Programación JAVA Web

Recuerdamelón

Bárbara Flores Aranda
José Fernández Sánchez
Juan Carrasco Guerrero
Juan García Moreno

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial

AGRADECIMIENTOS

Agradecemos primero la iniciativa de formación financiada con Fondos Europeos para jóvenes inscritos a Garantía Juvenil en colaboración con entidades de prestigio como la Escuela de Organización Industrial y modernas plataformas de desarrollo profesional como Generation Spain. Agradecemos el acompañamiento y tutoría de docentes de reconocida y demostrada profesionalidad y disposición. Sin ellos no hubiera sido posible alcanzar los conocimientos que a día de hoy completan nuestras metas laborales. También a todos los compañeros durante estos meses de aprendizaje. Gracias a nuestra colaboración y simpatía estos proyectos han sido posibles.

1.- ABSTRACT

Son muchas las aplicaciones web que acuden actualmente al framework de Spring para el desarrollo de su arquitectura. Entre sus principales ventajas nos encontramos con su capacidad de integrar tecnologías satélite como objetos POJO, Enterprise Java Beans, Java Persistance API e Hibernate (y por consiguiente JDBC), etc. Además de dichas virtudes, su arquitectura de Modelo Vista Controlador favorece la redacción de código limpio y claro facilitando también enormemente su mantenimiento. Todo esto incluyendo aspectos tan transversales como son la seguridad, diseño html, servicios REST y otros incluidos como frameworks adicionales a cualquier proyecto Spring. En definitiva, simplifica y dinamiza el desarrollo de las mismas sin perjuicio de los estándares de construcción fundamentales.

Con la ayuda de las herramientas propuestas hemos desarrollado una aplicación web que soluciona una necesidad básica e inherente a todo aquel que desarrolle una actividad: La gestión de tareas. El objetivo inicial era digitalizar la funcionalidad de la agenda personal. El resultado final añade un perfil para empresas, comunidades de usuarios, servicios de mensajería e incluso una sala de chat.

2.- INTRODUCCIÓN

La gestión de tareas ha sido una constante en el desarrollo de cualquier actividad humana. Está presente en los más diversos ámbitos de la vida profesional y personal. Generalmente resulta una labor tediosa y requiere de mucha precisión y memoria. Tiene su nacimiento en las primeras técnicas de registro de información y alcanza su óptimo a día de hoy con el desarrollo de las nuevas tecnologías informáticas; con especial repercusión tras el desarrollo de las tecnologías móviles.

Recuerdamelón es una útil herramienta web para facilitar el desarrollo de esta actividad. El nombre de la aplicación pretende combatir los formados prejuicios respecto a la gestión de tareas ofreciéndose como una solución refrescante, amigable y sencilla. En esta edición presentamos los resultados del proyecto como trabajo final al curso “Programación JAVA Web” impartido por la Escuela de Negocios Industrial en colaboración con Generation Spain.

2.1.- Objetivos

1.- Diseño de una aplicación web en lenguaje JAVA

2.- Construcción en el framework de Spring optando por soluciones como:

- 2.1- Uso del modelo vista-controlador
- 2.2- Uso de Spring Security
- 2.3- JPA e Hibernate para el desarrollo del modelo ORM

3.- Diseño web HTML&CSS a través de la herramienta thymeleaf

4.- Base de datos relacional con MySQL integrada en el modelo ORM de Spring

5.- Uso de funcionalidades en lenguaje JavaScript en el frontend de la aplicación.

2.2.- Enfoque Metodológico

En el desarrollo de esta aplicación se ha intentado simular un entorno de trabajo real incluyendo una fase de diseño de la aplicación, reuniones periódicas entre los miembros del equipo y uso de repositorio web de código (Git) para su producción mediante ramales de actividad. Para la integración posterior de ramales y comprobación de su ejecutabilidad hemos hecho uso de una herramienta muy práctica en este sentido conocida como CI con Maven (“ workflow-test”).

2.3.- Resultado

El resultado del proyecto se resume en los siguientes archivos:

- Código fuente de la aplicación. Tanto en fichero .war como enlace al repositorio web.
- Script SQL y diagrama relacional de la base de datos.
- Memoría técnica
- Presentación .ppt

3.- DISEÑO

3.1.- Diseño estético

3.1.1.- El nombre

Es el resultado de un brainstorming de posibilidades. Finalmente nos decantamos por Recuerdamelón por cumplir la pretérita intención de captar de manera amena al usuario. Adicionalmente el juego de palabras pensamos que facilita la socialización y difusión de la aplicación entre usuarios.

