

Proyecto Final FP Desarrollo de Aplicaciones Web.

Aplicación de gestión para Clínicas, ejemplificada con una Clínica
Psicológica.

Alicia Ortega Castro.

IES Francisco Ayala. 2024/2025.

Resumen.

El proyecto es una aplicación pensada para clínicas, con especial orientación a aquellas disciplinas sanitarias cuyos profesionales deben aplicar el tratamiento para el cliente en varias sesiones repartidas en el tiempo como son la psicología, fisioterapia o nutrición.

La aplicación permite la especificación dinámica de los **horarios** y **tratamientos** propios de la clínica y registro controlado de los **trabajadores** y de los nuevos **clientes** y creación de las correspondientes **citas**. Hay una administración completa de estas entidades que dependerá del rol del usuario y del paso del tiempo.

La aplicación ha sido desarrollada en Symfony principalmente. También se ha utilizado MySQL, Doctrine, Twig, AssetMapper, JavaScript, jQuery, Sass y Bootstrap.

Palabras clave: Symfony, MySQL, JavaScript, jQuery, Sass, clínicas.

Abstract.

The project is an application designed for clinics, with a particular focus on healthcare disciplines where professionals must administer treatment to clients over multiple sessions spread over time, such as psychology, physiotherapy, or nutrition.

The application allows for the dynamic specification of clinic **schedules** and **treatments**, as well as the controlled registration of **staff** and new **clients**, and the creation of the corresponding **appointments**. A complete management system for these entities is included, which will depend on the user's role and the passage of time.

The application has been developed mainly in Symfony. MySQL, Doctrine, Twig, AssetMapper, JavaScript, jQuery, Sass, and Bootstrap have also been used.

Keywords: Symfony, MySQL, JavaScript, jQuery, Sass, clinics.

Índice:

Resumen.....	2
Abstract.....	2
Justificación.....	5
1. Características generales.....	5
2. Restricciones generales.....	5
3. Aspectos a cubrir, junto con los que no se van a tratar.....	5
4. Estudio de las prestaciones de la herramienta que se propone frente a otras existentes en la misma categoría.....	5
Justificación de la tecnología empleada.....	6
Symfony.....	6
Doctrine.....	6
MySQL.....	7
Twig.....	7
Asset Mapper.....	7
JavaScript y jQuery.....	7
Sass.....	7
Bootstrap.....	8
Requerimientos hardware y software.....	8
Análisis y diseño.....	9
Diagrama de casos de uso.....	9
Diagrama de clases.....	10
Base de Datos.....	11
Modelo Entidad/Relación.....	11
Paso a Tablas.....	11
Interacciones con la base de datos desde php según el rol del usuario.....	12
Administrador.....	12
Terapeuta.....	12
Cliente.....	12
Implementación.....	12
Assets.....	12
Hojas de estilo.....	13
Archivos de Configuración.....	13
Migraciones.....	13
Código fuente.....	13
Controller.....	14
Data Fixtures.....	14
Entidades.....	14
Formularios.....	15
Repository.....	15
Services.....	15
Plantillas.....	16
Variables de Entorno.....	16
Importmap.....	16

Evaluación y pruebas.....	17
Pruebas sin sesión iniciada.....	17
Pruebas con Sesión Iniciada.....	17
Cambio de contraseña.....	17
Creación de un nuevo User.....	18
Borrado de usuario.....	19
Edición de usuarios.....	19
Creación de citas.....	20
Prueba de Ajax: Los tratamientos propios del terapeuta.....	20
Manual de estilos.....	21
Sketches.....	21
Página de Inicio.....	21
Área de Usuario: Página con lista y tabla.....	22
Área de Usuario: Página con formulario y tabla.....	22
Criterios de accesibilidad.....	23
Criterios de usabilidad.....	23
Tipografía.....	24
Mapa de colores de proyecto en tres formatos (RGB, Hexadecimal, nombre junto con el color).	
Elementos donde se aplica.....	24
Azul-oscuro.....	25
Azul-claro.....	25
Verde.....	25
Blanco.....	25
Negro.....	25
Dispositivos/vistas para las que se ha diseñado el proyecto.....	26
Software utilizado.....	26
Composer.....	26
Visual Studio Code.....	26
Symfony CLI.....	26
XAMPP.....	26
GitHub.....	26
Mejoras posibles y aportaciones.....	26
Bibliografía.....	28

Justificación.

1. Características generales.

En el aspecto técnico hay un especial énfasis en la parte backend de la aplicación por lo que se centra especialmente en la administración de las entidades y las relaciones entre sí. Dichas entidades son: User, Terapeuta, Cliente, Tratamiento, Horario, Cita. De todas ellas se cubre la creación, lectura, actualización y borrado (componiendo las siglas CRUD en inglés), atendiendo al rol del usuario.

El apartado de seguridad se cubre gracias a Symfony tanto por su metodología propia de login, su clase User, sus formularios propios y por la configuración en los archivos .yalm. En dichos archivos .yalm, por ejemplo, se restringe el acceso a ciertas URIs según el rol del usuario activo.

En cuanto al porqué de esta aplicación se busca cubrir una gestión de citas básica para pequeñas clínicas que no disponen de los recursos necesarios para acceder a una gestión de citas de tipo más desatendido para los profesionales sanitarios. Además se busca cubrir una capacidad mínima de gestión administrativa sobre los tratamientos, horarios y perfiles de los trabajadores. Esta es una herramienta implementable en una web ya existente con ciertos cambios y personalizaciones pero se le ha añadido un modelo de *home* para dar la sensación de una web completa e independiente.

2. Restricciones generales.

No es una aplicación pensada para gestionar una masiva cantidad de datos. Está pensada para pequeñas clínicas.

3. Aspectos a cubrir, junto con los que no se van a tratar.

Se cubre un inicio y cierre de sesión, capacidad de cambiar la contraseña, aquellos aspectos CRUD para cada entidad, control de accesos, y actualización del atributo 'estado' de las citas por el paso del tiempo.

4. Estudio de las prestaciones de la herramienta que se propone frente a otras existentes en la misma categoría.

Profesionales sanitarios autónomos y pequeñas clínicas tienden a utilizar portales como Doctoralia para anunciarse y como sistema para gestionar citas; sin embargo esta gestión de citas tiende a ser inflexible ya que para cancelar usualmente debe complementarse con un nuevo contacto entre terapeuta y cliente.

Otras veces, cuando el terapeuta y cliente ya han tenido al menos una cita, se utiliza un sistema de calendario compartido como Google Calendar, lo cual permite mayor flexibilidad pero transmite poca profesionalidad.

La aplicación desarrollada para este proyecto cubre esas necesidades básicas de las pequeñas clínicas de tener un lugar para anunciarse y a la vez tener una gestión suficiente de las citas.

Además cumple más funciones administrativas al permitir cierta gestión sobre los usuarios y los datos descriptivos propios.

No es tan completa como otras herramientas de pago y con años de trayectoria pero estas tienden a especializarse en gestión de citas, publicitar negocios o administrar personal. No engloban características básicas comunes a estos tipos de negocios.

Justificación de la tecnología empleada.

Symfony

Symfony es uno de los frameworks de PHP más populares hoy día y es común encontrarlo en las ofertas de empleo de backend por las ayudas que provee para los procesos comunes como por ejemplo serían el login, sesiones, caché, GET o POSTs. Además, por su implementación de Doctrine es sencillo construir la base de datos y trabajar con las Entidades como si fueran Clases y Objetos. Sin embargo para aplicaciones pequeñas podría funcionar también como framework full-stack por su uso de Twig, Asset Mapper, WebPack Encore y otras herramientas centradas en los estilos o en la velocidad de ejecución de la aplicación gestionando cachés.

Las aplicaciones construidas con Symfony son más fáciles de mantener a largo plazo que aquellas que son hechas puramente en PHP. Symfony renderiza las plantillas y llama funciones fuera de sus Controllers mediante urls. Las rutas no son construidas a partir del nombre del archivo donde se encuentran como pasa en PHP, sino que gracias a las etiquetas #Route se le puede adjudicar una ruta a una función para que esta sea llamada fácilmente desde cualquier parte de la aplicación. Esto al final resulta en una distribución más ordenada y lógica del código a través de la carpeta src/, por lo que al entender la lógica de negocio es fácil orientarse entre los directorios.

