Compte rendu final

Projet Vue.js / Symfony

1. Introduction

Le projet est de mettre en place, dans un premier temps, un site conçu en PHP grâce au Framework PHP Symfony, puis de créer un projet Vue.js, pour que dans un troisième temps nous mettions en place un système permettant d'utiliser le projet Symfony comme API afin de récupérer les données sur le projet Vue.js.

Mon projet est de créer un site permettant de consulter des événements en France pour y réserver des places. Il est aussi possible de gérer le site grâce à un compte administrateur où celui-ci peut créer, modifier et supprimer des événements.

Le template utilisé a été créé par HTML5 UP et s'appelle Phantom. Celui-ci m'a permis de ne pas vraiment m'occuper du design de mon site en plus de le rendre responsive.

Toutes les figures illustrant le travail effectué sont présents dans les annexes pour une meilleure visibilité.

2. Mise en place des projets

Projet Symfony :

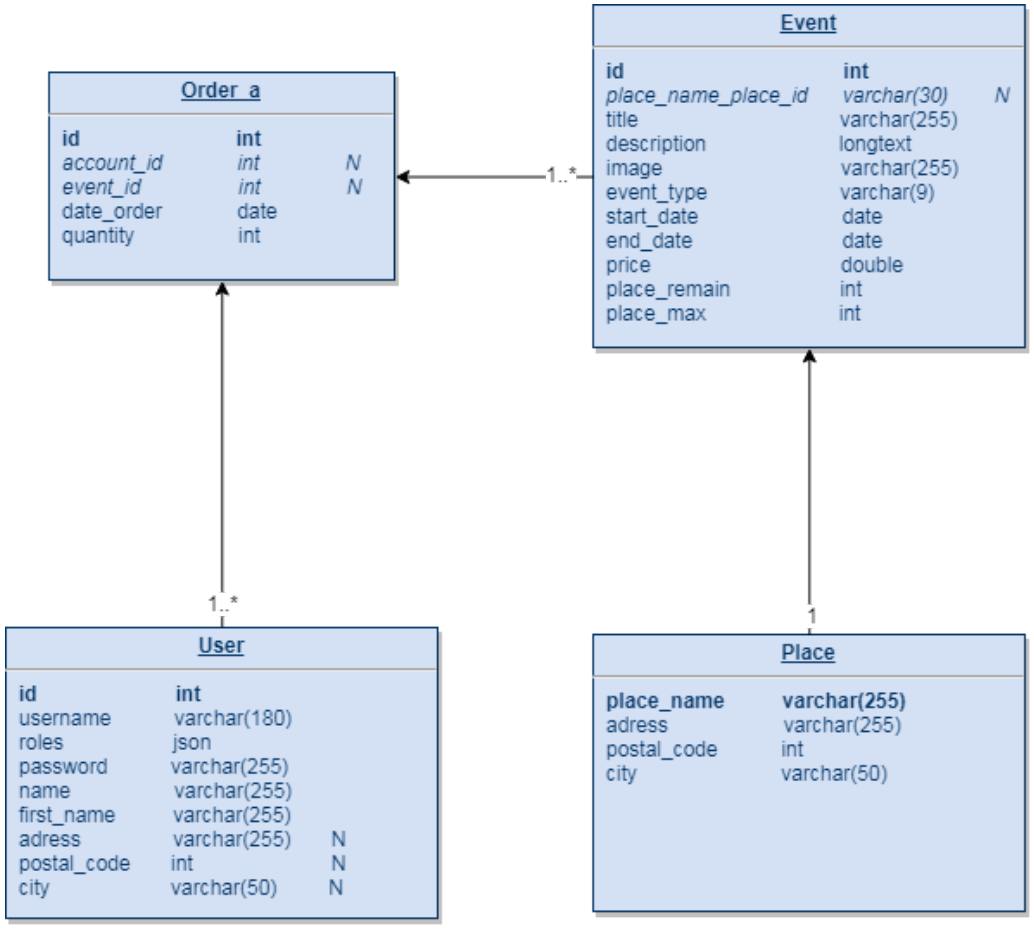
```
# Installer les dépendances du projet
composer install

# Lancer le serveur Symfony (sur le port 8000)
bin/console server:start

# Lancer le serveur de base de données MySQL
sudo service mysql start

# Créer la base de données sur le serveur
bin/console doctrine:database:create

# Créer les entités de base de données
bin/console doctrine:migrations:migrate

# Mettre en place des données dans la base de données
bin/console doctrine:fixtures load
```

Projet Vue.js :

```
# Installer les dépendances du projet
npm install
```

```
# Lancer le serveur Node.js (sur le port 8080)
npm run serve
```

3. Base de données



Base de données du projet Symfony

4. Partie Symfony

a. Outils

Voici la liste des bundles Symfony utilisées lors de la création du site en PHP (Seulement les bundles ajoutés et important)

Bundles	Descriptions
ApiPlatformBundle	Bundle permettant de créer une api en fonction des entités voulu
DoctrineBundle	Bundle permettant de mettre en place une base de données
DoctrineFixturesBundle	Bundle permettant de créer des données en bases de données
DoctrineMigrationsBundle	Bundle permettant de créer les tables de bases de données en fonctions des entités voulu
NelmioCorsBundle	Bundle permettant de filtrer les requêtes vers le serveur
SecurityBundle	Bundle permettant de sécuriser les connexions sur le site

b. Architecture des controllers

Voici la liste des controllers avec les méthodes correspondant aux URLs utilisées pour mon site