3.1.2.- El logo

Elegido también democráticamente por los integrantes del equipo. Refleja visualmente la identidad de nuestra aplicación. Contiene elementos propios de las aplicaciones de comunicación como el bocadillo utilizado por aplicaciones de mensajería tan conocidas como Whattsap o ChatMessenger. Esta simbología ayuda a identificar la aplicación con un modelo de relaciones entre usuarios a través de comunidades de actividad con tareas compartidas y servicios de mensajería.

3.1.3.- Estilo

El color verde *#88ad58* hexadecimal es el color predominante en la aplicación. Verde en asociación con Spring, tecnologías de uso público gratuito y por supuesto, el color del melón de piel de sapo. Las fuentes elegidas son “Quicksand” y “sans-serif” por su volúmenes redondeados y fácil lectura incluso cuando redimensionamos su tamaño para adaptarlo a distintos dispositivos.

3.1.4.- Diagrama de flujo

Para el diagrama de flujo entre las diversas pantallas que componen la aplicación así como el contenido de cada una de las mismas hicimos uso de la herramienta diagram (https://www.diagrams.net/). Esta fase nos ocupó las primeras semanas y sirvió de referencia en lo posterior para diseñar las páginas de navegación.

3.2.- Diseño técnico

3.2.1.- Modelo Objeto-Relación (ORM)

Comenzamos el diseño técnico desde su base, la base de datos. Identificamos las entidades necesarias para su ejecución y fuimos añadiendo algunas en el transcurso del desarrollo del negocio. Las entidades al final de la versión actual son:

1.- User: entidad primogénita de cualquier aplicación web que contiene los atributos propios y necesarios de cada usuario (nombre de usuario, contraseña y en nuestro caso también e-mail). Atributos de identificación (nickname, apellidos, fecha de nacimiento, nacionalidad y avatar como imagen de perfil). Además introducimos atributos boolean “active” y “business” para marginar usuarios según un perfil de actividad a configurar por los administradores de la aplicación. El perfil de usuario-empresa aporta funcionalidades discretamente relacionadas con este tipo de usuarios. Como consecuencia de esta última tipología de usuarios añadimos los atributos también opcionales de NIF, equipo y avatar de empresa.

2.- Task: La materia prima de nuestra aplicación. El objeto principal a crear, editar, guardar, eliminar y compartir por nuestros usuarios. Sus atributos obligatorios son el título de la tarea, las fechas de principio y fin y un campo tipo boolean “delete” que configuramos a valor “true” cuando la fecha de finalización de la tarea se ha cumplido. Otros atributos complementarios son la descripción de la tarea y direcciones web asociadas a la tarea.

3.- UserRole: Complementa a la entidad usuario dotando de atributos de rol administrador, usuario normal y usuario empresa. De este modo otorgamos acceso privilegiado a determinados usuarios con el rol de administrador y diferenciamos los ejecutables según el tipo de usuario empresa o individuo. Para permitir la interpretación de los mismos por nuestras clases de Spring Security (tanto para autentificación como para evaluar expresiones), los roles son precedidos por el prefijo “ROLE_”.

4.- TaskType: Entidad que permite a los usuarios clasificar sus tareas por categorías a través del atributo nombre de la categoría. Como atributo opcional se le puede añadir una imagen a la categoría.

5.- Community: Entidad que da nombre a la comunión de usuarios en un grupo de actividad. Las comunidades tienen un atributo obligatorio llamado nombre de la comunidad y otro entero donde guardamos el id del usuario administrador de la comunidad. Esta misma entidad es asimilada para el usuario con rol de empresa en su uso de equipos de trabajadores.

6.- Mensajes: Aquí guardamos los mensajes que intercambian los usuarios que pertenecen a una comunidad. Tiene como atributo obligatorio el mensaje. Añadimos atributos tipo *String* fecha transformado a tipo fecha *ZonedDateTime*. Además incluimos título o asunto, nombre del usuario emisor y nombre de la comunidad relacionada. Una serie de atributos tipos boolean nos permiten clasificar los mensajes en la carpeta de mensajes enviados, guardados en borradores, eliminados, recibidos e incluso invitaciones a participar en

comunidades y su posterior aceptación o rechazo a la participación. La entidad Mensajes ofrece el complemento necesario a la Comunidad para su mejor funcionalidad.

