CHUI Jacky
GAUVAIN Lucas

EISE 4

Compte rendu C++

Billetterie des JO de Paris 2024

Partie 1 : Description de l'application

Notre application est une plateforme de billetterie dédiée aux Jeux Olympiques, dans laquelle se trouvent différentes fonctionnalités. Les utilisateurs peuvent facilement consulter les événements actuels et à venir, acheter des billets, et annuler leurs achats si nécessaire. Les événements disponibles à la réservation sont des compétitions sportives ainsi que des cérémonies.

Certains utilisateurs bénéficient de privilèges VIP, qui leur donnent accès à des événements exclusifs. De plus, des administrateurs sont disponibles pour assister les clients, que ce soit pour la modification de leur compte ou pour ajuster certains évènements.

L'application offre également une fonctionnalité de suivi des actualités, permettant aux utilisateurs de rester informés sur les derniers résultats en direct. Ainsi, notre application vise à simplifier l'expérience de billetterie tout en offrant des fonctionnalités pour suivre les actualités des Jeux Olympiques de Paris 2024.

Partie 2 : Mettre en valeur l'utilisation des contraintes

Avant de réaliser notre application, plusieurs contraintes que nous devions respecter nous ont été imposées. Nous les avons donc prises en compte lors de l'écriture du code. Parmi elles se trouvent :

- Au moins 8 classes : Pour pouvoir réaliser notre application, les deux classes abstraites nous sont apparues rapidement : "événement" et "utilisateur". Ensuite les sous classes nous sont venues après un peu de réflexion. Tout cela nous a amené à valider les contraintes sur le nombre de classes. En effet, notre application contient exactement 8 classes (cf. diagramme UML dans la partie suivante de ce rapport).

- 3 niveaux de hiérarchie : Comme présenté dans le diagramme UML ci-dessous, notre application contient 3 niveaux de hiérarchie, principalement au niveau des utilisateurs, avec une classe principale "Utilisateur" qui présente deux sous classe "UtilisateurPublic" et "UtilisateurVIP". Cette dernière classe "UtilisateurVIP" possède une sous classe "UtilisateurAdmin".

- 2 fonctions virtuelles différentes et utilisées à bon escient : Nous avons été contraints d'utiliser différentes fonctions virtuelles dans notre application, que ce soit au niveau des utilisateurs comme au niveau des événements. Au niveau des utilisateurs, les fonctions virtuelles permettent notamment d'acheter ou de consulter une liste de billets. Ces derniers pouvant parfois être réservés que par des personnes de type VIP,

nous avons implémentés des méthodes spécifiques pour ces utilisateurs. De plus, au niveau des événements, une fonction getter virtuelle permet de retourner si un événement est un événement sportif ou une cérémonie.

- 2 surcharges d'opérateurs : Dans l'optique d'améliorer l'utilisation de l'application, au niveau des utilisateurs, nous avons écrit deux surcharges (== et <<).
    - La surcharge d'opérateur '==' permet de tester si deux utilisateurs sont égaux. Cette fonctionnalité pourrait s'avérer particulièrement utile si nous avions programmé un système de création de compte pour tester si un utilisateur existait déjà dans notre base de données.
    - La seconde surcharge d'opérateurs '<<' permet d'afficher directement toutes les caractéristiques d'un utilisateur. Cette fonction évite de devoir écrire tous les getters à chaque fois que l'on veut afficher un utilisateur.

- 2 conteneurs différents de la STL : Nous avons utilisé deux type de conteneurs différents :
    - Vector : Chaque utilisateur possède une liste d'événements qui est mise à jour à chaque achat ou annulation de billet. De plus, pour simplifier l'écriture de l'application SFML, des *vectors* ont souvent été utilisés pour raccourcir le code, notamment pour afficher tous les tickets achetés et leurs boutons correspondants.
    - Map : Nous n'avons pas jugé essentiel le fait de rajouter des *maps* dans nos classes. Ainsi, nous avons utilisé des *maps* dans la partie application SFML, au niveau de l'affichage des événements sur la page de consultation des évènements en cours et à venir. Les événements créés à afficher sont ajoutés soit dans la *map* des événements en direct, soit dans la *map* des événements à venir. Nous affichons ensuite ces deux *maps* d'événements.