A continuación voy a detallar estas tecnologías que acompañan a Symfony y que se han utilizado en el proyecto.

Doctrine

Doctrine es un ORM (object-relational mapper), y como he comentado: es lo que permite construir e interactuar con la base de datos con facilidad al trabajar con las Entidades como si fueran Clases.

No es exclusivo de Symfony, pero se hace más fácil de utilizar gracias al framework. Por ejemplo, si en consola escribimos “php bin/console make:entity” se iniciará una guía para construir la entidad como equivalente a una tabla de Sql y al terminar el proceso se habrán generado automáticamente dos archivos, uno en Entity/ y otro en Repository/. El archivo en Entity/ estará construido ya para poder trabajar con Doctrine y para guardarlo en la Base de Datos que tengamos será tan fácil como ejecutar los comandos “php bin/console make:migration” y “php bin/console doctrine:migrations:migrate”. No es necesario conocer de memoria los comandos, la terminal te sugiere el siguiente comando necesario o puedes escribir simplemente “php bin/console” para ver todas las opciones.

Otra de las grandes ventajas de Doctrine es que es agnóstico en cuanto a base de datos. No requiere aplicar una sintaxis diferente según el tipo de base de datos, por lo que el código que interactúe con la base de datos es más eficiente y reutilizable.

MySQL

Aunque realmente no acompaña nativamente a Symfony, en el archivo `.env` de las primeras cosas que tendremos que hacer es establecer la conexión con la base de datos que vayamos a utilizar. En mi caso he utilizado MySQL pero hubiera funcionado con otras también.

Symfony se encargará de establecer conexión y cerrarla en cada interacción con la base de datos de forma automática.

Twig

Es el equivalente a HTML que se utiliza con Symfony. Permite cierta capacidad de programación sin tener que añadir bloques de php. Tiene una sintaxis particular pero sencilla que permite condicionales y bucles. Además tiene acceso a la variable global “app” donde se guarda información de la sesión, las peticiones (request) y del usuario, aparte de las variables pasadas por el Controller.

Aunque no es necesario permite trabajar con “bloques” intercambiables. Muy útiles para aquellas partes de html que van a ser frecuentes en la mayoría de las páginas pero no en todas o que requieran personalización; como es el caso de los encabezados. Para ello es importante la correcta estructuración del archivo `base.html.twig`, del que van a extender las demás plantillas.

Asset Mapper

Es un empaquetador para la parte de frontend, equivalente a Parcel o a WebPack pero es propio de Symfony. La documentación de Symfony también explica WebPack Encore que trae una capa para optimizar su funcionamiento y es el recomendado para proyectos grandes.

En este proyecto se ha utilizado Asset Mapper ya que en la propia documentación del framework lo anuncian como el ideal para proyectos pequeños. Tiene la particularidad de no necesitar npm para instalar paquetes y de utilizar `importmap` para añadir las librerías o los archivos necesarios.

JavaScript y jQuery

Necesarias para mejorar la experiencia del usuario en cuanto a la interactividad de la aplicación. Se han utilizado para la página que incluye AJAX, aplicar clases específicas, controlar una cookie, divs de tipo *modal* y el DOM según correspondiera, así como aplicar eventos.

Sass

Más flexible y ágil que CSS. Aunque CSS permite tener variables para su reutilización, SASS es más flexible en cuanto a la reutilización gracias a los mixins.

Además de lo ágil que resulta establecer los estilos de forma anidada o en árbol o que si se necesitara permite realizar algunas operaciones matemáticas para determinar un valor.

Bootstrap

Es un pequeño framework de frontend que ayuda a aplicar estilos modernos y ágiles en poco tiempo. Para ello se deben establecer las estructuras y las clases adecuadas. Tiene una librería de componentes muy completa.

Requerimientos hardware y software.

Se ha utilizado Symfony 7.2, la cual requiere al menos PHP 8.2 (con las extensiones Ctype , iconv , PCRE , Session , SimpleXML y Tokenizer. Normalmente vienen ya habilitadas) y Composer. También es necesario Doctrine para que la aplicación funcione.

Si se quiere desplegar localmente la aplicación es altamente recomendable instalar también Symfony CLI para usarlo como servidor, sobre todo si el sistema operativo es Windows y el servidor Apache ya que los tiempos de ejecución son significativamente lentos en un entorno de esas características. Igualmente ejecutando el comando *symfony check:requirements* en la propia terminal aparecerán los requisitos a cubrir o las recomendaciones para mejorar la experiencia.

Con XAMPP recién instalado en septiembre de 2024, el cual trae una versión de php 8.2; y con composer actualizado, el comando nos dice que el equipo está preparado para symfony pero nos da las siguientes recomendaciones:

** intl extension should be available*

> Install and enable the intl extension (used for validators).

** a PHP accelerator should be installed*

> Install and/or enable a PHP accelerator (highly recommended).

** realpath_cache_size should be at least 5M in php.ini*

> Setting "realpath_cache_size" to e.g. "5242880" or "5M" in

> php.ini may improve performance on Windows significantly in some*

> cases.

** "post_max_size" should be greater than "upload_max_filesize".*

> Set "post_max_size" to be greater than "upload_max_filesize".

La extensión intl es muy útil cuando se quiere que una app funcione para clientes de diferentes regiones, pero como no es el caso no ha sido habilitada.

Como acelerador de php viene preinstalado OPcache, pero deshabilitado, en principio no necesita modificación.

Pongo `realpath_cache_size` en 5 M en el `php.ini`. También hago que el `"post_max_size"` sea mayor que el `"upload_max_filesize"`.

Como parte de las recomendaciones también aparece una nota para recordar que el entorno de desarrollo y producción deben tener la misma versión y configuración para PHP.

Análisis y diseño.

Diagrama de casos de uso

En el diagrama de Casos de Uso ya se puede observar que de los tres roles que puede tener un usuario el Admin es el que más puede interactuar con la aplicación para la realización de sus funciones administrativas de personal, gestión de los tratamientos y horarios propios de la clínica y crear citas.

Los Terapeutas pueden administrar las cuestiones directamente relacionadas con ellos, como son sus tratamientos, sus horarios o la creación de clientes para los que están realizando un tratamiento.

Los Clientes por su parte son los que menos pueden interactuar limitándose a ver la información que les afecte y poder cancelar citas.

El tiempo afecta a las citas.