7.- ChatMessage: Esta entidad proporciona el soporte necesario al servicio de WebSocket en ejecución de la Sala de Chat grupal. Está íntimamente relacionada con el programa JavaScript a través de sus funciones “sendMessage” y “onMessageRecieved”. Almacenan durante la ejecución del servicio el nombre del usuario que ingresa, el contenido de los mensajes que postea y el tipo de conexión que establece (entrada o salida del chat). La información solo permanece a nivel de servicio durante la ejecución del socket, sin llegar a ingresar en la base de datos. Ofrecemos así un servicio de chat grupal efímero con acceso a todos los usuarios de la aplicación con la idea de establecer un canal de comunicación directo con los administradores de la aplicación (siempre conectados al chat).

Establecidas las entidades partícipes las relacionamos del siguiente modo:

Relaciones uno a muchos(@ManyToOne):

Task - TaskType: El tipo de tarea comprende muchas tareas pero una tarea solo puede ser de un tipo

Relaciones muchos a muchos(@ManyToMany):
User - Task: Comprendiendo el concepto de compartir tareas.
User - UserRole: Existen usuarios que además tienen el rol administrador.
User - Community: Comunidades de muchos usuarios y usuarios participantes de muchas comunidades
User - Mensajes: Usuarios mandan múltiples mensajes y estos pueden ser recibidos por varios usuarios.

Terminadas las relaciones y entidades probamos su implementación en H2 con éxito. Seguidamente transitamos la base de datos a MySQL a través de su aplicación de escritorio Workbench en su versión 8.0. Generamos el diagrama añadiendo los atributos y sentando sus valores por defecto, las claves foráneas derivadas de las relaciones entre entidades y los identificadores como generados por autoincremento. Una vez dibujado lo añadimos al servidor directorio de nuestra base de datos local mediante ingeniería de avance generando así el script. Dicho script es el que integramos en nuestro proyecto mediante el fichero data.sql a nivel de la carpeta “resources” de nuestro proyecto.

Para la conexión con la base de datos en nuestro servidor local creamos el archivo YML application.yml con las rutas de conexión.

3.2.2.- Código de la aplicación

3.2.2.1.- Java

Para conseguir eficientemente este fin localizamos las distintas capas de servicio de la aplicación en los paquetes siguientes:

3.2.2.1.1.- config:

Contiene clases de configuración de nuestro proyecto identificadas por Spring con el comentario @Configuration

a) Security Config: Clase elemental de Spring Security. Definición para el framework con @EnableWebSecurity. Extiende de WebSecurityConfigurerAdapter. Con @Import({CustomAuthorizationConfig.class}) damos permiso a la clase incluida para unirse a la configuración predeterminada por la librería. En @Order definimos las propiedades básicas de autorización.

   En esta clase hacemos uso de la configuración de autenticación CustomAuthenticationProvider, el administrador de acceso AccessDecisionManager y una clase que introduce la seguridad web basada en roles (getRoleHierarchy()) y permisos basado en expresiones (getPermissionEvaluator()) llamada CustomWebSecurityExpressionHandler.

   Hacemos sobrescritura (en adelante, @Override) del método de configuración heredado “configure”. Incluimos a lo anteriormente introducido la definición de nuestro formulario de login (ruta, ruta en caso de error de credenciales, url de confirmación de autenticación y permisos) así como el formulario de logout. Deshabilitamos el Cross Site Request Forgery (CSRF) para poder usar el servidor de base de datos H2 y definimos las rutas principales a nivel de login del siguiente modo:

   Acceso completo a las rutas de registro y recuperar contraseña.
   Acceso exclusivo a la ruta” /admin” para usuarios con “ROLE_ADMIN”.
   Acceso a la API de swagger para usuarios autentificados.

   Por último ocultamos (ignoramos) las rutas de acceso a archivos javascript, imágenes, css y fuentes durante la navegación.

b) CustomAuthorizationConfig: Como añadido a la configuración de seguridad predeterminada de Spring Security incluimos esta clase que establece un método decisor de acceso. Este método permite el uso de evaluaciones de tipo @PreAuthorize y @PostAuthorize en peticiones http (Spring Expression Lenguage). Dichas expresiones se resuelven en acceso permitido, no permitido y no incluido a determinada expresión. Poniendo un ejemplo sencillo, el acceso permitido provee de toda la información pedida al http, no permitido solo a parte de la información incluida y no incluido nos mantendría fuera del acceso a la petición. Esta

configuración añadida es muy útil en el momento de listar datos según se trate de un usuario con “ROLE_ADMIN” o no.

c) CustomPasswordEncoderConfig: Encripta la contraseña del usuario en nuestra base de datos según un algoritmo de fuerza 4.

d) EmailConfiguration: Esta clase desarrolla el método JavaMailSender usado en Spring para enviar mensajes electrónicos desde la aplicación. Aquí iniciamos la implementación de la interfaz de JavaMailSender especificando el protocolo simple de transferencia de correo (SMPT) para el envío de email al usuario en caso de que haya olvidado su contraseña y quiera acceder a la url de recuperar contraseña. De este modo añadimos seguridad extra al proceso de autentificación del usuario al tener cada uno asociado un email único en nuestra base de datos.El contenido del email esta definico en el paquete service, clase EmailService.java, método sendSimpleMessage de nuestra aplicación.

