Billetterie Lien du site, Lien du trello, Lien de travis-ci

Il s'agit d'un site de réservation pour une association en lien avec le théâtre.
Billeterie est un site de vente en ligne de billet de spectacles/théâtre.
Enumération des fonctionnalités :

- consulter le catalogue
- la possibilité de commenter un meeting
- constituer un panier
- choisir ses réservations en fonction des places disponibles
- la possibilité de payer en ligne via paypal
- la possibilité de savoir si un administrateur est connecté (websocket)
- la possibilité de communiquer en direct avec un administrateur grâce au websocket avec un système de notification des messages
- Si on est administrateur, la possibilité de voir les revenues générés sur le site

Sur ce site, côté utilisateur vous avez la possibilité de retrouver l'historique de toute vos commandes et d'imprimer vos billets.
Côté administrateur, vous avez la possibilité de retrouver la liste des revenues générés par les ventes (classé selon les mois) et d'accéder à l'interface d'administration django.
Au niveau du serveur, j'utilise un cron qui désactive, de façon intelligentes, les commandes impayé ayant une duré vie supérieur à 15 minutes. Par exemple si un utilisateur payes une commande après 20 minutes le code va devoir recalculer les places disponibles et générer ou non le billet.

Sur les technologies utilisés on a le framework django pour la logique principale du site, django channels et javascript pour le websocket et bootstrap pour le css.

Bilan de mon expérience sur ce projet

Sur ce projet j'ai surtout appris à utiliser le websockets côté frontend (javascript) et coté backend (python). La difficulté étant de bien agencer les requêtes websocket qui peuvent s'exécuter parallèlement.

Les 12 Bonnes Pratiques De l'Xtreme Programming

Client sur site:

Le client, un ami, m'a régulièrement envoyé ses attentes via les réseaux sociaux.

Jeu du planning ou planning poker

Au préalable le travail à été découpé en *fonctionnalités* dans un tableau trello. Si on ajoute le fait de développer seul, il est aisé de faire un point quotidien (à la livraison d'une fonctionnalité).

Intégration continue

Utilisation de travis-ci pour l'intégration continue. Quand je push un commit sur github, les tests sont effectué sur travis-ci puis le serveur est mis à jour.

Petites livraisons

L'intégration continue et les tests réduisent considérablement le coût de livraison.

Rythme soutenable

L'équipe ne fait pas d'heures supplémentaires.

Tests unitaires/fonctionnels

Avant de mettre en œuvre une fonctionnalité, j'écris un test qui vérifiera que mon programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, je relance tous les tests que j'ai écris.

Conception simple

Conception simple et intuitive du projet.

Refactoring (ou remaniement du code)

Une fois le prototype du code construit et fonctionnelle, j’ai pu commencer le refactoring.

Appropriation collective du code

Développement effectué seul.

Convention de nommage

Python d'une part et *Django* ensuite sont deux cadre de travails qui prône les conventions de nommage avant le codage.

Pair-programming

Développement effectué seul.