Nom des méthodes	URLs	Descriptions
AdminController.php		
addEvent	/admin/addEvent	Permet de créer un nouveau événement
addPlace	/admin/addPlace	Permet de créer un nouveau lieu
deleteEvent	/admin/deleteEvent/{idEvent}	Permet de supprimer un événement existant
updateEvent	/admin/updateEvent/{idEvent}	Permet de modifier un événement existant
DefaultController.php		
allEvent	/	Permet d'afficher tous les événements
detailEvent	/allEvent/{idEvent}	Permet d'afficher un événements précis
RegistrationController.php		

register	/register	Permet d'enregistrer un nouvel utilisateur
SecurityController.php		
login	/login	Permet de se connecter
UserController.php		
myOrders	/user/orders	Permet de voir la liste des réservations faites pour un utilisateur
order	/user/order/{idEvent}	Permet de réserver des places d'un événement
validOrder	/user/validOrder	Permet de valider sa réservation

c. Architecture des fichiers backend

Voici la liste des fichiers utilisées pour le backend de mon site

Nom des fichiers	Descriptions
DataFixtures	
AppFixtures.php	Permet de créer des données non-utilisateur en bases de données
UserFixtures.php	Permet de créer des données utilisateur en bases de données
Entity	
Event.php	Entité Événement
OrderA.php	Entité Commande
Place.php	Entité Lieu
User.php	Entité Utilisateur
Form	
AddEventType.php	Formulaire d'ajout d'événement
AddPlaceType.php	Formulaire d'ajout de lieu
EmptyType.php	Formulaire vide
EventQuantityType.php	Formulaire du nombre de place d'événement à réserver
RegistrationFormType.php	Formulaire d'enregistrement d'utilisateur

UpdateEventType.php	Formulaire de modification d'événement
Migrations	
Version20190502110435.php	Fichier permettant de créer les tables de bases de données en plus de leurs relations (généré automatiquement grâce à DoctrineMigrationsBundle
Repository	
EventRepository.php	Fichier permettant de créer une "relation" entre l'entité Événement et la base de données
OrderARepository.php	Fichier permettant de créer une "relation" entre l'entité Commande et la base de données
PlaceRepository.php	Fichier permettant de créer une "relation" entre l'entité Lieu et la base de données
UserRepository.php	Fichier permettant de créer une "relation" entre l'entité Utilisateur et la base de données
Security	
LoginFormAuthentificator.php	Formulaire de connexion, sécurisant les connexions au site

d. Architecture des fichiers frontend

Voici la liste des fichiers avec les URLs correspondant utilisées pour le frontend de mon site

Nom des fichiers	URLs	Descriptions
base.html.twig	*	Header, Menu et Footer du site (présent sur toutes les pages)
admin		
addEvent.html.twig	/admin/addEvent	Page d'ajout d'événement
addPlace.html.twig	/admin/addPlace	Page d'ajout de lieu
deleteEvent.html.twig	/admin/deleteEvent/{idEvent}	Page de suppression d'un événement
updateEvent.html.twig	/admin/updateEvent/{idEvent}	Page de modification d'un événement
default		
detailEvent.html.twig	/allEvent/{idEvent}	Page d'affichage d'un événement

index.html.twig	/	Page d'affichage de tous les événements
registration		
register.html.twig	/register	Page d'enregistrement utilisateur
security		
login.html.twig	/login	Page de connexion
user		
myOrders.html.twig	/user/orders	Page d'affichage des réservations faites
order.html.twig	/user/order/{idEvent}	Page de réservation

e. Le site

i. Affichage des évènements

L'affichage des événements (qui est aussi la page d'accueil) s'effectue grâce au controller **DefaultController**, celui-ci permet d'afficher tous les événements***(fig1)*** sur la route **"/"** on y retrouve alors, le titre, le type et la description des événements disposés en bloc.

Lors du clique sur l'un des blocs, l'utilisateur est redirigé sur la route **"/allEvent/{idEvent}"** on y retrouve la description complète d'un événement et propose à l'utilisateur de réserver des places pour l'événement***(fig2)***. La totalité des informations sont récupérées de la base de données

fig1. Listing des événements

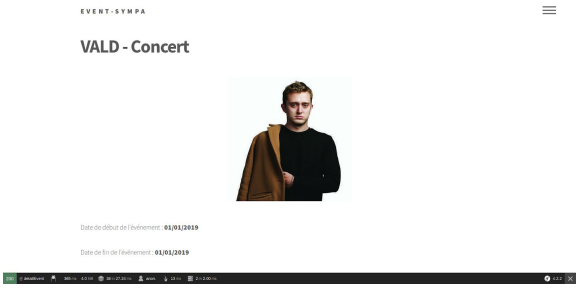


fig2. Description d'un événement

ii. Administrations

La partie administration est gérée avec le controller **AdminController**, seul un utilisateur identifié comme administrateur peut accéder aux boutons et aux URLs

d'administration. On reconnaît ces pages grâce à l'URL, si celui-ci commence avec **"/admin"** alors il faut être connecté comme administrateur si ce n'est pas le cas, l'utilisateur est redirigé sur la page de connexion.

Il est possible de modifier un événements grâce au "crayon" placé à côté des blocs, le bouton redirige l'administrateur sur la route **"/admin/updateEvent/{idEvent}"** vers un formulaire (**UpdateEventType**) pré-remplie avec les informations de l'événement voulant être modifié***(fig3)***.

La suppression d'un événement se fait de la même manière que la modification, un bouton "moins" est présent sur le côté des blocs, celui-ci ouvre une page **"/admin/deleteEvent/{idEvent}"** demandant seulement la confirmation pour la suppression de l'événement sélectionné***(fig4)***.