   Como posibilidad de desarrollo futuro de este servicio de la aplicación hemos incluido un bloque de código comentado en esta clase de configuración que permitiría el diseño html del email de petición (con su correspondiente método sendHtmlMessage en la clase EmailService.java).

e) LeafConfig: Clase necesaria para introducir Spring Security Dialect en nuestros templates de html en thymeleaf. Gracias a esta configuracion podemos añadir Spring Security Expressions del tipo “xmls:sec” en nuestro thymeleaf. Con esto autorizamos a mostrar contenido html a usuarios según qué rol tengan. Para ello usamos expresiones “sec:authorize=”hasRole(‘ROLE’)”

f) MvcConfig: La finalidad de esta configuración es localizar e integrar un nuevo directorio (user-photos) en nuestro proyecto. En el mismo archivamos las fotos de perfil o avatar que elija el usuario desde el entorno local de su sistema en el directorio de nuestra aplicación.

   Para completar esta función añadimos una clase en nuestro paquete util (FileUploadUtil.java) que guarda el archivo seleccionado por el usuario en la ruta establecida por nuestra clase de configuración.

   Un aspecto muy interesante de este proceso es la introducción de la librería MultiPart (documento de varios componentes) de spring framework para su consecución. Esta librería permite realizar la función de carga de archivos mediante un formulario html. Nuestro formulario de perfil de usuario introduce así la codificación enctype=”multipart” que permite que el archivo seleccionado sea transmitido en el método POST. En dicho método recibimos el archivo, lo guardamos en el directorio “user-photos” y guardamos la ruta de ubicación del mismo en nuestro atributo Avatar de tipo BLOB (objeto binario grande) destinado para dicho fin.

g) WebSocketConfig: En esta clase de configuración habilitamos un broker de mensajes web socket @EnableWebSocketMessageBroker e implementamos los métodos incluidos en la interfaz de WebSocketMessageBrokerConfigurer. Modificamos aquí nuestro socket servicio de chat haciendo @Override de los métodos heredados registerStompEndPoints y configureMessageBroker.

En el primero registramos el endpoint (ruta de conexión) en "/ws" y habilitamos la misma para cualquier tipo de navegador añadiéndole el método ".withSockJS()". El protocolo (normas y reglas) para el intercambio de mensajes elegido es STOMP (Simple Text Oriented Messaging).

En el segundo establecemos las rutas de mensajerías entre clientes como "/app/topic" desde la cual nuestro broker de mensajes emitirá los mismos a todos los usuarios conectados (suscritos) en dicha ruta en tiempo real.

3.2.2.1.2.- data:

a) Entity: Localizamos aquí nuestros objetos entidad añadiendo metainformación a través de anotaciones tipo @Entity, @Id, @Column, @GeneratedValue, @OneToMany, @ManyToMany y @JoinTable. Originalmente incluimos @Getter, @Setter y @AllArgsConstructor y @NoArgsConstructor como herramienta de la librería Lombok para ahorrar código; pero llegado el momento de introducir la librería de Mapstruct nos surgieron excepciones provocadas por conflicto entre ambas librerías y nos decidimos por conservar Mapstruct

.

b) Repository: Contiene las interfaces que extienden de la interfaz pública JpaRepository. Con dicha inyección heredamos los métodos básicos para nuestro CRUD de respuesta (Create, Read, Update & Delete). Además de los heredados de esta interfaz incluimos aquí de manera genérica algunos necesarios y definidos en la capa "service" para determinadas consultas:

User:
findByUsernameAndActiveTrue
findByEmailIgnoreCase
Community:
findByName

Community:
findByName

ConfirmationToken:
findByConfirmationToken

3.2.2.1.3.- dto:

Continuamos con la capa constituida por los Data Transfer Object. Estos objetos completan el transporte de los datos desde nuestra base de datos a las capas superiores y viceversa.

Contiene pues los objetos DTOs correspondientes a cada entidad con los atributos que el usuario necesita en su navegación por la aplicación. Además incluimos aquí un DTO auxiliar para uno de los procesos de la capa controller: “RecieversCreationDTO”. Aquí guardamos atributos-nombre de usuarios que se añaden a una comunidad para la creación del atributo Set<Mensajes> a enviar a cada usuario invitado a la comunidad.

