins de l’utilisateur |
| Mesure de la réponse | * Réduction du temps lors de l’exécution de la tâche. * Réduction du nombre d’erreur. |
| Questions | 1. Devrait-on réduire le nombre de paramètre à entrer lors de la complétion du formulaire ? 2. Comment pourrait-on réduire le nombre d’erreur ? |

# 

## Tactiques

## Tactique 1 : Support System Initiative - *Maintain user model*

Description : Le modèle d’usager représente les connaissances d’un usager par rapport au système actuel ainsi que ses habitudes et autres. On contrôle ainsi plus facilement l’assistance à fournir à l’utilisateur dans l’accomplissement de ses tâches.

Justification : Pouvant considérer les actions des utilisateurs, il sera possible de placer les éléments de la page associés aux habitudes de celui-ci. Ainsi, la possibilité de garder certaines données liées au compte de l'utilisateur pourrait aussi être pris en compte, par exemple, l'auto complétion de certains formulaires entrés précédemment. Ceci apporterait un support à l’utilisateur quant à la rapidité d’exécution. On rend la plateforme beaucoup plus facile à utiliser et les tâches sont réalisées en favorisant les normes d’utilisabilité en visant l’efficience et la satisfaction.

## Tactique 2 : Support User Initiative - *Aggregate*

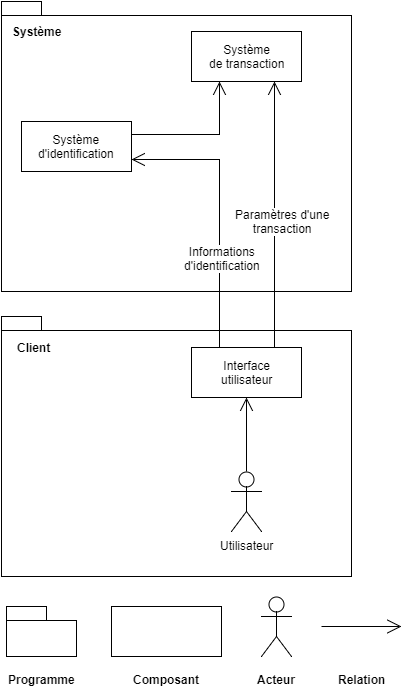
Description : On regroupe des opérations répétitives en une seule entité de manière à appliquer des opérations sur un groupe. Ainsi, on réduit les chances d’erreurs et d’actions répétées inutilement.

Justification : En étant déjà connecté, le système agrège les informations en lien avec la transaction que l’usager veut effectuer. On limite ainsi la répétition dans la récupération des informations lorsqu’un utilisateur désire entreprendre une démarche de transaction au marché boursier. Au final, on simplifie grandement les interactions en limitant le nombre de fois qu’un utilisateur doit entrer ses informations pour effectuer une manipulation sur le marché et ce peu importe le statut de l’usager.

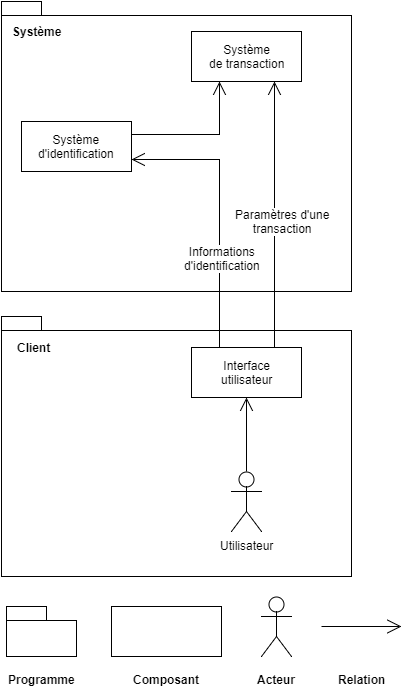
## 

## Vue architecturale

### Diagramme



### Légende



### Texte de description du diagramme

Le diagramme est composé de deux sections: le programme client et le programme système. Le programme client montre la relation entre l’utilisateur et l’interface du programme qui lui permet de faire des transactions. Le programme système est le système en lui-même qui gère le traitement des transactions reçues par les requêtes du programme client. Les relations entre les deux programmes décrivent les informations envoyées pour effectuer une transaction. D’une part, l’utilisateur s’inscrit au système en entrant les données d’identifications qui sont propres à l’utilisateur et requises lors de chaque transaction. De cette façon, l’utilisateur n’a qu’à rentrer ces informations une seule fois dans le système, ce qui évite la redondance. D’autre part, les paramètres d’une transaction spécifique sont demandés à l’utilisateur pour chaque transaction. Toutes ces informations sont ensuite acheminées vers le système afin d’effectuer la transaction avec tous les paramètres en main pour pouvoir la traiter entièrement.