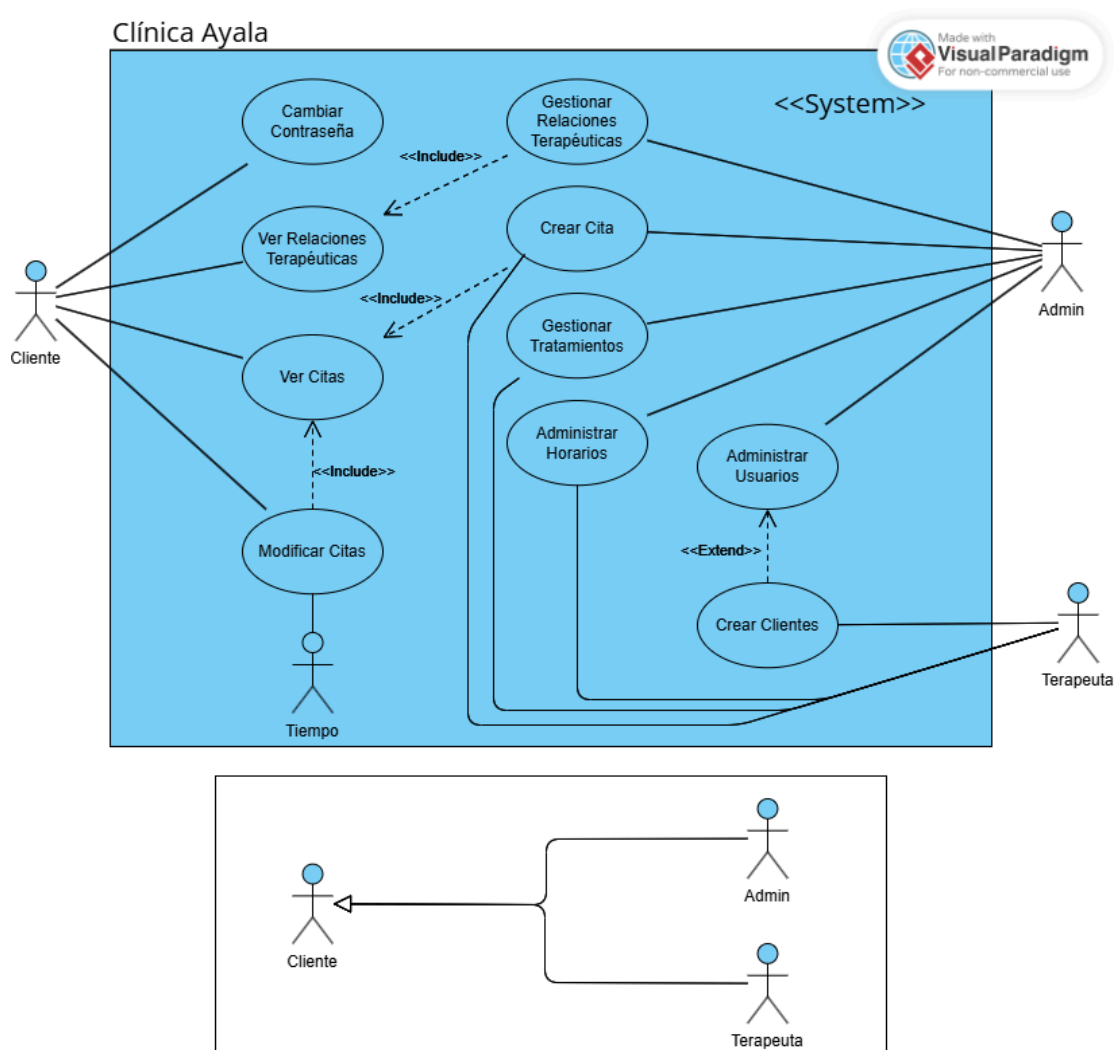
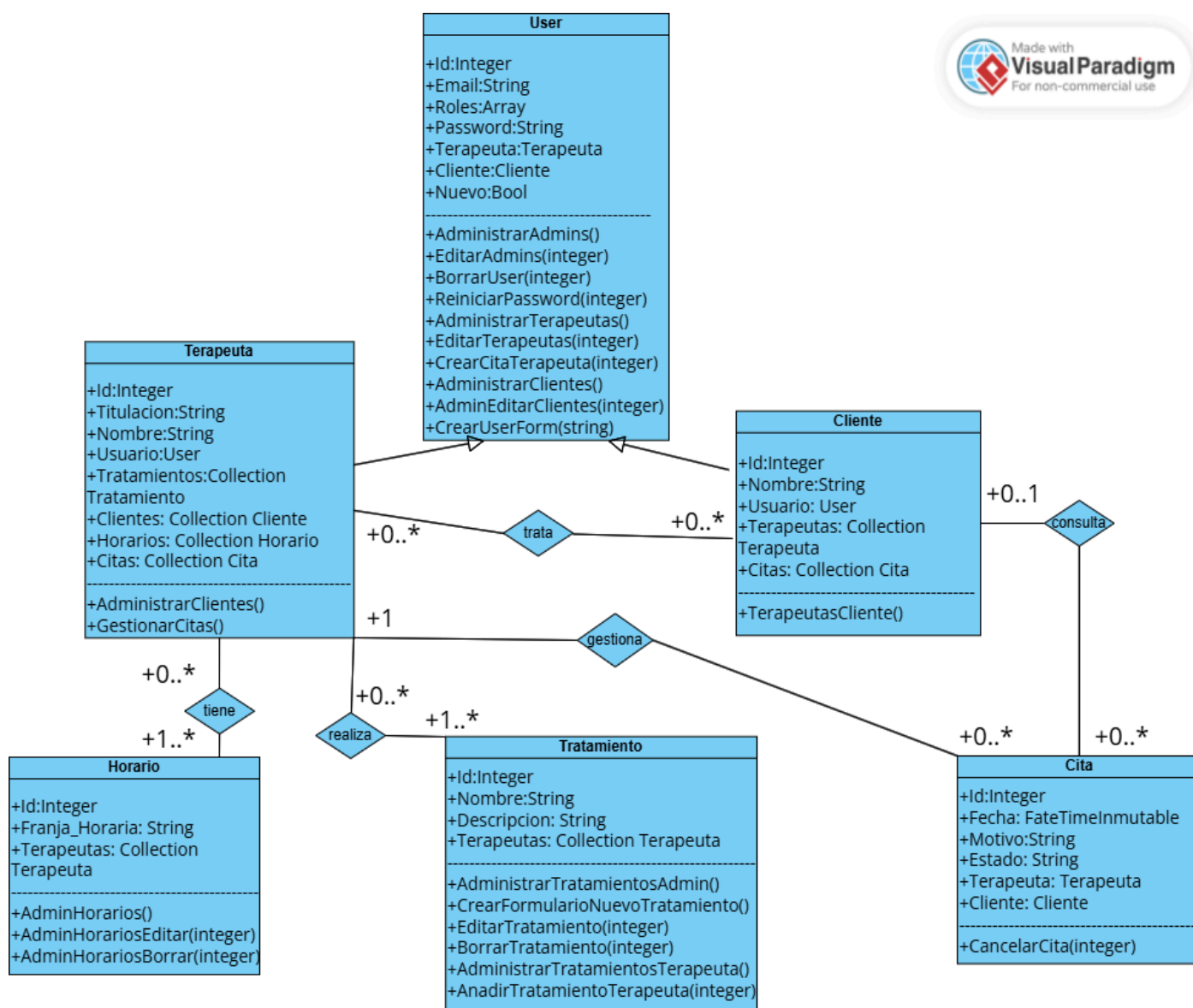