3.2.2.1.4.- service:

a) Paquete mapper: Aquí incluimos nuestros elementos mapeadores. Ejecutan el volcado de la información entre DTOs y entidades. Llegados a esta capa nos decidimos por introducir las librerías de Mapstruct y así ahorrar mucho código como expresamos en líneas anteriores. Para este fin, incluida la dependencia, creamos una interfaz (en nuestro caso llamada “IEntityMapper”) que dibuja el esquema de los métodos propios de cada mapper. El esquema es sencillo, comprendidos atributos de entrada tipo entidad o DTO, se incluyen métodos que efectúan el volcado en los dos sentidos posibles: toDto y toEntity.

   Nuestras interfaces “serviceMapper” desarrollan dicho esquema para cada caso incluyendo metainformación como anotación en el encabezado del tipo @Mapper(componentModel =”Spring”) para darle cabida en nuestro framework y ahorrarnos la repetición y redundancia de código.

b) Paquete security: Parte integrante de nuestro Spring Security que autentifica al usuario cuando se logea en la aplicación. Acude al repositorio de usuario para ver si se encuentra en la base de datos, desencriptar la contraseña y comprobar qué rol tiene el mismo (de cara a futura autorización sobre funcionalidades). Si sus credenciales coinciden con las almacenadas le otorga un token de acceso e identificación.

c) Nuestras clases service con los métodos CRUD embebidos de nuestros repositorios más los aquí definidos para funcionalidades específicas de la navegación por nuestra web. Además incluimos un service para la edición del correo electrónico para restablecer contraseña y otro para la publicación del mensaje-evento que nombramos más adelante.

3.2.2.1.5.- event:

Incluimos aquí la clase que extiende de “ApplicationEvent”. Usamos este objeto como mensaje (MensajesDTO) contenido del evento “crear un nuevo mensaje” para nuestro listener atento a la ocurrencia del mencionado evento. En resumen, incluye la clase que genera un evento cuando se envía un mensaje.

3.2.2.1.6.- listener:

En nuestra aplicación tenemos tres listeners atentos a la generación de eventos:

a) LoginListener está atento al momento en el que un usuario se registra en la aplicación identificándose por su id.
b) WebSocketEventListener: Listener de nuestro servicio por sockets de chat. Registra el acceso o salida de usuarios del chat.

3.2.2.1.7.- utils:

Aquí contenemos tres “clases-herramienta” para limpiar código de nuestras clases principales. Tenemos “DateUtil” para transformar nuestras fechas a String y viceversa. “FileUploadUtil” para el guardado de imágenes de nuestro directorio local en la base de datos. “Invitation” para redactar los mensajes de invitación a la lista de usuarios invitados a una comunidad, “SetRoleToUser” para asignar el rol al usuario según el registro, “SentMensaje” para guardar una copia de cada mensaje enviado en la carpeta de mensajes enviador y por último “TaskHorario” para almacenar los datos del formulario de horario en un Map<String, String> basado en dos listas de dimensión variable con las fechas de inicio y fin de las actividades.

3.2.2.1.8.- web:

Nos encontramos ya ante la capa inmediatamente inferior a la actividad del usuario en nuestra web. Se comprende de:

a) Rest: Para su desarrollo nos servimos primero de la creación de una carpeta de @Controller de servicios REST. El objetivo es evaluar a través de la herramienta swagger la respuesta los métodos CRUD en sintonía con todas las capas inferiores hasta su comunicación con la base de datos.

b) Acces. Expression: Propios del Security Configuration. Incluye dos clases.

   i) CustomWebSecurityExpressionHandler: Extendemos de DefaultWebSecurityExpressionHandler y damos valor a atributos de la expresión a continuación. El evaluador de permisos, autenticación y jerarquía de roles.
   ii) CustomWebSecurityExpressionRoot: Comprueba que nuestro servidor local coincide con el demandado para otorgar autentificación a la petición de desplegar la aplicación.

c) Nuestras clases @Controller donde se suceden las peticiones tipo @GetMapping y @PostMapping en nuestro caso por parte de los usuarios. Completamos aquí el modelo MVC que permite una navegación fluida, responsive y con un backend limpio y de fácil mantenimiento.

3.2.2.1.9.- RecuerdaMelonApplication:

Ejecutable de nuestra aplicación.