### Table de description des éléments du diagramme

| **Élément** | **Responsabilité** |
| --- | --- |
| Interface utilisateur | Ceci correspond à l’affichage. L’utilisateur peut voir ce qui est dans l’interface et ainsi agir en conséquence. |
| Système de transaction | Le système de transaction est responsable de produire la transaction. Celle-ci reçoit le formulaire envoyé par l’utilisateur et procède à la vente ou l’achat d’une action. |
| Système d’identification | Permet d’identifier l’utilisateur et stocker toutes les informations communes à toutes les transactions de cet utilisateur. |

### 

### Table décrivant la relation entre les éléments et les tactiques

| **Tactique** | **Description** |
| --- | --- |
| Support System Initiative - *Maintain user model* | L’interface utilisateure contient les éléments utilisés fréquemment par l’utilisateur. Celle-ci sont gardées à jour en permanence en communiquant avec le système de transaction ou bien avec le système d’identification. |
| Support User Initiative - *Aggregate* | Le système d’identification permet de réduire les informations entrées de la part de l’utilisation. Celle-ci sont alors réutilisées lors de l’exécution d’une transaction. Ce qui en découle d’une augmentation de la vitesse d’exécution pour des transactions courantes. |

# Scénario : Un actionnaire de la bourse modifie une transaction en cours

| **objectifs d'affaires** | Permettre à l’actionnaire d’avoir un contrôle total sur la transaction en cours et de savoir exactement les comportements de transactions |
| --- | --- |
| **Source** | L’actionnaire |
| **Stimulus** | Modification d’une transaction |
| **Artéfact** | La transaction en cours |
| **Environnement** | En temps réel |
| **Réponse** | La transaction est modifiée |
| **Mesure de la réponse** | Satisfaction de l’utilisateur  Action réussit |
| **Questions** | 1. Une transaction peut-elle être annulée après l’envoie? 2. Une transaction peut-elle être modifiée après l’envoie? 3. Une transaction mets combien de temps pour être envoyée? 4. Des retours à l’utilisateur se font-ils lors de la transaction et après son envoie? |

## Tactique 1 : Cancel

**Description**: Permet d’annuler la transaction en cours

**Justification**: L’actionnaire pourra alors annuler une transaction, lorsque des doutes se présente sur la validité de son choix après l’envoie de la transaction

## Tactique 2 : Undo

**Description**: Permet de revenir en arrière et changer les infos d’une transaction, sans l’annuler

**Justification**: L’actionnaire pourra alors changer un champ de la transaction, décider s’il veut ou bien achète des actions, le montant, etc.

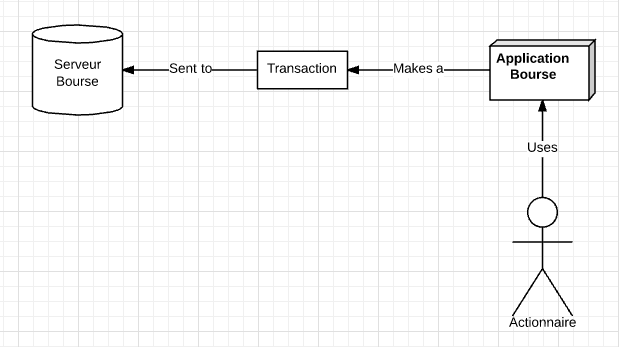
## Tactique 3 :Maintain System Model

**Description**: Permet de donner un retour visuelle utilisé pour déterminer les comportements prévus du système

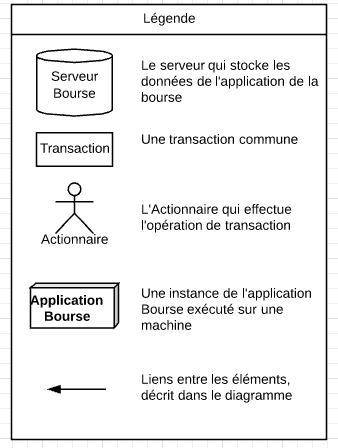
**Justification**: Va permettre à l’actionnaire de voir la progression d’une transaction en cours