Diagrama de clases

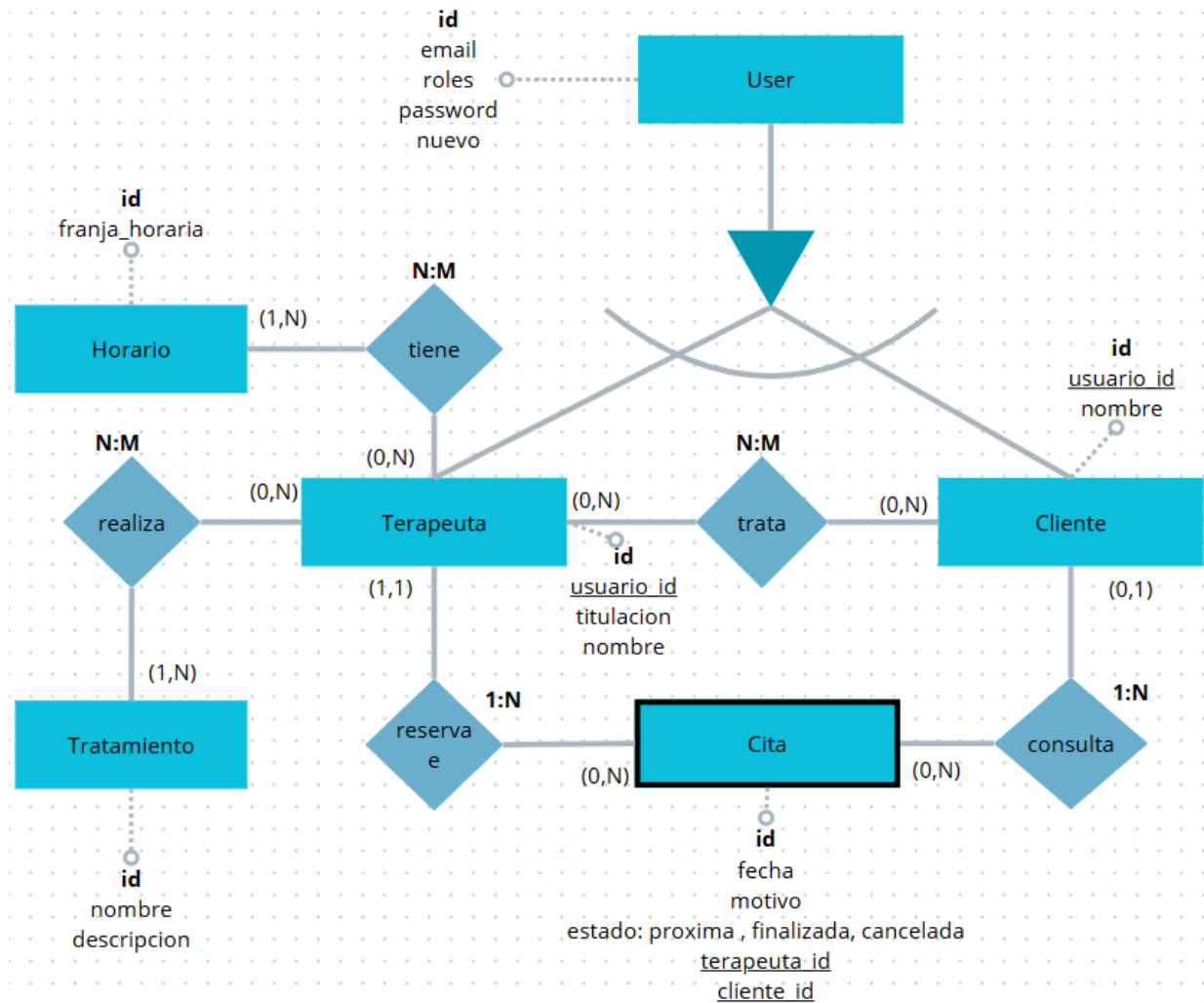
Hay que tener en cuenta que Symfony es un framework con orientación a objetos y que sigue la filosofía: Modelo-Vista-Controlador. Tanto los Modelos como los Controladores son clases. Cada Modelo es una Entidad y contiene la parte de los atributos y por cada Modelo hay un Controlador que contiene las funciones. He unificado los Modelos (Entities) y Controladores (Controllers) para hacer el diagrama.

De mi estructura hay que tener en cuenta que las funcionalidades del User dependen de su rol. El Admin, Terapeuta o el Cliente no pueden hacer lo mismo, pero puesto que el Admin tiene los mismos atributos que User no es necesario crear una entidad para Admin y se usa User con rol de administrador.



Base de Datos

Modelo Entidad/Relación



Paso a Tablas

Siendo el texto en negrita las claves primaria y el subrayado las claves foráneas, tras el paso a tablas quedaría lo siguiente:

1. Usuario: **id**, email, roles, password, nuevo.
2. Terapeuta: **id**, usuario_id, titulación, nombre.
3. Cliente: **id**, usuario_id, nombre.
4. Tratamiento: **id**, nombre, descripción.
5. Cita: **id**, fecha, motivo, estado (próxima, finalizada, cancelada), terapeuta_id, cliente_id.
6. Horario: **id**, franja_horaria.
7. terapeuta_horario: **terapeuta_id**, **horario_id**.
8. terapeuta_cliente: **terapeuta_id**, **cliente_id**.
9. tratamiento_terapeuta: **id_tratamiento**, **terapeuta_id**.

Interacciones con la base de datos desde php según el rol del usuario

Administrador

- Cambiar su contraseña
- Restaurar el acceso a la cuenta para otros usuarios cuando olvidan su contraseña.
- Crear cuentas de administradores y terapeutas.
- Eliminar cuentas de otros usuarios.
- Editar cuentas de otros usuarios.
- Ver próximas citas de un terapeuta.
- Reservar cita para clientes.
- Cambiar a un cliente de terapeuta o añadir otro terapeuta al cliente.
- Cambiar horario de un terapeuta
- Añadir y eliminar tratamientos de la lista de la clínica.

Terapeuta

- Cambiar contraseña.
- Ver sus citas.
- Reservar citas para clientes.
- Crear cuenta para nuevo cliente si necesita continuidad en el tratamiento.
- Cancelar citas.
- Añadir y eliminar tratamientos que realiza de la lista de la clínica.

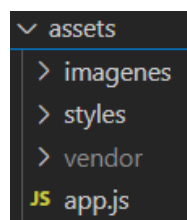
Cliente

- Cambiar contraseña.
- Ver sus terapeutas.
- Ver sus próximas citas.
- Cancelar citas.

Implementación..

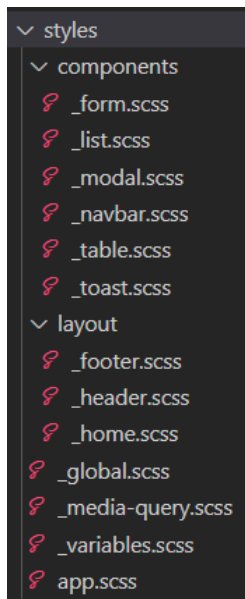
A continuación se va a describir la organización de los archivos y su función para la implementación de la aplicación siguiendo el orden de aparición de los directorios contenedores.

Assets



En los proyectos de Symfony las imágenes, estilos y archivos .js normalmente se colocan en el directorio public/, pero al usar Asset Mapper se colocan en el directorio assets/ el cual puede ser llamado desde los archivos de twig. En el caso de app.js puede ser incluido en los archivos twig mediante un importmap.

Hojas de estilo



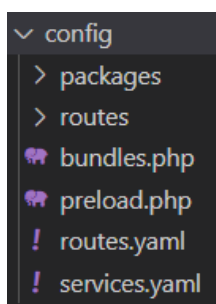
Se encuentra en el directorio `assets/`. Se han dividido los archivos `.scss` en tres bloques: Globales (en la carpeta `styles/` directamente), Layout y Components.

Los archivos globales afectan a toda la web: `app.scss` llama a todos los demás archivos, `_variables.scss` contiene los colores, puntos de ruptura para el diseño responsive y las fuentes, `_media-query.scss` contiene los `@mixins` utilizados para hacer el diseño responsive, y `_global.scss` describe los estilos que afectan a varios elementos.

El directorio `layout/` contiene archivos de estilos según la etiqueta `html` en la que se aplican, son estilos que no pertenecen a un componente diferenciado.

Por su parte en el directorio `components/` se encuentran los estilos diferenciados por componente. Así aunque el `navbar` está solo en el `header`, por ser un componente específico está dentro de este directorio en vez de en el archivo `_header.scss`.

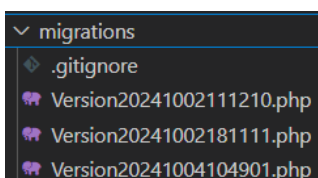
Archivos de Configuración



En el directorio `config/` hay numerosos archivos de configuración. Voy a detenerme en el `config/packages/security.yaml` ya que en este archivo podemos determinar a dónde redirigimos si un usuario sin sesión iniciada está intentando acceder a una url para la que se requiere haber realizado un login, o a dónde redirigimos después de cerrar sesión.

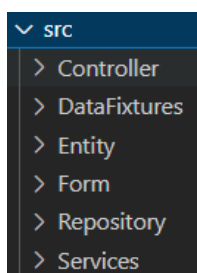
Es especialmente interesante configurar un `access_control` para aquellas urls dónde además de requerir una sesión iniciada queramos limitar el acceso a roles concretos.

Migraciones



Cada vez que se cree o modifique una entidad debe ejecutarse el comando `php bin/console make:migration` para generar estos archivos de versiones que cuando se ejecuten aplicarán los cambios en las tablas de la base de datos. Actúa como un control de versiones de las tablas ya que permite deshacer los cambios o ejecutar hasta una versión específica.

Código fuente

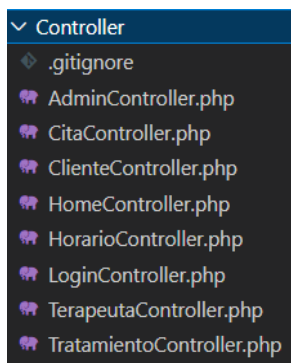


En el directorio `src/` se encuentra “lo que hace” la aplicación. Symfony sigue la metodología Modelo-Vista-Controlador; aquí está el *Modelo* en `Entity/` y el *Controlador* en `Controller/`, (la *Vista* está en `templates/`, fuera de `src/`).