3.2.2.2.- Resources

3.2.2.2.1- Templates

Todas nuestras páginas de navegación HTML. Quedan divididos en el conjunto de páginas a nivel de Login (registro, recuperar contraseña, registro de empresas y login) y a nivel de la aplicación una vez el usuario ha sido registrado y ha accedido. Esta división se hace presente con el uso de dos plantillas o layout de diseño. Como elemento común de estilo hemos hecho uso de las facilidades proporcionadas por Bootstrap en su versión 5.

A nivel de login tenemos el “layoutLog” que unifica el estilo para las páginas del Login (Fondo de pantalla, estilo de los formularios, llamadas a estilos propios y a algunos estilos recogidos de internet)

Las páginas de navegación una vez accedido tienen como elemento común la barra de navegacion superior a modo de menú con el logo. Esta es extendida a todas las páginas a traves del “layoutHome” que estructura todas con un fragmento “navbar” como cabecera y otro “content” donde incluimos el contenido propio de cada html.

Todas nuestras páginas son responsive y están normalizadas a los formatos de pantalla más comunes en la actualidad de modo que se adaptan a los cambios de resolución de las pantallas sin derivar en errores de visualización. Como ejemplo, nuestro menú de navegación o “navbar” se transforma en un desplegable tipo “hamburger” cuando reducimos las dimensiones de la pantalla.

3.2.2.2.2- Static

css

Estilos propios de la aplicación:

2.1.1.- loginStyle, recoverPasswordStyle y regidto son los archivos que aplican estilo a nivel de login

2.1.2.- menu-usuario, serviceSt, serviceStB, TBS, mensajes y mensajesB dan estilo a las páginas “internas” de la aplicación (Las copias serviceStB y mensajesB son usadas por los html del perfil business para añadir una tonalidad de azul distintiva #88a9db hexadecimal)

2.1.3.- normalize es un archivo descargado del repositorio github.com/necolas que contiene el código que estandariza nuestra aplicación a los distintos tipos de navegadores web más comunes.

2.1.4.- main es un css encargado de formatear la sala de chat grupal

img

Directorio de imágenes usadas en la aplicación (logo, iconos y otras imágenes)

js

Contiene los siguientes scripts de código en javascript:

2.3.1.- addUser es una función que va descubriendo campos a rellenar por el usuario en la página de añadir usuarios a una comunidad “users.html”

2.3.2.- main ejecuta todas las instrucciones necesarias para el funcionamiento del socket de la sala de chat grupal. Inicializa la conexión, añade a los usuarios a la misma y los categoriza en paralelo al @ChatController

2.3.3.- rePassword es la función encargada de señalar al usuario la coincidencia de sus campos contraseñas. Además habilita el botón submit del registro cuando el elemento password y su repetición repassword son coincidentes.

2.3.4.- serviceJs añade funcionalidades de css al despliegue de tareas para ajustarlas a contenedores que respondan a la interacción del usuario con las mismas (cambiando la clase de etiqueta y consiguientemente, el estilo).

2.3.5.- sidebarNav despliega una barra de navegación lateral dentro de los html del grupo mensajes donde se contiene el menú propio del mismo.

2.3.6.- TBS para generar un nuevo campo input en el formulario de horario de empresa con cada click en el botón asociado.

3.2.2.2.3.- application.yaml

Nuestro YML (application.yml) contiene las siguientes configuraciones:

3.1.- Pretty print para devolver textos de manera legible al usuario

3.2.- Conexión con el servidor de base de datos local mediante usuario y contraseña.

3.3.- Instrucciones a Hibernate (configuration DDL - Lenguaje de definición de datos) para que conserve por defecto todas los elementos existentes en nuestra base de datos. La

introducción de cambios la adapta mediante la creación de nuevos elementos. Además especificamos que nos muestre en los logs las consultas SQL que se ejecuten.

3.4.- Configuración de thymeleaf para que no guarde memoria en caché, lea lenguaje HTML y utilice UTF-8 como codificación general. Además especificamos las rutas de nuestras páginas html y elementos de estilo.

3.5.- Ya que se considera la aplicación presentada en fase de prueba, el script SQL lo definimos para que se reinicie con cada ejecución del programa.

3.6.- Configuración de ruta para los email de recuperar contraseña. Para proteger la identidad del email y contraseña del emisor de emails de recuperación introducimos su definición con la codificación "${MAIL_"variable": definición de la variable}"

3.7.- Por último definimos el puerto local de ejecución de nuestra aplicación.

3.2.2.2.4.- data.sql

Nuestro archivo data.sql contiene el script completo de nuestra base de datos en su última actualización a fecha de la presentación.

3.2.2.2.5.- messages.properties

En nuestro caso contiene la definición de variables de texto que incluimos en nuestras páginas html. Están divididas por bloques para facilitar su identificación.