L'ajout d'événement est possible en accédant au menu et en cliquant sur "Ajouter un événement", l'utilisateur accède alors à l'URL **"/admin/addEvent"** avec un formulaire vide (**AddEventType**) demandant les informations pour créer un événement***(fig5)***. Les nom des images téléchargées servant de vignette sont redéfini pour éviter les problème de nom d'image.

Pour finir, on peut ajouter un nouveau lieu d'événement sur la page **"/admin/addPlace"**. On y accède grâce au bouton dans le menu "Ajoute un lieu", alors la page affiche un formulaire vide (**AddPlaceType**) avec les champs pour créer un lieu***(fig6)***.

Lors de la validation de chacun des éléments d'administration, l'administrateur est redirigé sur la page d'accueil

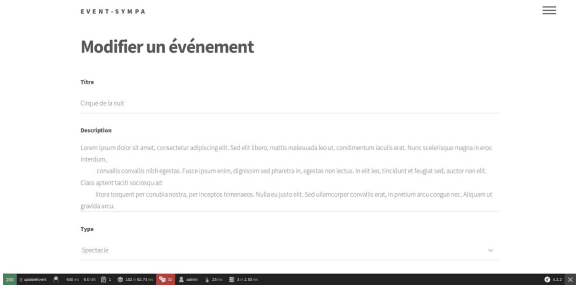


fig3. Modification d'un événement

fig4. Suppression d'un événement

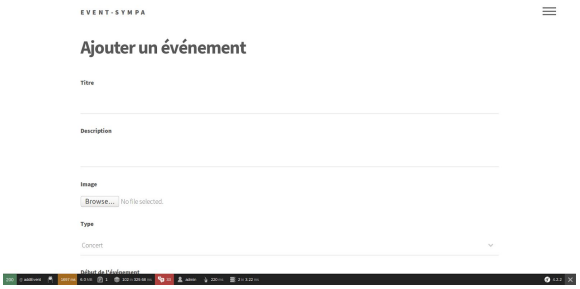


fig5. Ajout d'un événement

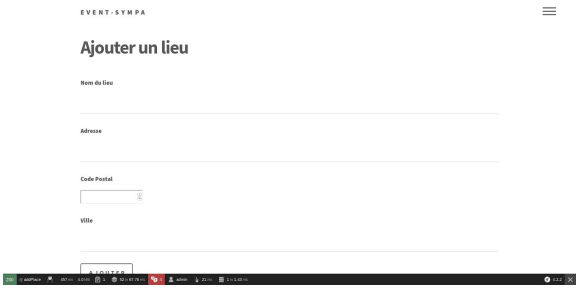


fig6. Ajout d'un lieu

iii. Connexion, déconnexion, gestion des comptes et menu

Le système de gestion des comptes à été créer grâce au bundle SecurityBundle qui gère presque la totalité de cette gestion.

De base 2 comptes existe, un compte utilisateur classique (nom de compte : **user**, mot de passe : **user**) et un compte administrateur (nom de compte : **admin**, mot de passe : **admin**), ces comptes sont créés et enregistrées en base de données grâce au fichier **UserFixtures** où les mots de passes sont cryptés.

L'inscription se fait via le menu où le bouton "S'inscrire" est disponible si personne n'est connecté, ou sur la page de connexion, alors l'utilisateur accède à l'URL **"/register"** où le formulaire d'inscription (**RegistrationFormType**) est vide***(fig7)***, après l'inscription, l'utilisateur est directement connecté.

Si l'utilisateur a déjà un compte, il peut accéder au formulaire de connexion (**LoginFormAuthentificator**) à l'URL "/login" pour se connecter***(fig8)*** en cliquant sur le bouton "Se connecter" qui est disponible de la même façon que le bouton "S'inscrire"

L'utilisateur peut se déconnecter avec le bouton Déconnexion disponible dans le menu.
Comme dit implicitement ci dessus, le menu s'adapte en fonction de l'utilisateur, le menu n'est pas le même si l'utilisateur est connecté ou non***(fig9 et fig10).***

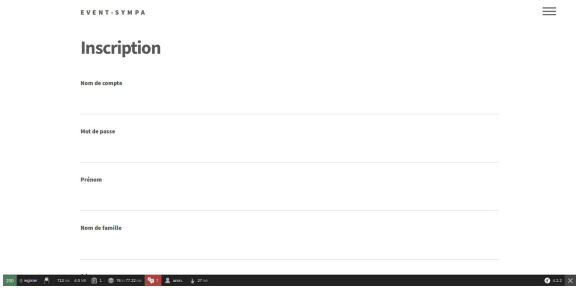


fig7. Inscription utilisateur

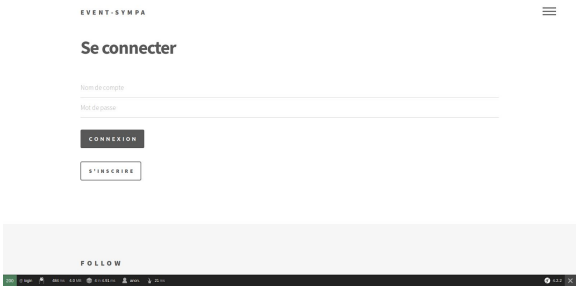


fig8. Connexion utilisateur

fig9. Menu non connecté

fig10. Menu administrateur connecté

iv. Réservation

Une réservation peut s'effectuer seulement si un utilisateur est connecté. S'il n'est pas connecté alors il est redirigé vers la page de connexion.