Hay un archivo en `Repository/` por cada uno generado en `Entity/`.

A continuación voy a describir lo más importante de cada uno.

Controller



Es la parte de *Controlador* y contiene las funcionalidades.

Es una práctica común que en el directorio Controller se contenga un archivo por cada entidad por lo que se ha seguido esa indicación y además se ha intentado organizar los archivos también por el rol de usuario que ejecuta la función, por ello no hay un archivo User pero sí Admin. Así por ejemplo en el archivo AdminController.php hay funciones sobre terapeutas debido a que son funciones que solo podrían ejecutar los usuarios con rol Admin.

También hay un archivo para funcionalidades muy específicas como son el caso de HomeController.php y LoginController.php.

Definitivamente es uno de los directorios donde más tiempo se invierte en el desarrollo de una aplicación.

Data Fixtures

Este directorio contiene únicamente el archivo InitialAdminFixture.php.

Por la lógica de negocio la aplicación no tiene un formulario de registro autónomo para usuarios por lo que la aplicación necesitaba traer por defecto un usuario de rol Admin preconfigurado. Se podría haber generado ese usuario inicial manualmente una vez generadas las tablas, pero dicho sistema no le proporciona autonomía al supuesto cliente para quien estuviésemos desplegando la aplicación.

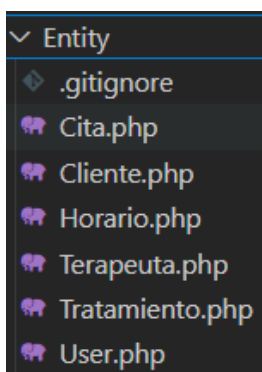
Una forma para arreglar esto de forma limpia y segura es teniendo datos preconfigurados; eso son los data fixtures y aunque están pensados para entornos de desarrollo puede utilizarse para producción.

En InitialAdminFixture.php está la función para crear este usuario inicial. La contraseña se determina en el .env y es el motivo por el que a github he subido un .env.example: para que esa supuesta contraseña no fuera pública.

Para tener acceso a la contraseña desde InitialAdminFixture.php en el archivo config/services.yaml hay que añadir en 'parameters' la siguiente línea: `app.super_admin_password: "%env(SUPER_ADMIN_PASSWORD)%"`.

Y así podemos usarla en el código fuente con: `params->get('app.super_admin_password')`.

Entidades

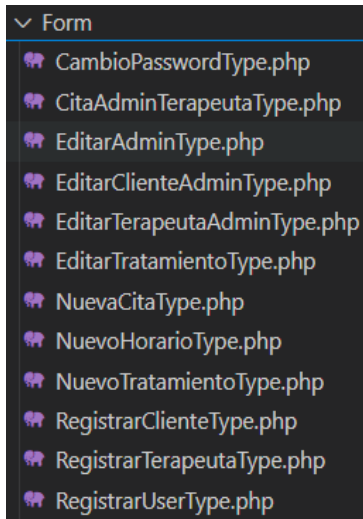


Son la parte de *Modelo*. Son Clases y equivalen con matices a las tablas que se van a generar tras la migración de doctrine.

Como digo no son exactamente iguales a las tablas porque dentro de las clases se puede establecer si hay relación con otras entidades y el tipo de relación (de muchas a muchas, de uno a muchas y demás). Dependiendo del tipo de relación se generarán tablas de tipo 'entidad1-entidad2' sin un reflejo exacto en el directorio Entity/, a no

ser que se hubiera realizado el diseño creando esas entidades que establecen las relaciones, pero en este caso se ha optado por reflejar las entidades en sí y no las tablas.

Formularios

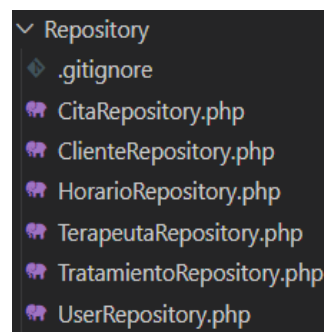


Todos los formularios son controlados en funciones de su correspondiente Controller y renderizados como parte de una plantilla, sin embargo son descritos en los archivos del directorio /Form según los inputs y datos que requieran.

El único formulario que no sigue esta estructura es el formulario de login, que al ser generada la plantilla a través del comando `php bin/console make:security:form-login` ya introduce directamente en la plantilla el formulario.

El resto de formularios de la aplicación son para crear entidades o para editarlas por lo que cuando se ejecuta el comando `php bin/console make:form` lo primero que pregunta la guía de creación es la entidad en la que se basa el formulario.

Repository



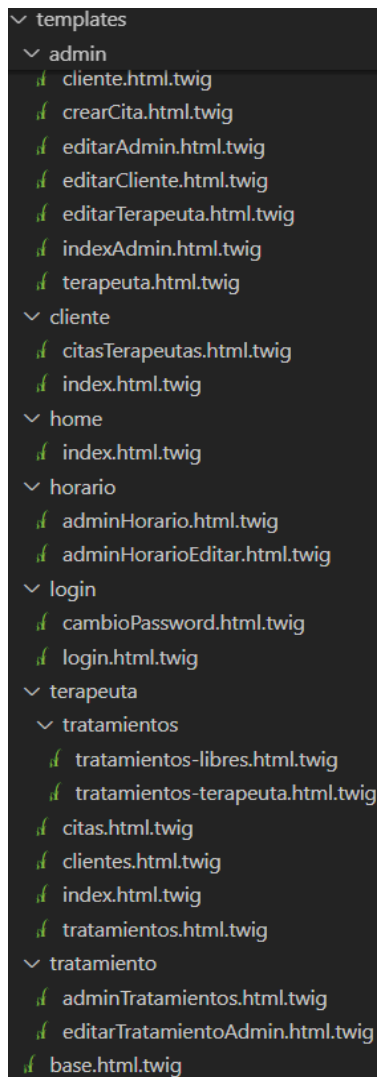
Este directorio y cada archivo se genera en el momento en el que generamos entidades con el comando correspondiente. Contiene las funciones que realizan las consultas a la base de datos. Las funciones pueden escribirse con:

- SQL, lo cual es la opción menos recomendada por ser la más vulnerable.
- Doctrine Query Language: usa una sintaxis parecida a la SQL pero con entidades en vez de con tablas.
- Query Builder. Es una opción muy cómoda ya que trabaja con objetos. Por ejemplo: `where('c.cliente = :clienteId')`

Services

En este directorio se colocan aquellas funciones que van a utilizar más de un controlador. En mi caso está el archivo UserService.php que contiene la función `crearUser(User $user, string $rol)` y se utiliza en el AdminController.php y en TerapeutaController.php.

Plantillas



Las plantillas están en el directorio `templates/` y son archivos `.html.twig`. Ejecutando el comando `composer require symfony/twig-bundle` el framework crea el directorio y el archivo `base.html.twig` del cual extienden todos los demás. Se podría considerar el “frontend nativo” de Symfony por su integración.

No es necesario tener un directorio por entidad, pero es práctica común tener uno por controlador relacionado. En este caso no se ha realizado un directorio específico para las renderizaciones relacionadas con citas porque van integradas en las renderizaciones de los terapeutas y clientes.

No es necesario tener un archivo por página ya que cada renderización de página completa puede estar compuesta por varios archivos. Por ejemplo: los archivos que en el nombre contengan la palabra ‘index’ son los encabezados y se añaden a los archivos correspondientes de contenido.

El directorio `terapeuta/tratamientos/` contienen dos tablas que se usan en `tratamientos.html.twig` como parte del AJAX. La página con URI `/terapeuta/tratamientos` es la única dónde se ha aplicado el método AJAX por lo que las plantillas relacionadas siguen una distribución diferente.