3.2.2.3.- user-photos

Esta carpeta asociada a la ruta de nuestro proyecto contiene las fotos de nuestros usuarios en almacenamiento local.

4.- TECNOLOGÍAS

4.1.- Base de datos

Durante las primeras etapas del proyecto nos servimos de H2 Database por ser de modelado relacional, de acceso inmediato para aplicaciones integradas en lenguaje JAVA a través de ejecución directa de lenguaje SQL y por su simplicidad.

Cuando empezamos a construir las relaciones entre entidades contenedoras nos apoyamos del servidor de base de datos MySQL Workbench 8.0. porque nos ofrecía mayor abanico de interacción con su aplicación de escritorio. Hemos usado principalmente su vista de diagrama de la base de datos para el diseño de la misma y sus ingenierías de construcción para el volcado del diseño sobre un script. También resulta una herramienta muy accesible en la definición de las características de los atributos propios de cada entidad y la comprobación de cargas de datos al momento.

4.2.- Spring

El proyecto spring parte de su modelaje inicial en la herramienta web de “Spring Initializr” (https://start.spring.io/) que permite la creación de un directorio con los contenidos elegidos ejecutable por nuestro IDE.

En nuestro caso hemos estado trabajando con IntelliJ en aplicación del JDK 11 de Java. La versión de Spring ha sido la 2.6.7

Inyección de dependencias

Para el desarrollo de nuestra aplicación hemos inyectado en nuestro archivo XML pom las siguientes dependencias:

1.- Spring JPA: Facilita la implementación de repositorios basados en JPA conteniendo a su vez implementación de Hibernate para funcionar y herramientas básicas de JDBC para consultas nativas.

2.- Spring Security: Aporta servicios de seguridad al framework basados en J2EE. En nuestro caso los hemos usado para dar seguridad a los “HttpRequest” relacionados con los procesos de autentificación y autorización.

3.- Spring Thymeleaf: Biblioteca que implementa un motor de plantillas de visualización web. Puede utilizarse en otros entornos no web pero en nuestro caso lo hemos introducido para el fácil manejo de las consultas en HTML en relación con el modelo MVC.

4.- Spring framework: Framework elegido por sus ventajas a nivel de redacción (mediante Java Annotations) y por su arquitectura por capas que permite fácil escalabilidad y mantenimiento. Sin olvidar su cada vez más frecuente uso en el ámbito empresarial al que vamos dirigidos.

5.- H2 Database: Sistema de base de datos relacional. Lo utilizamos en las primeras etapas de desarrollo de la aplicación para ir testeando la arquitectura de datos por su sencillez, fácil integración con Spring y su ejecución en modo cliente-servidor.

6.- MySQL connector: Necesario para conectar con el script nuestra base de datos final en MySQL. Elegimos esta base de datos por su extendido uso y su naturaleza relacional. Complementa de mejor modo nuestros objetivos que H2 al ser una herramienta más desarrollada y frecuente en el entorno empresarial.

7.- Spring Doc: Realizada la arquitectura de la aplicación por capas hasta la de servicio decidimos optar por comprobar la efectividad de nuestros controladores antes de su construcción final mediante la herramienta “Swagger” aplicada a servicios Rest Controller. De este modo depuramos posibles ocurrencias de excepciones antes de definir nuestra capa controladora.

8.- Mapstruct: Esta librería nos ahorra gran cantidad de código al usar una interfaz para el mapeo de objetos sin necesidad de redactar los métodos encargados de esto. Su implementación en el framework de Spring es muy sencilla a través de su sistema de anotaciones con @Mapper.

9.- Lombok: Librería Java que como la anterior nos permite ahorrar código a través de anotaciones. Principalmente a nivel de la capa de entidades permitiéndonos definir todos los métodos getter, setter y constructores con simples comentarios. Sin embargo entraba en conflicto con Mapstruct y optamos por mantener Mapstruct en el desarrollo de nuestro proyecto.

10.- WebSocket: Protocolo de red que establece conexiones entre dos puntos finales de comunicación (sockets) permitiendo comunicación bidireccional a tiempo real. Lo usamos para el servicio de chat entre usuarios de la aplicación.

11.- Maven: Herramienta de Apache para estandarizar la configuración de nuestro proyecto en sintonía con Spring.

4.3.- M.V.C.

Con el objetivo de separar la lógica del negocio de los datos usamos el modelo vista controlador como patrón de arquitectura de software. Se compone de tres elementos. Un modelo con datos y los procesos involucrados en su transformación, una capa controlador que ejecuta dichos procesos según se sucedan las peticiones del usuario y una vista adaptada a las necesidades del usuario para que interactúe a través de peticiones.