L'utilisateur peut réserver après clique sur le bouton "Réserver" sur la page de description d'événement. Il accède à l'URL **"/user/order/{idEvent}"** alors à la page demandant le nombre de places voulues (**EventQuantityType**)***(fig11)***.

S'il valide la quantité, il peut voir les différentes réservations faites sur l'URL **"/user/orders"** disponible dans le menu grâce à "Mes réservations", un tableau récapitulatif apparaît***(fig12)***.

fig11. Réservation

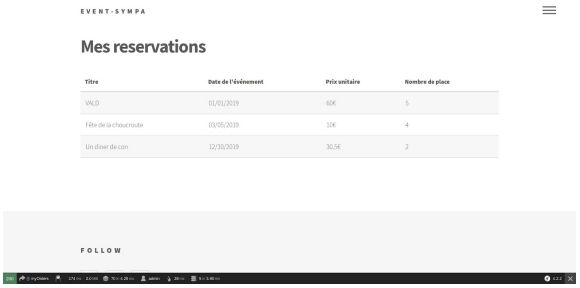


fig12. Listing des réservation effectuées

5. Partie Vue.js

a. Outils

Voici la liste des dépendances Vue.js utilisées lors de la création du site

Dépendances	Descriptions
Axios	Permet de faire des requêtes vers une API et de traiter des données reçu
Moment	Permet le formatage des dates
Router	Permet de créer des URLs correspondant à des vues
Vuex	Permet de stocker des données

b. Architecture des fichiers

Voici la liste des composants utilisées pour mon site

Nom des fichiers	Descriptions
components	
EventDescription.vue	Composant description d'un événement
EventImage.vue	Composant correspondant aux "carrés" visant les événements
Events.vue	Composant titre et mise en place des "carrés"
views	
Event.vue	Vue de description d'un objet
Home.vue	Vue de listing des événements

c. Affichage des données

Pour la partie Vue.js, j'ai décidés de reprendre le design du site en Symfony. J'ai alors créer une vue pour le listing des événements, ainsi qu'une vue pour la description d'un événement.

Le listing des événement se fait avec la vue "**Home**" de l'URL **"/"** qui appelle le composant "**Events**" qui lui même créer le nombre d'"**EventImage**" équivalent au nombre d'événement existant***(fig13)***. Lors du clique sur l'une des images alors l'utilisateur est redirigé vers l'URL "**/Event/:id**" et la vue "**Event**" apparaît qui fait appelle au composant "**EventDescription**" qui rédige la totalité des informations d'un événement***(fig14)***.

fig13. Listing des événements

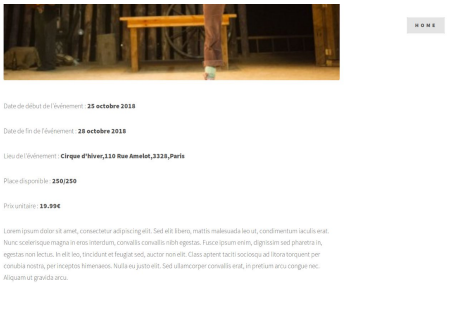


fig14. Description d'un événement

6. Partie API

a. URLs

Voici la liste des URLs et méthodes créer par API Platform du projet Symfony et utilisées par le projet Vue.js

URLs	Méthodes	Descriptions
/api/events	GET	Récupère les événement
/api/events	POST	Ajoute un nouveau événement
/api/events/{idEvent}	GET	Récupère l'événement à l'identifiant idEvent
/api/events/{idEvent}	DELETE	Supprime l'événement à l'identifiant idEvent
/api/events/{idEvent}	PUT	Modifie l'événement à l'identifiant idEvent
/api/places	GET	Récupère les lieux
/api/places	POST	Ajoute un nouveau lieu
/api/places/{idPlace}	GET	Récupère le lieu à l'identifiant idPlace
/api/places/{idPlace}	DELETE	Supprime le lieu à l'identifiant idPlace
/api/places/{idPlace}	PUT	Modifie le lieu à l'identifiant idPlace

b. Mise en place de l'API

Pour l'API il a fallu mettre en place le bundle API Platform qui permet de créer automatique l'API en fonction des entités voulues, j'ai choisi les entités Event et Place qui sont les deux seuls entités importante dans mon cas et qui vont être utilisé par la suite dans mon projet Vue.js.

c. Récupération des données

La récupération des données se fait simplement grâce à Axios. Pour ce faire, il a seulement été enregistré dans le store de Vuex les URLs de l'API. **"localhost:8000/api/events/"** pour les événements et **"localhost:8000/api/places/"** pour les lieux. Suite à cela, j'ai tout simplement demandé à Axios de faire des requêtes sur ces URL quand j'en avais besoin, bien sûr, les URLs ont possiblement été modifié afin qu'ils soient utilisés dans certains cas.

7. Conclusion

Pour conclure, le projet m'a apporté énormément de compétences en développement Web, que ça soit dans la façon de réfléchir ou même dans mes compétences de programmation.

Ce n'est pas la première fois que j'élabore un projet sous Symfony, mais cela m'a néanmoins permis de me remémorer la façon de comment fonctionne le Framework en plus de me montrer les différences entre Symfony 3 et 4.

Le projet Vue.js quant à lui m'a réellement appris beaucoup de chose sur le fonctionnement de Javascript. J'ai découvert une façon de programmer très intéressante, utiliser un système de composants réutilisable est très intelligent.

J'ai réellement apprécié mettre en place les projets ainsi que de les développer.

