Informe trabajo final - Grupo 13

Integrantes: Pasquale Julian, Soloaga Ignacio, Tettamanti Santiago

Framework utilizado: Symfony
Version: 3.4
Lenguaje: PHP
Version: 7.0
Motor de templates: Twig

Fundamentación de nuestra elección

Elegimos Symfony principalmente porque ninguno de los 3 lo habia usado y queríamos aprender. Buscamos frameworks php y los dos más conocidos para desarrollo web son Laravel y Symfony. Nos inclinamos por el último porque utiliza como gestor de plantillas Twig, y pensamos en reutilizar los templates del trabajo previo.

Referencias

Utilizamos como fuente principal de apoyo la documentación oficial de Symfony 3.4
https://symfony.com/doc/3.4/setup.html

Consultas específicas sobre manejo del framework
https://stackoverflow.com

Reutilización de módulos

Como dijimos en la fundamentación Symfony usa como gestor de plantillas Twig, que fue el usado por nosotros en el trabajo anterior. Debido a esto, pudimos reutilizar todas las views, teniendo que modificar muy pocas cosas en los html.
Por otro lado, si bien programamos los controladores de 0 nuevamente, mantuvimos la misma lógica que llevaban los controladores en la entrega anterior.

Mecanismo provisto para Seguridad y Routing

Routing

Symfony provee un ruteador que permite:

-Definir urls mapeadas a distintas áreas de la aplicación.
-Crear rutas complejas que mapean a controladores.
-Generar URLs dentro de templates y controladores.
-Cargar recursos de ruteo de paquetes (o cualquier otro lado).
-Hacer debugging de las rutas.

Puede definirse utilizando anotaciones, xml, yaml o php. Nosotros elegimos volcar todo en un archivo .yaml para modularización y legibilidad.

Seguridad

Symfony provee múltiples mecanismos de seguridad según lo que se quiera restringir o proteger. Nosotros utilizamos el módulo de Autorización, Roles y Acceso denegado para poder manejar de manera prolija y eficiente los roles de usuarios.

El proceso de autorización se basa en dos aspectos diferentes:

1. El usuario recibe un set específico de roles al loguearse (ej. ROLE_Admin).
2. Añadir código para que un recurso (ej. URL, Controlador) requiere un atributo específico (generalmente un rol como ROLE_Admin) para que puede ser accedido.

Mecanismo provisto para operaciones CRUD

El ORM utilizado por Symfony es Doctrine, y provee un conjunto de comandos para generar Entidades y los Repositorios de las mismas.

doctrine:generate:entity - Genera una entidad con los atributos deseados y sus respectivos getters y setters.

doctrine:schema:validate - Valida los esquemas de entidades contra la base de datos y refleja los cambios en la misma.

Estos fueron los utilizados por nosotros.

Además, provee un comando generate:doctrine:crud que genera un controlador con las operaciones CRUD en base a una entidad. Nosotros no hicimos uso del mismo porque quisimos mantener la lógica de los controladores que definimos en el trabajo previo, aunque más tarde en el desarrollo del trabajo con el framework nos dimos cuenta que hubiese sido más fácil y rápido hacer uso del generador que provee Doctrine.

Mecanismo provisto para manejar el modelo MVC

Symfony estructura una aplicación con una rama de directorios generada automáticamente. Las carpetas a destacar son:

app/config - Todos los archivos de configuración (ruteo, seguridad, base de datos, etc.)

app/Resources/views - Todas las vistas

src/AppBundle/Controller - Todos los controladores

src/AppBundle/Entity - Todas las entidades

src/AppBundle/Repository - Todos los repositorios basados en las entidades

vendor - Todos los paquetes externos a Symfony utilizados por el mismo.

web/css - Todos los estilos definidos por nosotros.