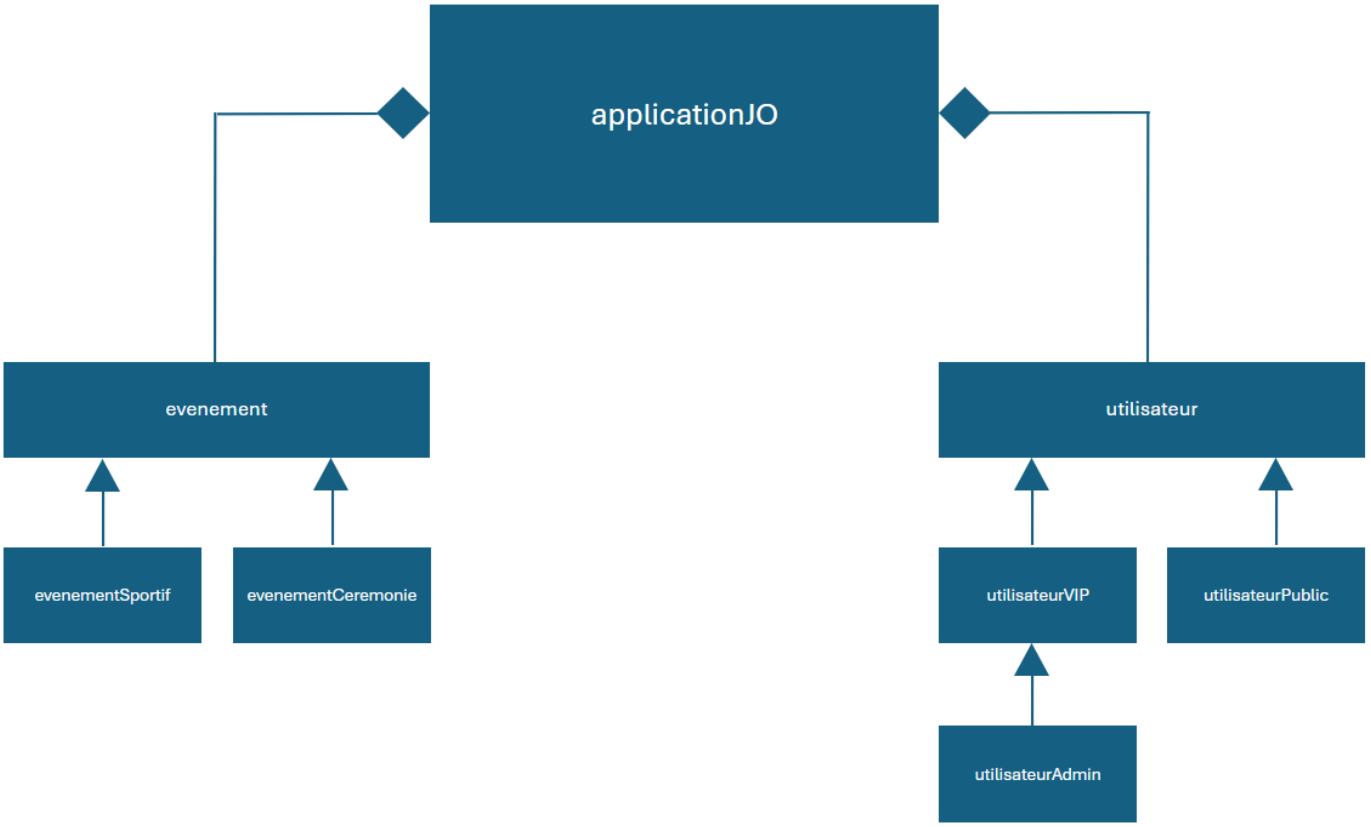
- Pas de méthodes/fonctions de plus de 30 lignes : En dehors de notre fonction SFML ou chaque page correspond à une fonction pour ne pas surcharger le *main()*, nous n'avons pas de fonction ni de méthodes de plus de 30 lignes dans nos classes.

- Utilisation de tests unitaires : Pour tester le bon fonctionnement de nos méthodes dans nos classes, nous avons écrit des tests pour vérifier si nous obtenons bien les résultats que nous attendons. Les tests se trouvent dans le dossier *tests*.

- Utilisation d'un Makefile avec une règle "all" et une règle "clean" : Nous avons un makefile dans le dossier de nos sources qui possède une règle "all" et une règle "clean". Ce makefile compile le code et produit l'exécutable *applicationJO.out*. Au niveau des tests, nous avons un autre makefile dans le dossier de tests qui permet de compiler et d'exécuter les tests avec l'exécutable *testcase.out*.

- En plus : Nous avons utilisé des template au niveau du code SFML de la page des utilisateurs. En effet, si ces derniers achètent beaucoup de billets, l'affichage des billets ne tiendra pas sur une seule page et donc nous avons écrit une fonction qui permet de scroller vers le haut ou vers le bas. Les fonctions vont donc déplacer tous les objets vers le haut ou vers le bas peu importe s'il s'agit de boutons ou de texte.

Partie 3 : Diagramme UML



Partie 4 : Procédure d'installation et exécution du code

Pour pouvoir utiliser notre application graphique, il faut installer les dépendances concernant SFML. Pour cela, sous Linux il suffit de taper dans un terminal la commande suivante :

sudo apt-get install libsfml-dev

Ensuite, pour compiler le projet, un makefile a été fait. Pour compiler, dans le dossier projet ou se trouve tous les fichiers sources entrer la commande *make.* La commande va ensuite générer l'exécutable *applicationJO.out* que l'on peut exécuter en faisant *./applicationJO.out*.

Pour effectuer les tests, il faut se rendre dans le dossier *tests*. A l'intérieur de ce dossier se trouve un autre makefile. Compiler en exécutant la commande *make,* puis on peut lancer les tests en exécutant l'exécutable *testcase.out*, à l'aide de la commande *./testcase.out*.

Partie 5 : Les parties d'implémentation dont nous sommes le plus fiers

Après avoir réalisé ce projet, nous sommes particulièrement fiers de l'interface graphique que nous avons fait avec SFML. En effet, nous avons découvert SFML lors de ce projet donc nous n'avions pas du tout l'habitude de l'utiliser et il s'agit de notre première interface avec cet outil. Nous avons ainsi pu découvrir que SFML est un outil assez puissant avec beaucoup de fonctionnalités graphiques.

De plus, nous sommes également fiers du sujet que l'on a choisi à travers le thème des jeux olympiques car, réaliser une application de billetterie et de suivi d'actualités pourrait s'avérer être utile dans la réalité.