8. Annexes

fig1. Listing des événements

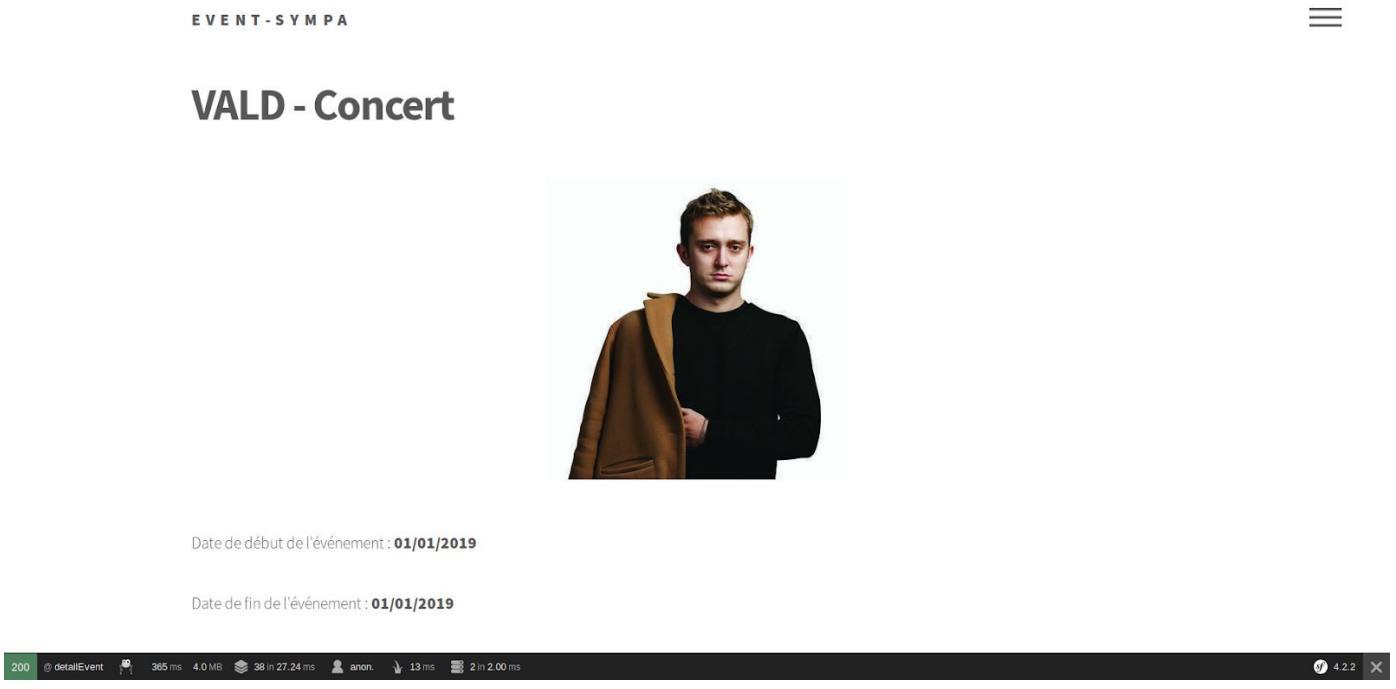


fig2. Description d'un événement

fig3. Modification d'un événement

fig4. Suppression d'un événement