Variables de Entorno

En el apartado de Data Fixtures ya he comentado un poco del archivo `.env`. Aquí se introduce si estamos en desarrollo o en producción, la variable ‘`APP_SECRET`’ que utiliza Symfony para securizar, cookies, tokens y demás (se puede generar una con el comando de Symfony CLI: `symfony console secret:generate`), la ruta de conexión con la base de datos, la cola de mensajes DNS (lo utiliza Doctrine) y otras variables de entorno que se requieran. En este caso se estableció una contraseña para el primer administrador.

Importmap

Al habilitar el Asset Mapper se creó este archivo. Aquí se establecen las librerías de terceros que se han añadido mediante el comando `php bin/console importmap:require nombre-libreria`.

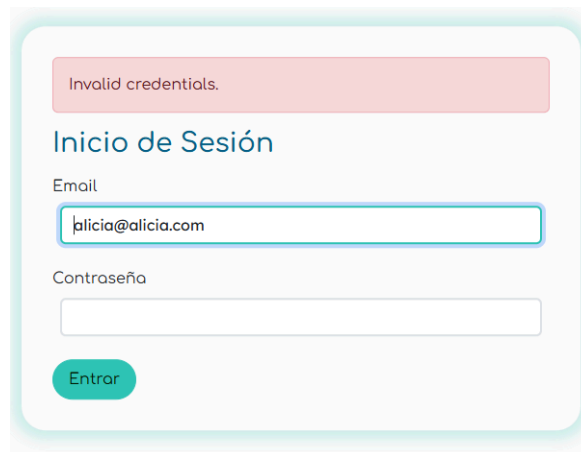
Evaluación y pruebas.

Las pruebas se han realizado manualmente y se han añadido mensajes de error para cuando sean requeridos.

Además Symfony tiene sistemas de comprobación y retroalimentación. Por ejemplo, cuando se usan los formularios del framework este controla que un email siga la estructura adecuada o que se rellenen los campos obligatorios.

Pruebas sin sesión iniciada.

Cuando el login es incorrecto aparece un mensaje de error.



The screenshot shows a login form titled "Inicio de Sesión". At the top, there is a red error message box that says "Invalid credentials.". Below the title, there are two input fields: "Email" with the value "alicia@alicia.com" and "Contraseña" (password). At the bottom, there is a green "Entrar" (Enter) button.

Cuando se intenta acceder a una URL que necesita sesión iniciada la aplicación redirige al formulario de login.

Cuando es una URL reservada para un usuario con un rol específico sin tenerlo sale una pantalla de "Access Denied" o de error 403 según esté en desarrollo o producción.

Pruebas con Sesión Iniciada

Cambio de contraseña

Si no coinciden aparece un mensaje de error.



The screenshot shows a password change form titled "Cambio de Contraseña". At the top, there is a message that says "Las contraseñas no coinciden" (The passwords do not match). Below the title, there are two input fields: "Nueva contraseña*" (New password*) and "Confirmar contraseña*" (Confirm password*). At the bottom, there is a green "Guardar" (Save) button and a link that says "Volver al espacio personal" (Return to personal space).

Creación de un nuevo User

Requiere introducir los campos en el formato requerido y los campos obligatorios.

The first two screenshots show the email field with the value 'blablabla'. The first shows an error message: 'Incluye un signo "@" en la dirección de correo electrónico. La dirección "blablabla" no incluye el signo "@".' The second shows the value 'blablabla@' and an error message: 'Introduce texto detrás del signo "@". La dirección "blablabla@" está incompleta.'

The next two screenshots show the email field with the value 'blablabla@blabla'. The third shows a success message: 'La contraseña provisional será lo que haya antes del @ del email y se deberá cambiar tras el primer acceso.' and a checkbox for 'Selecciona esta casilla de verificación si quieres continuar'. The fourth shows the 'Email*' field filled with 'blabla@blabla', the 'Titulación*' field empty, and the 'Nombre*' field with a 'Completa este campo' error message.

Si el email está en uso aparece un mensaje de error.

The left screenshot shows the email field with 'pruebas@pruebas.es' and a success message. The right screenshot shows the same email and a success message, but with an error message above it: 'Este correo ya está en uso.'

Cuando introduzco todos los campos aparece en la tabla, en este caso es un terapeuta:

Correo Electrónico	Nombre	Titulación	Tratamientos	Horario	Número de Clientes	Citas Próximas	Acciones
blabla@blabla	Borrar Borrar Borrar	Para Borrar			0		Nueva cita Borrar Editar

Compruebo que aparece en la base de datos

Recorte de la tabla User

<input type="checkbox"/>	Editar	Copiar	Borrar	74 blabla@blabla	["ROLE_TERAPEUTA"]
--------------------------	------------------------	------------------------	------------------------	------------------	--------------------

Recorte de la tabla Terapeuta

40	74 Para Borrar	Borrar Borrar Borrar
----	----------------	----------------------

Borrado de usuario

Como borrar un usuario, cancelar una cita, reiniciar contraseña o borrar un tratamiento es una acción irreversible aparece este mensaje de confirmación.



Si pulso cancelar el mensaje desaparece sin hacer nada, si confirmo el terapeuta desaparece de la tabla de la web y de la base de datos

Edición de usuarios

Voy a editar este terapeuta realizando cambios en todos los campos, he añadido, eliminado, dejado vacío o cambiado el texto

pruebas@pruebas.com	pruebas	pruebas	Caries	pruebasss	2	Nueva cita
			Blablaba			Borrar
			Depresión			Editar
			Lolailo			Reiniciar Contraseña

pruebas@pruebas.es	Pruebas De Edición	Pruebas de edición	Sábados de 9h a 12h	pruebasss	3	Nueva cita
			Blabla a las 2h			Borrar
						Editar
						Reiniciar Contraseña

Ahora voy a reiniciar la contraseña por lo que desaparecerá esa acción

pruebas@pruebas.es	Pruebas De Edición	Pruebas de edición	Sábados de 9h a 12h	3	Nueva cita
			Blabla a las 2h		Borrar
			pruebasss		Editar

Voy a cerrar sesión e iniciar como pruebas@pruebas.es, que antes era pruebas@pruebas.com.

Inicio de Sesión

Email

Contraseña

Entrar

Cambio de Contraseña

Nueva contraseña*

Confirmar contraseña*

Guardar Volver a la página de la Clínica

Redirige automáticamente a cambio de contraseña ya que en las pruebas como administradora se la había reiniciado. Realmente ya ha iniciado sesión por lo que puede posponer el cambio de contraseña si conoce la URL a la que quiere acceder para la que tiene permiso pero desde la página de inicio hasta las páginas de su rol se le va a recordar que cambie de contraseña cada vez hasta que la cambie. Puede ser la misma que la provisional, lo importante es que complete el formulario correctamente.

Creación de citas

Nueva Cita

Fecha y hora*

Motivo de la cita

Cliente*

Guardar

Error: La fecha de la cita no puede ser anterior a la fecha actual.

Si intenta crear una cita con una fecha y hora posterior a la actual no la crea y aparece un mensaje de error

Si es posterior la crea:

Fecha	Estado	Motivo	Cliente	Acciones
23-01-2025 21:07	pendiente		Cliente1	Cancelar Cita

Prueba de Ajax: Los tratamientos propios del terapeuta

Hace los cambios sin recargar página.

Tratamientos	
Mis Tratamientos	Acciones
Ansiedad	Quitar
Ansiedad1	Quitar

Tratamientos Disponibles	
Tratamientos Disponibles	Acciones
Blablabla	Añadir
Caries	Añadir
Depresión	Añadir
Dolor	Añadir
Drghasrghs	Añadir
Lolailo	Añadir

Tratamientos	
Mis Tratamientos	Acciones
Ansiedad	Quitar
Blablabla	Quitar

Tratamientos Disponibles	
Tratamientos Disponibles	Acciones
Ansiedad1	Añadir
Caries	Añadir
Depresión	Añadir
Dolor	Añadir
Drghasrghs	Añadir
Lolailo	Añadir

Manual de estilos.

Sketches

La página dónde más diseño se va a aplicar es en la página de inicio. Las demás páginas van a ser visualmente sencillas ya que las que implican un inicio de sesión tienden a ser de dos posibles formas: una lista seguida de una tabla, o un formulario seguido de una tabla.

