Interface SAV

L’équipe de goto4ever.com a travaillé pour vous et vous présente ici leurs solution

Table of Contents

[Contexte 1](#_Toc481449886)

[Définition du besoin 1](#_Toc481449887)

[Solution professionnelle 1](#_Toc481449888)

[Solution maintenable 2](#_Toc481449889)

[Expérience utilisateur 2](#_Toc481449890)

[Gestion de projet 3](#_Toc481449891)

[Documentation technique 4](#_Toc481449892)

[Base de donnée 5](#_Toc481449893)

[Fin 6](#_Toc481449894)

# Contexte

<https://symfony.com>

<https://symfony.com/doc/current/doctrine.html>

<http://knpbundles.com/>

## Définition du besoin

Il y a un besoin pour une interface utilisée par les clients de votre entreprise qui leurs permet d’obtenir des information (FAQ, documentation), ou d’en demander (SAV).  
L’interface doit aussi pouvoir être utilisé par l’administrateur afin d’ajouter, modifier les comptes clients. Et d’ajouter les ressources et informations à disposition du client.

Avec l’aide de votre pdf et des réponses données à nos question nous avons extraits quelques points qui nous semblent essentiel au projet :

1\ La solution doit être une solution professionnelle, sécuritaire et robuste.  
 2\ La solution doit être maintenable et extensible le plus facilement possible  
 3\ L’expérience utilisateur de l’interface client est un point indispensable.

Nous allons développer les points ci-dessus dans les quelques pages suivantes.

## Solution professionnelle

Nous avons choisi d’utiliser un framework ayant fait ses preuves depuis de nombreuses années. Symfony est au cœur de notre projet il possède d’énorme avantage par rapport à un site développé sans framework :

* La structure du site est standard au framework ce qui le rend d’autant plus facilement maintenable.
* Le framework pousse à utiliser des design pattern rendant le code d’autant plus lisible et navigable. La logique des données est séparée de la logique d’affichage et de la logique de manipulation de ces premières.
* La manipulation des données et la maintenance de la base de données est grandement simplifié grâce à Doctrine qui permet de manipuler des objets, de les stocker tel quel avec toutes leurs relations dans la base de données relationnelle de votre choix.
* L’ajout de fonctionnalité sensible peut être fait par l’installation de modules crée et validée par la communauté. L’authentification du projet pourra par exemple à terme être assurée par le bundle Oauth qui nous permet d’ajouter une fonctionnalité éprouvé et sécuritaire facilement à notre projet.

## Solution maintenable

La séparation des différentes logiques de notre application permet de définir un rôle défini et explicite à chaques fichiers. Et encore plus précisément l’utilisation de Symfony force le développeur à utiliser des noms de méthodes standardisées une méthode finissant par Action dans un Contrôleur aura toujours le même objectif d’envoyer à l’utilisateur une page ou des données selon la requête obtenue.

La solution que nous vous proposons est disposé en packages, standardisé par le framework Symfony. Ce qui permet à n’importe quel développeur Symfony de trouver tout de suite ses marques et de pouvoir se mettre à travailler sur le code sans temps d’apprentissage / d’adaptation.

Symfony en tant que framework vous assure aussi que dans 10 ans votre solution sera toujours aussi maintenable et supportée par Symfony qui se veut un framework avec un support à très long terme.

## Expérience utilisateur

Nous avons fait un usage intensif d’Ajax et des avantages considérable qu’il nous apporte par rapport à un site ‘Une page un chargement’ classique.  
A la connexion, le client charge le squelette de toute les pages qui lui sont disponible, les pages en elle-même sont très légère, elles n’ont pas d’images seulement une structure html, et des fichiers CSS ne pesant pas lourd sur la balance.  
Le premier chargement est donc instantané, et la navigation l’est aussi, à chaque changement de pages, une requête ajax est envoyé pour récupérer les données à afficher à l’utilisateur. Les données étant récupérée en JSON cette requête est donc ultra light, la structure html/CSS dans laquelle les données doivent être incorporé étant déjà été chargé au login, le changement de pages est donc virtuellement instantané et surtout ne de demande pas le rechargement de la page.

Le prototype que nous vous proposons à une interface identique à la maquette fournie.

# Gestion de projet

Le projet a été lancé le jeudi 26 avril. Les premières étapes ont été comme sur tout nos projets la mise en place d’un dépôt github privé qui nous permet une synchronisation du code, un suivi et une vision globale de l’avancement du projet, une gestion des erreurs et problèmes rencontrés.