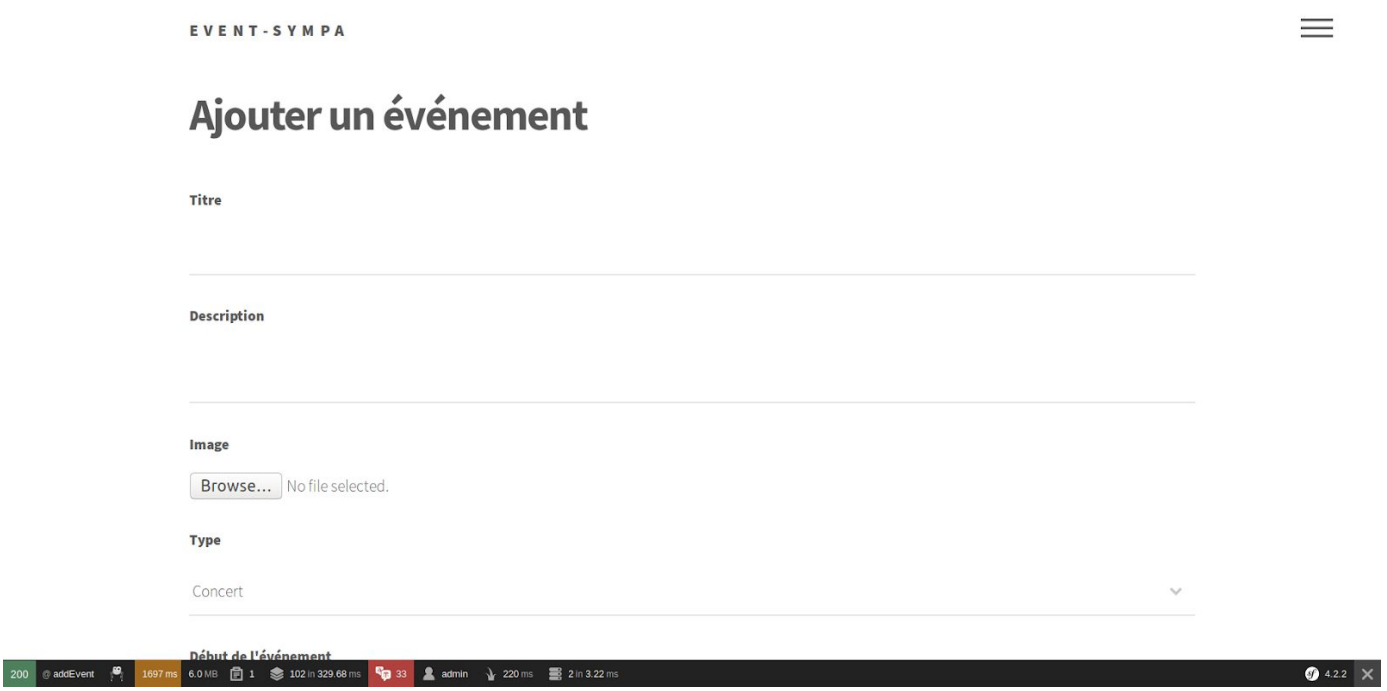


fig5. Ajout d'un événement

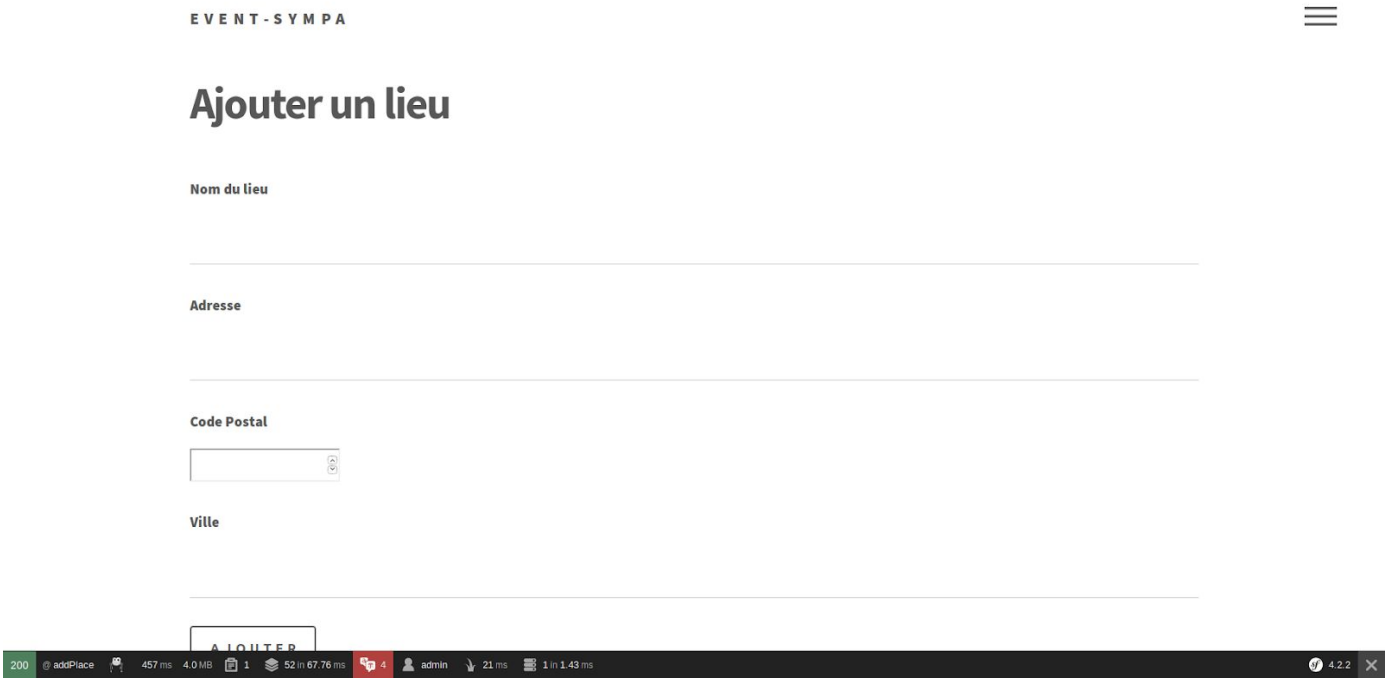


fig6. Ajout d'un lieu

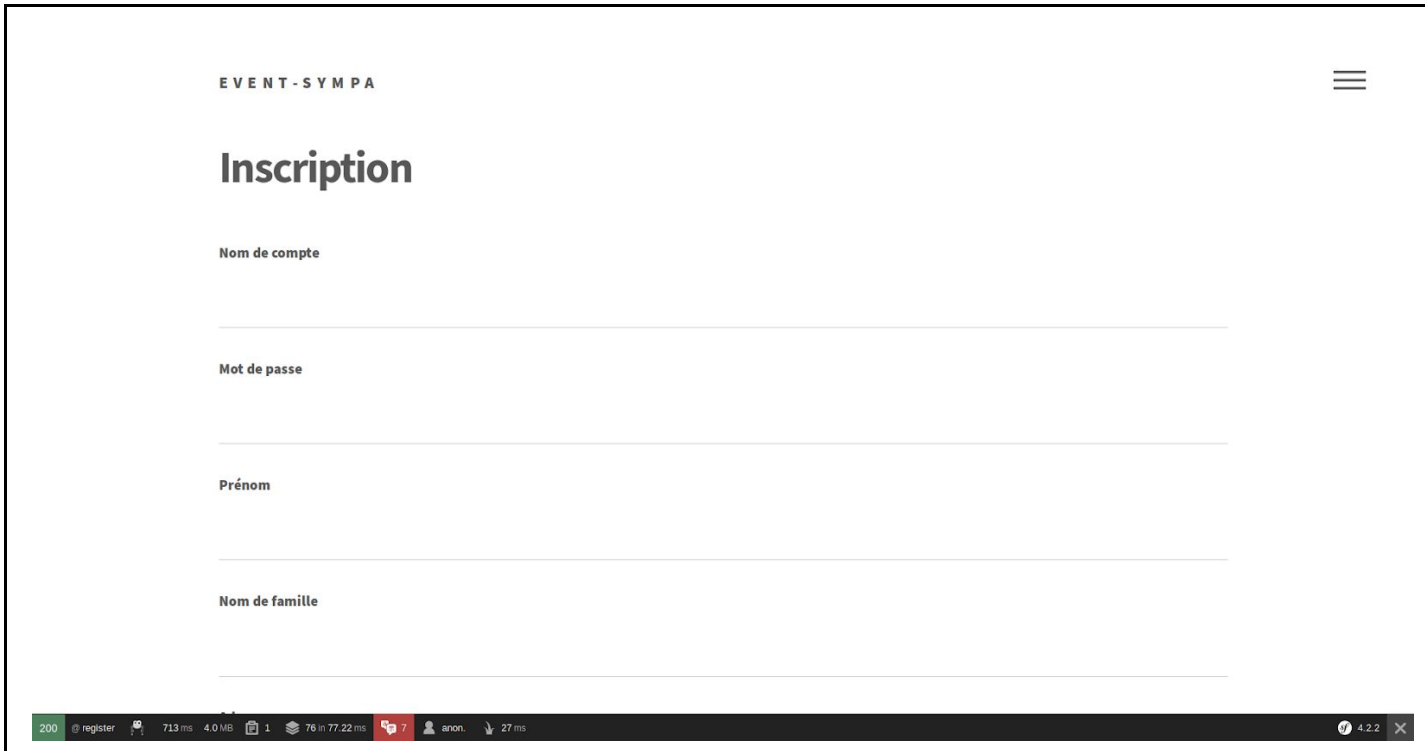


