Seminario 2: TFG - De la Idea al Prototipo Arquitectónico

Asignatura: Trabajo de Fin de Grado (TFG) **Autor:** Manuel Prieto Macias

---

1. Objetivos y Entregables

Este seminario es el **"Paso Cero"** de la codificación de tu TFG. El objetivo no es escribir la aplicación completa, sino **construir un esqueleto funcional y robusto** sobre el cual puedas desarrollar tu proyecto de forma ordenada y profesional. Conecta directamente con el trabajo de análisis que ya has realizado.

Entregables Obligatorios

1. **Presentación del Diagrama E/R (5-10 min)**: Deberás presentar tu diagrama Entidad-Relación final a la clase, explicando las entidades principales, sus atributos y, lo más importante, justificando las relaciones (1-N, N-M) que has elegido.
2. **Presentación de la Especificación Técnica (5-10 min)**: Deberás exponer un resumen de tu especificación técnica, centrándote en la arquitectura de alto nivel, las tecnologías elegidas (y por qué) y el alcance funcional del proyecto (MVP - Mínimo Producto Viable).
3. **Prototipo Arquitectónico en Symfony**: El código resultante de seguir los pasos de este seminario.

2. El Enfoque: Arquitectura Primero

Antes de escribir una sola línea de lógica de negocio, un buen arquitecto de software define la estructura. Nuestro proceso será:

1. **Configurar el Entorno**: Adaptar Docker a las necesidades de *tu* proyecto.

2. **Modelar los Datos**: Traducir tu diagrama E/R a entidades de Doctrine.
3. **Definir las Rutas**: Crear un "mapa del sitio" funcional con rutas y controladores vacíos.
4. **Construir el Esqueleto Visual**: Diseñar la plantilla `base.html.twig` que usarán todas las páginas.

3. Paso 1: Tu Entorno Docker a Medida (Tu Tarea)

Cada TFG es único. Algunos necesitarán un servidor de correo, otros un motor de caché como Redis, o quizás una base de datos NoSQL. Es tu responsabilidad definir el entorno que tu proyecto necesita.

1. **Inicia tu proyecto Symfony**: Usa `symfony new` con la opción `--webapp`.
2. **Analiza y personaliza tu `docker-compose.yaml`**: Por defecto, Symfony te da Nginx, PHP y PostgreSQL. Si tu proyecto necesita MySQL, MariaDB o cualquier otro servicio, este es el momento de añadirlo. Deberás buscar la imagen de Docker adecuada y configurar sus variables de entorno y volúmenes.
3. **Configura el `.env`**: Una vez modificado el `docker-compose.yaml`, ajusta la variable `DATABASE_URL` en tu archivo `.env` para que Symfony sepa cómo conectarse a tu base de datos.
4. **Levanta el entorno y crea la BD**: Usa `docker compose up -d` y el comando de consola de Doctrine para crear la base de datos.

> ***Píldora de Investigación**: Tu TFG necesita enviar correos para registro o notificaciones. Investiga cómo añadir un servicio de **MailHog** a tu `docker-compose.yaml`. MailHog es un contenedor que captura todos los correos enviados por tu aplicación, permitiéndote verlos en una interfaz web sin enviarlos realmente. ¿Qué variables de entorno necesitarás ajustar en Symfony para que use MailHog?*

4. Paso 2: Del E/R a las Entidades Doctrine (Tu Tarea)

Este es el paso más crucial. Vas a tomar tu diagrama Entidad-Relación y lo convertirás en clases PHP que Doctrine pueda entender.

Tu Tarea:

Para **cada una de las entidades** de tu diagrama E/R:

1. Usa `make:entity` para generar la clase.
2. Añade cada atributo, eligiendo el tipo de dato de Doctrine más apropiado.
3. **Implementa las relaciones**: Cuando `make:entity` te pregunte por el tipo de campo, puedes escribir `relation`. Deberás elegir el tipo de relación (`ManyToOne`, `OneToMany`, `ManyToMany`) que corresponda con tu diagrama E/R.

> ***Píldora de Investigación (Relaciones):***
>
> - *Investiga la diferencia entre una relación **unidireccional** y **bidireccional** en Doctrine. ¿Cuándo deberías usar una sobre la otra?*
> - *¿Qué significan y qué implicaciones tienen las opciones `orphanRemoval=true` y `cascade={"persist"}` en una relación? Busca ejemplos en la documentación de Doctrine.*

Una vez que hayas creado **todas** tus entidades y relaciones, genera la migración que creará el esquema completo de tu base de datos y aplícala.

5. Paso 3: El Mapa de Rutas y Controladores (Tu Tarea)

Ahora, crea un esqueleto de la navegación de tu app. No necesitan tener lógica, solo existir para confirmar que la estructura de tu aplicación es coherente.

Tu Tarea:

Para cada página principal identificada en tu especificación técnica (p. ej., Home, Perfil, Listado de Productos, Contacto):

1. Crea un controlador apropiado usando `make:controller`.
2. Dentro del controlador, define un método con su correspondiente atributo `#[Route]`. Dale un nombre a la ruta (p. ej., `home`, `user_profile`).
3. El método debe, como mínimo, renderizar una plantilla Twig, aunque esta aún no exista o esté vacía. Por ejemplo: `return $this->render("profile/index.html.twig");`

El objetivo es tener todas las rutas principales de tu aplicación definidas y accesibles, creando un “mapa del sitio” navegable.

6. Paso 4: El Esqueleto Visual (`base.html.twig`) (Tu Tarea)

Finalmente, crea la estructura visual que compartirán todas tus páginas. Esto es fundamental para la consistencia y mantenibilidad de tu frontend.

Tu Tarea:

1. Abre y edita el archivo `templates/base.html.twig`.
2. Diseña la estructura HTML principal de tu aplicación. Debe incluir, como mínimo:
    - Un `<header>` o `<nav>` con la navegación principal. Los enlaces deben usar la función `path()` de Twig y los nombres de ruta que definiste en el paso anterior.
    - Un `<main>` donde se renderizará el contenido específico de cada página, usando `{% block body %}{% endblock %}`.
    - Un `<footer>` con la información de copyright o enlaces secundarios.
3. Define otros bloques que consideres necesarios, como `{% block javascripts %}` o `{% block stylesheets %}`.
4. Para que tu prototipo no se vea como HTML plano, enlaza una librería CSS. No es necesario que la diseñes tú. Puedes usar un framework ligero como Pico.css, Bootstrap, o Tailwind CSS si te sientes más aventurero.

Conclusión

Si has completado todos los pasos, ahora tienes un **prototipo de arquitectura sólido**. No hace mucho, pero la estructura está perfectamente definida. Tienes un entorno Docker funcional, una base de datos que coincide con tu E/R, y una navegación y layout básicos.

Estás listo para empezar a construir las funcionalidades de tu TFG sobre una base profesional.