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende



### Texte de description du diagramme

L’actionnaire utilise une instance de l’Application Bourse afin de faire une transaction qui va ensuite être transmise au serveur à des fins de traitements/stockages

### Table de description des éléments du diagramme

| **Éléments** | **Description** |
| --- | --- |
| Serveur Bourse | Ceci est le serveur de l’application. Toutes les transactions sont acheminées vers le serveur. |
| Application Bourse | Une instance de l’application qui est utilisée afin de faire des transactions. Est utilisée par l’utilisateur afin d’être utilisé et effectuer les opérations liées aux transactions. |
| Actionnaire | Un actionnaire qui interface avec l’application afin de faire des transactions |
| Transaction | Une transaction qui a été effectué par l’actionnaire. Peut être annuler, revenir en arrière pour les champs remplie. |

### Texte décrivant la relation entre les éléments et les tactiques

L’actionnaire qui utilise l’application Bourse peut décider à tout moment d’annuler sa transaction (Cancel), ou bien faire un undo afin de modifier un champ (Undo). Lorsque la transaction est envoyé au serveur, une barre de progression est affiché afin de tenir au courant l’actionnaire de l’état de la requête (Maintain System Model)

# Scénario 1: Le client veut pouvoir annuler une transaction.

| **Objectifs d'affaires** | Offrir un meilleur service aux clients en leur permettant retirer une transaction. |
| --- | --- |
| **Source** | Client du système boursier |
| **Stimulus** | ~~L’utilisateur~~ veut pouvoir annuler des transactions. |
| **Artéfact** | Système boursier. |
| **Environnement** | Runtime |
| **Réponse** | Le système va offrir une nouvelle fonctionnalité pour annuler une transaction. |
| **Mesure de la réponse** | * Temps requis pour annuler une transaction. * La satisfaction du client |
| **Questions** | 1. Combien de transactions seront annulées par semaine? |

**Tactique 1: Cancel**

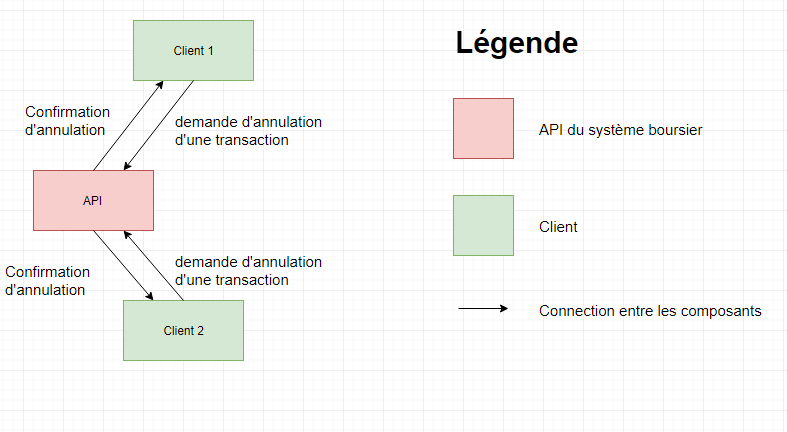
**Description:**

Cette tactique permet aux clients du système boursier d’annuler des transactions.

**Justification**:

Nos clients vont pouvoir annuler des transactions pour leurs utilisateurs, ce qui va augmenter la qualité du service qu’ils offrent à leurs clients.

### Diagramme et légende



### Texte de description du diagramme

Notre diagramme représente la façon dont les interfaces de nos clients vont interagir avec l’API pour annuler une transaction et pour recevoir une confirmation de l’annulation.

### Table de description des éléments du diagramme

| **Élément du diagramme** | **Fonction** |
| --- | --- |
| API | API du système boursier qui est utilisé par les clients |
| Client | Client utilisant le système |
| Relation demande d’annulation d’une transaction | Cette relation est créée lorsque le client fait une demande d’annulation d’une transaction |
| Relation confirmation d’annulation | L’API retourne une confirmation d’annulation d’une transaction au client |

### Texte décrivant la relation entre les éléments et les tactiques

L’API offre une fonction cancel qui est accessible par nos clients via leur interface. Ceux-ci peuvent donc simplement annuler des transactions qui ne sont pas encore traitées.