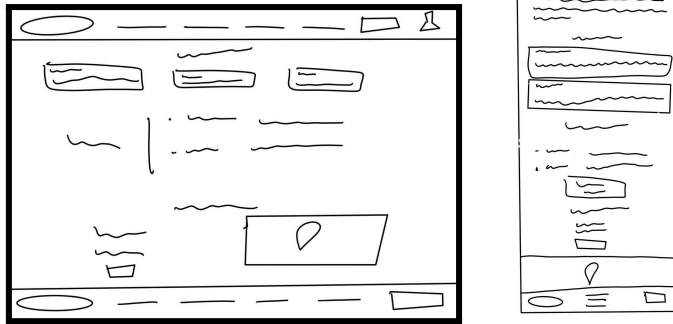
Por ello he realizado sketches de pantallas pequeñas y grandes de la página de inicio, de una página con lista y tabla y otra con formulario y tabla.

Página de Inicio

En el encabezado está el logo, el menú que lleva a las diferentes partes del body de esta página (en pantallas pequeñas el menú es desplegable y está a la izquierda), un botón de pedir cita y el botón que lleva al área personal o al inicio de sesión).

En el cuerpo están los tratamientos que se realizan, los terapeutas y las formas de contacto.

En el footer aparecen los mismos elementos que en el header salvo por el inicio de sesión.

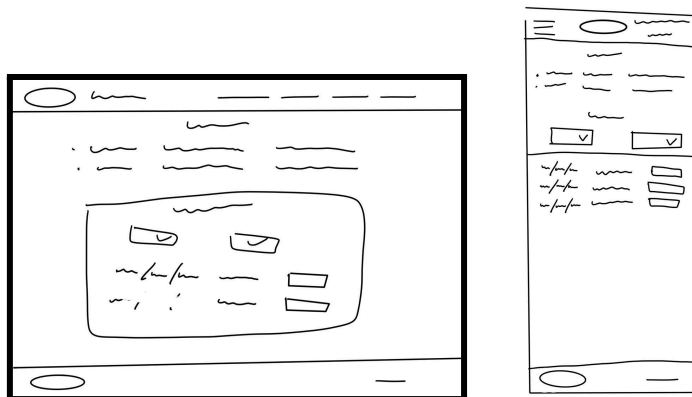


Área de Usuario: Página con lista y tabla

En el encabezado está el logo que lleva a la página de inicio, el tipo de usuario o su nombre y el menú que lleva a las diferentes páginas a las que tenga acceso este tipo de usuario.

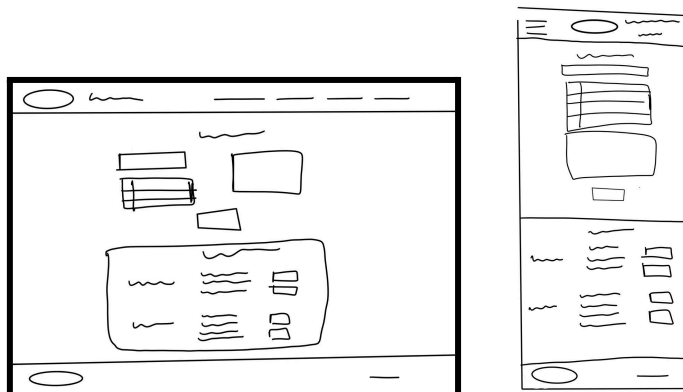
En el body está la lista, y la tabla, la cual incluye opciones para filtrar u ordenar los datos. En la derecha de la tabla están los botones de acción (cuando correspondan) de ‘borrar’, ‘restaurar contraseña’, ‘editar’, ‘crear una cita’ o ‘cancelar una cita’.

En el pie de página aparece de nuevo el logo y un enlace para cerrar sesión.



Área de Usuario: Página con formulario y tabla

Similar al anterior pero con un formulario.



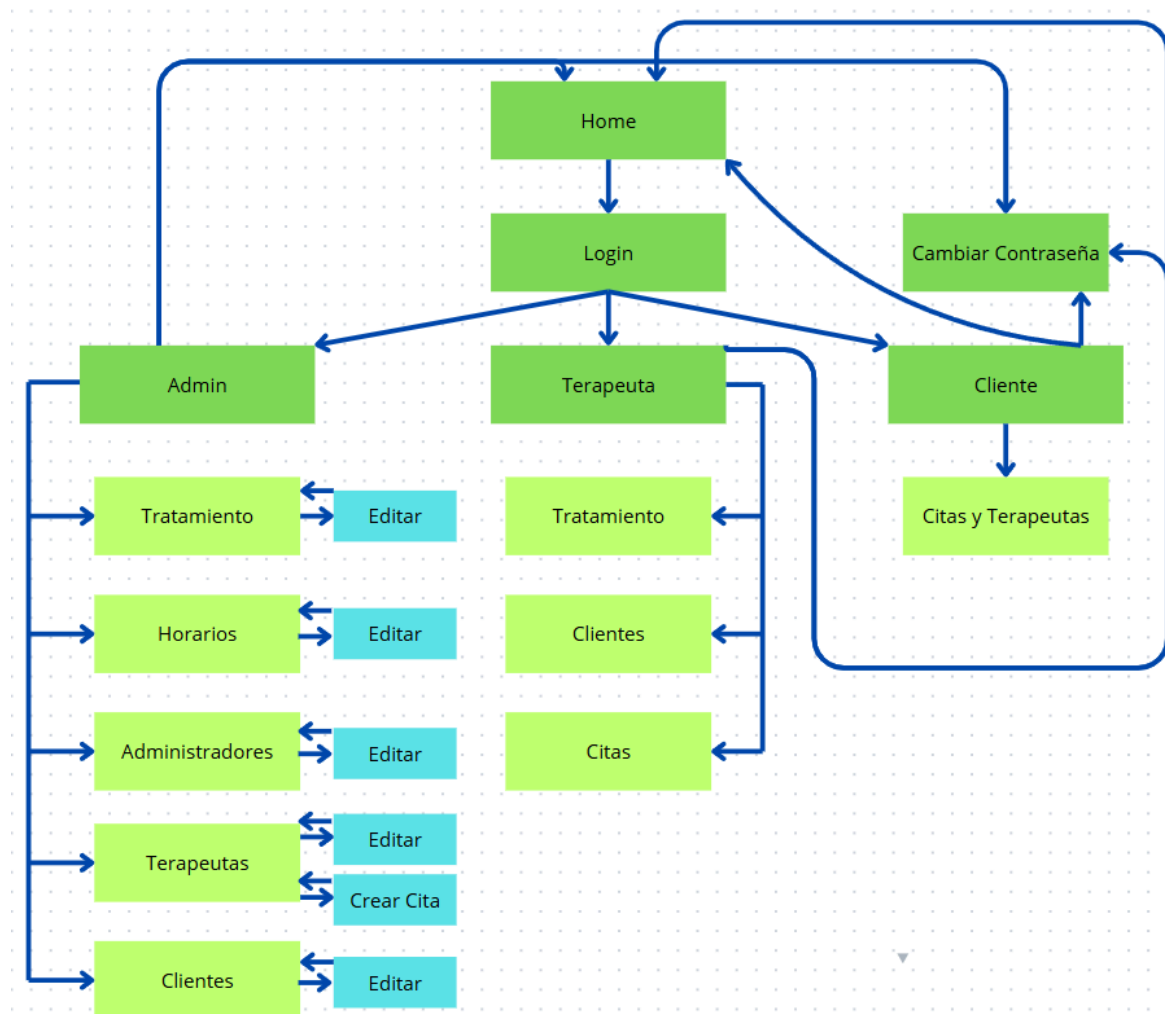
Criterios de accesibilidad

Siguiendo la guía de diseño web accesible de la iniciativa WAI se han realizado las siguientes cuestiones para mejorar la accesibilidad:

- Las imágenes contienen el atributo alt con texto descriptivo.
- El color del texto tiene un contraste alto con respecto al fondo.
- Una de las fuentes (Lexend) fue diseñada para facilitar la lectura para aquellas personas con dislexia.
- Las etiquetas html utilizadas son descriptivas, por ejemplo, para un botón no se usa la etiqueta <div> se usa <button>.
- Los elementos interactivos contienen transiciones cuando se pasa el ratón por encima para indicar la interactividad del mismo.
- Se ha realizado un diseño para pantallas grandes y otro para pantallas pequeñas.
- En el Home hay diferentes tipos de imágenes para adaptar mejor el diseño según necesidad.
- Se utilizan encabezados y secciones para agrupar el contenido relacionado.
- El estilo de la barra de navegación es consistente.