fig7. Inscription utilisateur

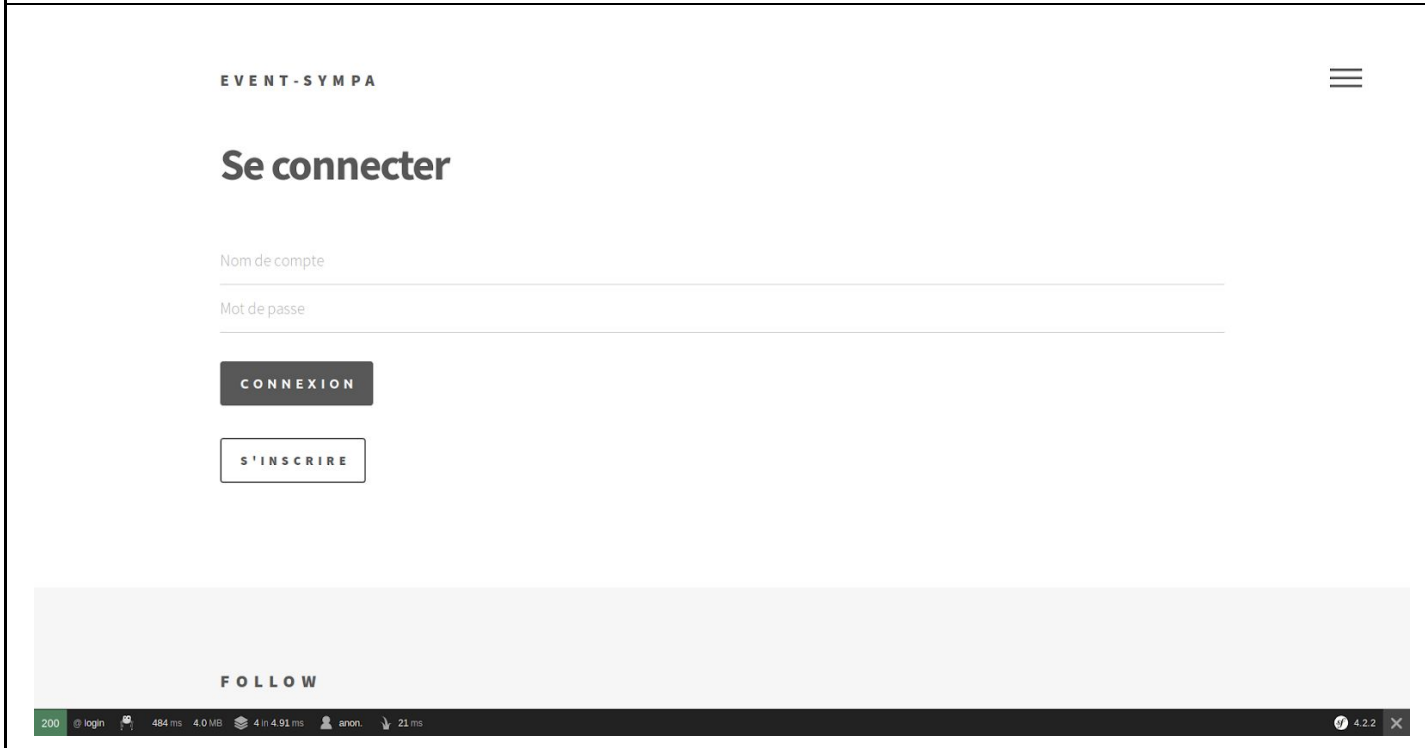


fig8. Connexion utilisateur

fig9. Menu non connecté

fig10. Menu administrateur connecté

fig11. Réservation

