1. `src/`

Esta carpeta contiene el código fuente de tu aplicación.

- **`Controller/`**: Aquí están los controladores, que se encargan de manejar las solicitudes del usuario (por ejemplo, cuando visitas una URL) y devolver una respuesta (HTML, JSON, etc.).
- **`Entity/`**: Define las entidades que representan tablas en la base de datos. Cada archivo suele estar vinculado a una tabla y tiene relaciones con otras entidades.
- **`Repository/`**: Proporciona métodos personalizados para interactuar con la base de datos, asociados a las entidades.
- **`Form/`**: Contiene clases para generar formularios y gestionar su lógica.
- **`Security/`**: Clases relacionadas con la seguridad, como autenticación y autorización.

Relación: Controladores usan entidades y repositorios para obtener datos, procesarlos y enviarlos a las vistas.

---

2. `templates/`

Contiene las vistas (HTML con Twig).

- Aquí defines el diseño visual que el usuario verá. Twig permite incluir lógica simple (como bucles o condiciones) en las plantillas.

Relación: Los controladores renderizan plantillas desde esta carpeta, pasando datos a Twig para su presentación.

---

3. `config/`

Contiene los archivos de configuración del proyecto.

- **`services.yaml`**: Define cómo se gestionan los servicios (clases) en la aplicación.
- **`routes.yaml`**: Gestiona las rutas (URLs) y qué controlador las manejará.
- **`packages/`**: Configuración específica de paquetes (por ejemplo, seguridad, base de datos, etc.).

Relación: Configura cómo interactúan los componentes entre sí, como los controladores y los servicios.

---

4. `public/`

Carpeta pública, accesible desde el navegador.

- **`index.php`**: Punto de entrada de la aplicación. Recibe todas las solicitudes y las redirige al kernel de Symfony.
- **Archivos estáticos (`css/`, `js/`, `images/`)**: Recursos públicos como estilos, imágenes y scripts.

Relación: `index.php` carga el núcleo de Symfony y distribuye las solicitudes según las rutas.

---

5. `var/`

Carpeta para datos temporales.

- **`cache/`**: Archivos generados por Symfony para acelerar el rendimiento.
- **`log/`**: Registra errores y eventos.

Relación: Symfony usa esta carpeta para guardar información temporal mientras ejecuta la aplicación.

---

6. `vendor/`

Contiene las dependencias instaladas con Composer (bibliotecas de terceros).

- Symfony Framework está aquí, junto con otras librerías necesarias para tu proyecto.

Relación: El núcleo de Symfony y otros paquetes son esenciales para que tu aplicación funcione.

---

7. `tests/`

Carpeta para tus pruebas automatizadas.

- Aquí defines casos de prueba para verificar que tu aplicación funciona como esperas.

Relación: Los tests usan las clases y funcionalidades de tu aplicación para comprobar su comportamiento.

---

8. `.env`

Archivo de configuración del entorno.

- Define variables como la conexión a la base de datos, el entorno (`dev`, `prod`), etc.

Relación: Configura parámetros clave que afectan a todo el proyecto.

---

9. `composer.json` y `composer.lock`

Archivos de Composer.

- **`composer.json`**: Lista las dependencias del proyecto.
- **`composer.lock`**: Detalla las versiones exactas de las dependencias instaladas.

Relación: Estos archivos permiten instalar y mantener las bibliotecas necesarias para tu proyecto.

---

10. `package.json`

Archivo de configuración para Node.js.

- Define las dependencias y scripts necesarios para gestionar recursos frontend como CSS, JavaScript, y otras herramientas modernas (por ejemplo, Webpack, Vite, o TailwindCSS).
- Incluye:
    - **`dependencies`**: Librerías necesarias en producción (ej.: Bootstrap, React).
    - **`devDependencies`**: Librerías necesarias solo durante el desarrollo (ej.: Webpack, Babel).
    - **`scripts`**: Tareas automatizadas que puedes ejecutar (ej.: compilar CSS, lanzar un servidor de desarrollo).

Relación: Se utiliza con herramientas como Webpack Encore o Vite para compilar y gestionar recursos frontend. Los resultados suelen colocarse en la carpeta `public/build`.

---

11. `Dockerfile`

Archivo de configuración para Docker.

- Define cómo construir una imagen Docker personalizada para tu proyecto.
- Incluye instrucciones para instalar PHP, Symfony, extensiones necesarias, servidores web (Nginx/Apache), y otras dependencias del sistema.

Relación: Permite ejecutar tu proyecto en un contenedor reproducible, asegurando que funcione igual en cualquier máquina.

---

12. `docker-compose.yaml`

Archivo de configuración para Docker Compose.

- Define y coordina varios servicios Docker necesarios para tu aplicación.
- Ejemplo típico:
    - **`php`**: Contenedor con PHP y Symfony.
    - **`database`**: Contenedor para la base de datos (ej.: MySQL, PostgreSQL).
    - **`nginx`**: Contenedor para el servidor web.
    - **`node`**: Contenedor para compilar recursos frontend.

Relación: Simplifica la ejecución de todos los servicios necesarios para tu proyecto con un solo comando (`docker-compose up`).

---

13. `webpack.config.js` (si usas Webpack Encore)

Archivo de configuración para Webpack.

- Define cómo se compilan y agrupan tus recursos frontend.
- Configura loaders para manejar diferentes tipos de archivos (CSS, JS, imágenes) y plugins para optimizar los resultados.

Relación: `package.json` y `webpack.config.js` trabajan juntos para producir archivos que se servirán desde `public/`.

---

14. `.gitignore`

Archivo que indica a Git qué archivos o carpetas no deben incluirse en el control de versiones.

- Ejemplos comunes:
  - **`/vendor/`**: Dependencias de Composer.
  - **`/node_modules/`**: Dependencias de Node.js.
  - **`/var/cache/`** y **`/var/log/`**: Archivos temporales.

Relación: Ayuda a mantener limpio el repositorio al ignorar archivos generados automáticamente.

---

15. `README.md`

Archivo de documentación.

- Explica cómo instalar, configurar y ejecutar tu proyecto.
- Puede incluir información sobre tecnologías usadas, dependencias y comandos importantes.

Relación: Es la primera referencia para entender y colaborar en el proyecto.

Resumen de la relación entre todos

1. **Backend (PHP/Symfony)**:
   - Se gestiona desde `src/` y se configura en `config/`.
   - Las entidades interactúan con la base de datos configurada en `.env` o `docker-compose.yaml`.
2. **Frontend (CSS/JS)**:
   - Usa `package.json` para dependencias y herramientas.

    - Los resultados se generan con `webpack.config.js` y van a `public/build`.
3. **Infraestructura (Docker)**:
    - `Dockerfile` y `docker-compose.yaml` coordinan cómo correr el proyecto en un entorno replicable.
4. **Control de versiones (Git)**:
    - `.gitignore` evita guardar archivos innecesarios.