Criterios de usabilidad

En el siguiente mapa de navegación se puede observar lo intuitivo que resulta encontrar el contenido, facilitando el uso de la aplicación:



El diseño se ha buscado que sea “limpio” con espacios en blanco para oxigenar, los bloques de contenido relacionados dispuestos juntos con un lenguaje sencillo. Todo ello para facilitar que el usuario no necesite leerlo todo para saber rápidamente encontrar lo que le interese.

Las opciones de navegación siguen un orden lógico jerarquizado y los elementos que se repiten en distintas páginas están dispuestos en las mismas posiciones.

El diseño es responsive, haciendo más ameno el uso de la aplicación cuando cambia el tamaño de la ventana o cuando se usan diferentes dispositivos.

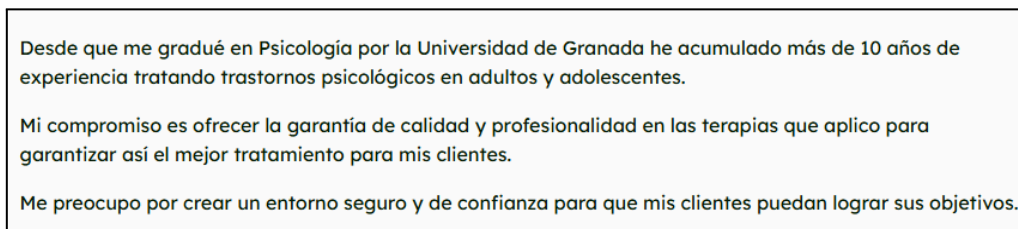
Tipografía

Se ha elegido usar dos tipografías: Comfortaa y Lexend.

Comfortaa es una tipografía ideal para textos de gran tamaño. Es redondeada, amplia y muy fácil de leer. Se ha utilizado para títulos y pequeños textos importantes como en el menú de navegación.



Lexend es una tipografía especialmente diseñada para personas con dislexia al reducir el estrés visual. Resulta perfecta para el texto del cuerpo y aquellos textos de menor tamaño o textos largos:



En conjunto ambas tipografías quedarían así:



Mapa de colores de proyecto en tres formatos (RGB, Hexadecimal, nombre junto con el color). Elementos donde se aplica.




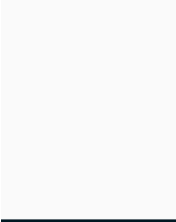

Al tratarse de una clínica de psicología se quería transmitir tranquilidad pero con colores con

algo de energía. No hay un color que se asocie a la salud mental por lo que entre los principales significados de los colores resultan interesantes los colores azul y verde ya que el azul se asocia a la tranquilidad y el verde a la esperanza.

Una paleta monocromática de azul podría resultar demasiado fría y una paleta monocromática verde podría evocar demasiado a la naturaleza y no asociarse al objetivo de la web.

También se descartó la idea de un esquema análogo de tonos de tres colores. El color amarillo podría haber sido un buen color para la web al transmitir energía y positividad, pero es un color que puede ser percibido negativamente por algunas personas con neurodivergencias, por lo que se descarta con el fin de que la web sea agradable para el mayor número de posibles clientes.

Por lo que finalmente se ha elegido la siguiente paleta de color:

	Azul-oscuro Sus códigos son #065f88 y rgb(6 95 136). Es el color secundario y acompaña con frecuencia al color principal. Se ha usado en títulos pequeños y algunos textos que se querían resaltar porque aportaba más contraste que el color verde. También se usa como color de fondo del pie de página y de botones.
	Azul-claro Sus códigos son #afe4f3 y rgb(175 228 243). Se ha usado como color de énfasis que acompaña al color verde y al azul oscuro por ser un color que aporta suficiente contraste. Se ha usado especialmente en botones.
	Verde Sus códigos son #31c5b7 y rgb(49 197 183). Es el color principal. Se usa en el logo, en la mayoría de <i>hover</i> y <i>border</i> y en general en los lugares a los que se ha querido llamar la atención del cliente. No se ha utilizado en los subtítulos por accesibilidad ya que implicaba menos contraste con el fondo.
	Blanco Sus códigos son #fafafa y rgb(250 250 250). Se ha utilizado como color de fondo. Se ha buscado un blanco roto para que la página no tuviera un intenso blanco absoluto.
	Negro Sus códigos son #031a03 y rgb(3 26 3). Se ha usado como color base de texto.

Dispositivos/vistas para las que se ha diseñado el proyecto.

Como se aprecia en el apartado de Sketches se han pensado los diferentes diseños para pantallas pequeñas y para pantallas grandes. Se ha utilizado SASS y Bootstrap. En pantallas medianas como las tablets los componentes de bootstrap se adaptan correctamente y el resto de elementos se adaptan por las unidades en ‘em’ o porcentaje.

Software utilizado.

Puesto que en el apartado ‘Justificación de la tecnología empleada’ ya comenté los aspectos básicos de Symfony, Doctrine, Twig, MySQL, Asset Mapper, JavaScript, jQuery, Sass y Bootstrap; voy a comentar otras herramientas utilizadas para el desarrollo de la aplicación cuyo papel es facilitar la fase del desarrollo.

Composer

Composer es una herramienta para la gestión de dependencias en PHP. Permite declarar las bibliotecas de las que depende el proyecto y se encarga de gestionarlas (instalarlas/actualizarlas). Por ello uno de los primeros comandos que se deben ejecutar cuando se clona el código de este proyecto desde GitHub es composer install.

Visual Studio Code

Realmente no es un entorno de desarrollo (IDE) es un editor de texto pero altamente configurable y potenciado gracias a las extensiones. Permite integración con GitHub.

Symfony CLI

Un servidor de desarrollo integrado en Symfony. Es rápido y tiene sus propios comandos que facilitan tareas de desarrollo.

XAMPP

Contiene PHP, Apache y MySQL por lo que resulta útil como herramienta de desarrollo para estudiantes, pero es más lento que otras alternativas. En mi caso ha sido útil salvo por la parte de servidor de desarrollo ya que al poco opté por Symfony CLI por sus mejores tiempos.

GitHub

Para el control de versiones. En mi caso he trabajado en una única rama, pero aún en desarrollo en solitario es especialmente útil para tener un control de los cambios realizados en el código.

Mejoras posibles y aportaciones.

No se ha implementado un mailer, lo que hubiera permitido securizar aún más las contraseñas provisionales tras la creación de un nuevo usuario.

Para una versión 2.0:

- Se implementaría un sistema para filtrar y ordenar dinámicamente los datos de las tablas.
- Ampliar el número de páginas que implementan AJAX.
- Una interfaz de calendario para que clientes y profesionales.
- Un sistema de borrado de las citas más antiguas.

En cuanto a la organización del código: los archivos de Controller/ se distribuyen las funciones por entidad afectada y por el rol del usuario que las ejecuta, eso puede ser confuso para aquellas entidades que no heredan de *User* ya que hay dos posibles ubicaciones para sus funciones. Esta organización es mejorable.

Bibliografía.

- Documentación de Symfony: <https://symfony.com/doc/current/index.html>
- Documentación de Doctrine: <https://www.doctrine-project.org/projects/doctrine-orm/en/3.3/index.html>
- Documentación de Twig: <https://twig.symfony.com/doc/3.x/index.html>
- Para consulta de métodos frontend: <https://developer.mozilla.org/en-US/>
- Documentación de Bootstrap: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- Documentación de jQuery: <https://api.jquery.com/>
- Fuente de imágenes svg: <https://www.svgrepo.com/>
- Guía de accesibilidad de la iniciativa WAI: <https://www.w3.org/WAI/tips/designing/>