En nuestro proyecto el modelo en su conjunto esta completamente integrado en un mismo archivo continente del proyecto. El modelo es el grueso de los directorios de datos, el controlador se desarrolla en el paquete Web del mismo y la vista se define en el directorio de resources.

4.4.- Security

Para el control de la seguridad de peticiones y navegación en nuestra aplicación nos servimos de Spring Security en la configuración del contexto y los extras de Spring Security 5 para autorización en la visualización de contenido. Estamos de este modo utilizando las librerías de seguridad sugeridas por nuestro framework para proteger nuestra aplicación contra ataques de fijación de sesiones, clickhacking o falsificación de solicitudes entre otros.

Su puesta en ejecución se formula en las clases CustomAuthorizationConfig.java, CustomPasswordEncoderConfig.java, LeafConfig.java, SecurityConfig.java y CustomUserAuthenticationProvider.java. Estas clases quedan explicadas en apartados anteriores.

Además de la implementación de Spring Security añadimos un codificador de la definición de variables de nuestro archivo YML para ocultar sus valores y establecemos un protocolo de restablecimiento de contraseña por medio del email único asociado al usuario.

5.- EJECUCIÓN DE LA APLICACIÓN

La ejecución de la aplicación tiene lugar en el entorno local del sistema. Debemos de completar los siguientes pasos:

- Guardar el directorio con el contenido de la aplicación en nuestro entorno local (puede descargarse en el siguiente enlace: https://github.com/Recuerdamelon/RECUERDAMELON)
- Crear un servidor local en MySQL (Se puede trasladar la base de datos a otro motor haciendo uso del script guardado en el archivo data.sql y cambiando en la configuración del archivo YAML el código referente a la conectividad con la base de datos).
- Crear un esquema en la base de datos con el nombre “recuerdamelon” (se puede cambiar el nombre reflejando el nuevo nombre en el codigo del archivo YAML referente a ese campo)
- Añadir las variables de conexión con el servidor local de base de datos en el archivo YAML.
- Abrir el directorio del proyecto en nuestro IDLE.
- Cargar los módulos de Maven en nuestro IDLE (se recomienda hacer una limpieza y reinstalación de los módulos antes de ejecutar la aplicación).
- Editar la configuración de ejecución de “springboot:run”
- Ejecutar y ver el resultado en nuestro puerto de ejecución local (puerto 8080 por defecto, puede configurarse en el archivo YML)

6.- IDEAS DE DESARROLLO

En la elaboración del proyecto “Recuerdamelón” se ha hecho presente una realidad anticipada; las aplicaciones web no tienen una versión final. Conforme avanzábamos en su desarrollo iban surgiendo ideas, posibilidades, mejoras. Junto al resultado actual del proyecto aportamos la identificación de futuras actualizaciones.

1.- La creación de una entidad calendario que nos permita visualizar las tarear en un calendario. Para conseguir este fin se nos planteaban dos opciones; integrar un servicio ya existente de calendarios (ejemplo: Google Calendar) a través de peticiones a su API o bien desarrollar el MVC que de soporte y visualización a nuestro calendario.

2.- Edición de una entidad notificaciones que funcionara como elemento transmisor, receptor y registro de nuestros eventos. Esta entidad simplificaría considerablemente el código de los listener, añadiría interactividad comunicacional entre usuarios en tiempo real y además registraría la actividad de los mismos.

3.- Un archivo registro de errores de ejecución y ocurrencia de excepciones nos permitiría hacer un diagnóstico de la evolución de la aplicación. Este elemento sería una herramienta de gran utilidad para el mantenimiento de la aplicación en línea.

4.- Un desarrollo más especializado del servicio de chat con sockets podría añadir servicios de chat entre usuarios por comunidad o incluso individualizado. Este elemento podría llegar incluso a sustituir por complementación al actual servicio de mensajería. Proyectemos un chat con posibilidad al usuario de archivarlos y clasificarlos según unas condiciones.

5.- La creacion de una entidad horarios con relación a tareas permitiría agrupar tarear en bloque de modo que el usuario solo visualize un elemento horario en su menu de tareas y al interactuar con el mismo se suceda el desplegable con fines de edición o consulta.

6.- El desarrollo de la aplicación con microservicios revolucionaría la arquitectura del mismo transformándola en codigo más versatil, flexible, facil de mantener y actualizado. La transición del código actualmente “monolítico” a la separación de procesos en comunicación por APIs propiciaría un escenario mucho más favorecedor a la mejora del código.