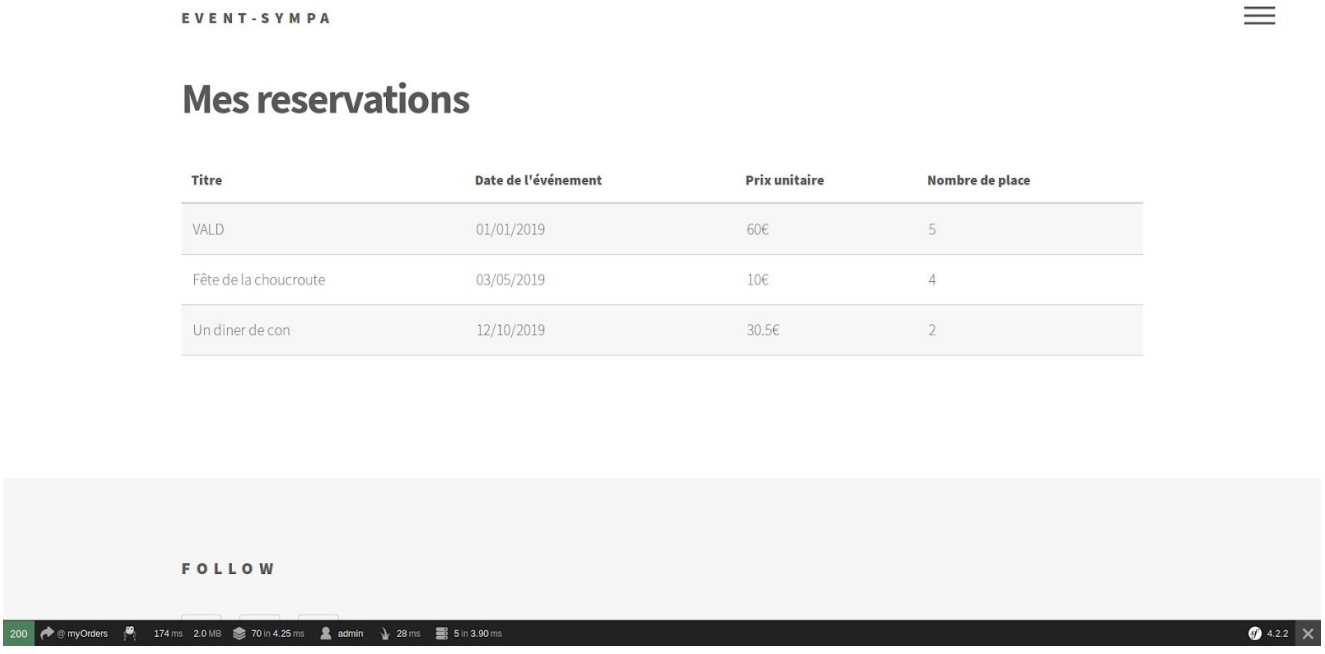


fig12. Listing des réservation effectuées

fig13. Listing des événements

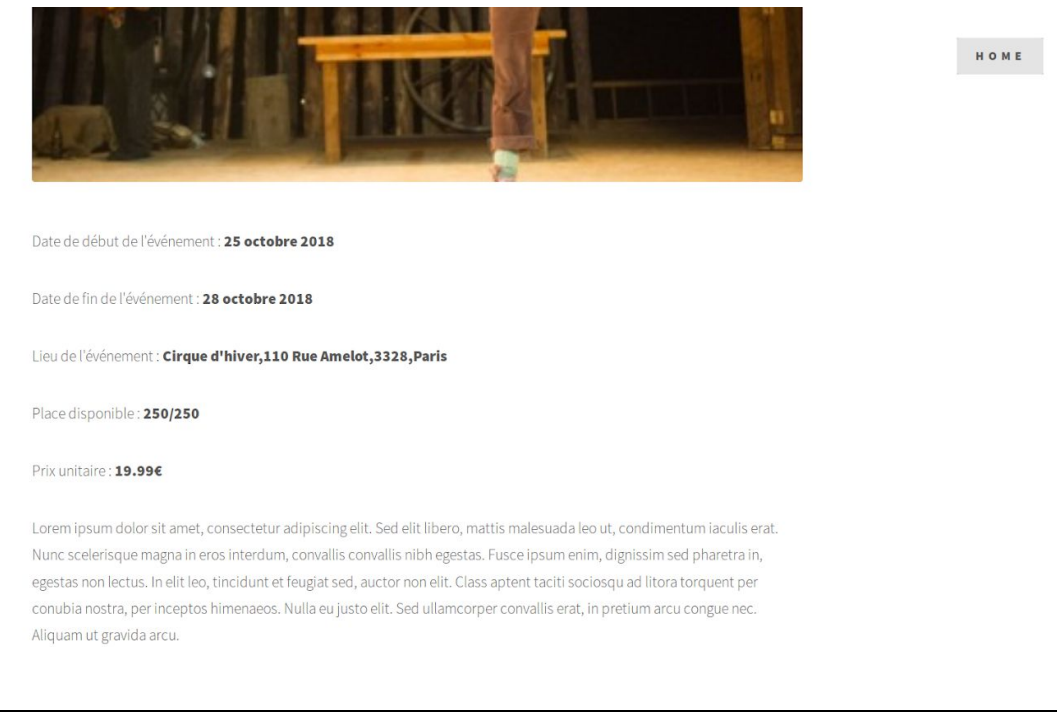


fig14. Description d'un événement