La seconde étape a été l’analyse du besoin et la mise en place d’une solution répondant le plus pertinemment à ce dernier. Ce qui nous a conduit sur une solution Symfony. Les tâches ont été séparée comme suit :

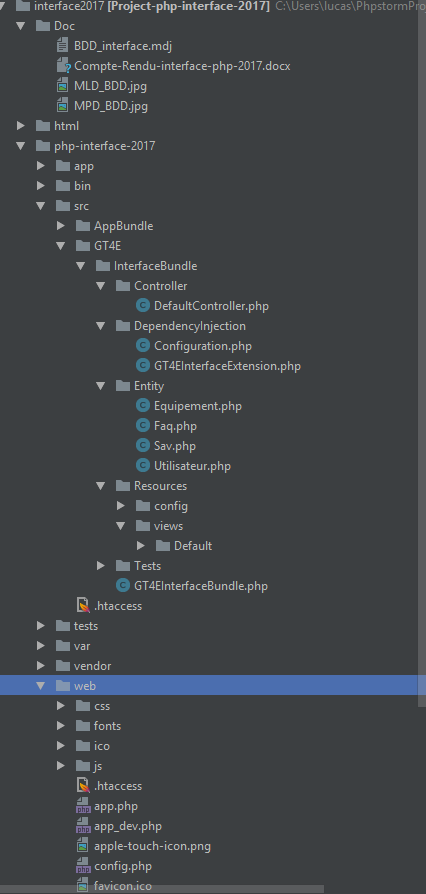
Quentin à crée l’interface utilisateur grâces aux templates Twig de Symfony, il a aussi mis en place les routes et la logique de ces dernières.

Lucas s’est occupé de mettre en place les modèles doctrines permettant l’interface avec la base de données.

Dorian s’est occupé de la recherche sur toute les fonctionnalités demandé, leurs faisabilités ainsi que la vérification de la bonne réponse du cahier des charges tout au long du projet.

La communication s’est fait par voie électronique essentiellement par messagerie instantanée. L’utilisation extensive de design pattern a permis de réduire la communication purement technique, tout étant déjà standardisé par ces design pattern nous n’avons pas eu de discussion enflammée sur le design d’application à adopter.

# Documentation technique



Voici la structure de notre solution :

Le projet se trouve essentiellement dans php-interface-2017

App est un dossier comportant les configurations par défaut de Symfony que nous ne modifions pas ici.

Bin contient la console Symfony qui nous permet de créer la base de données, de démarrer le serveur.

Dans src\GT4E\InterfaceBundle  
se trouve la structure de notre projet à proprement parlé.  
(GT4E pour goto4ever)

Dans Controller se trouve le contrôleur par défaut c’est là ou toute les routes en directions de l’url se retrouvent c’est le C du modèle MVC.

Dans Entity se trouvent toutes les classes décrivant et permettant la manipulation des données c’est le M du modèle MVC.

Dans Ressources views se trouvent toutes les vues pouvant être servie, c’est le V du modèle MVC.

Web contient les ressources devant être accessible publiquement, par exemple le CSS et le JS utilisé coté client.

## C:\Users\lucas\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BDD.JPGBase de donnée

Voici la base de données de l’application.  
Grâce à Doctrine les liaisons que vous pouvez voir ci-dessus sont complètement transparente dans le code, il n’y a aucun besoin de requetes SELECT ou JOIN longues et obscures.  
Si vous désirez un Utilisateur Doctrine vous servira un objet Utilisateur avec un getter et un setter pour chaque propriété ci-dessus. Avec les relations de façon transparente aussi, un Utilisateur aura par exemple un tableau d’équipement ce qui traduit la relation oneToMany ci-dessus.

Pour le reste concernant le fonctionnement des Classes ou du flow de l’information de notre application tout est standardisé grâce à Symfony, nous vous invitons donc à lire cette page :

<https://symfony.com/doc/current/quick_tour/the_architecture.html> qui expliquera de manière beaucoup plus claire et précise le fonctionnement des différentes parties de notre application.

# Fin

Voici la fin de cette documentation,

Le projet a été très agréable pour tous les développeurs impliqués, nous sommes fiers de pouvoir vous présenter un prototype fonctionnel qui tiens la route et est robuste sur toute les fonctionnalité développé et qui peut être terminé en l’espace de quelques soirées.

Nous restons à votre disposition pour toute question ou pour toutes envie de continuer l’aventure sur ce projet où sur un autre.

-L’équipe goto4ever

[contact@goto4ever.com](mailto:contact@goto4ever.com)