Componentes proyecto Spring Boot:

**Clase MAIN de aplicación (En wiku src.main.java.WikuApplication):**

* Se encarga de arrancar Spring boot como tal. Levanta las instancias de Spring, la conexión con la base de datos, el servidor tomcat embedido…

**src.main…**

* **Java**: en esta están todas las clases java del proyecto
* **Resources**: aquí se guardan todos los ficheros de configuración del proyecto. Esta carpeta en ocasiones está fuera del paquete src. Normalmente está fuera cuando se encuentra en un entorno de producción para que, en caso de necesidad, poder cambiar los ficheros de propiedades sin la necesidad de compilar de nuevo el código.
* **Webapp:** en esta se guardan todos los ficheros referentes al front.

**src.main.java:** en el caso de wiku tenemos varios paquetes que se organizan bajo el paquete com.camaras.wiku que corresponde al groupid y al artifactid del proyecto maven. Cada uno tiene su propia función:

* **Controller:** aquí se guardan todos los controladores de Spring MVC que se van a utilizar. Hay varias anotaciones a tener en cuenta:
  + **@Controller:** se coloca siempre delante de la definición de la clase. Indica a Spring que esa clase es un controlador y debe incluirla en el contexto.
  + **@RequestMapping(“/url”):** esta anotación se puede usar tanto en la propia definición de la clase como en la definición de los métodos. RequestMapping si no se lo especificas sirve tanto para el método GET como para el método POST. Generalmente se da un valor en el controlador para definir las llamadas al mismo.
    - **@GetMapping(“/url”):** funciona igual que RequestMapping pero únicamente para el método GET. Se usa en las definiciones de los métodos para indicar la url por la que se accede al método.
    - **@PostMapping(“/url”):** funciona igual que RequestMapping pero únicamente para el método POST. Se usa en las definiciones de los métodos para indicar la url por la que se accede al método.
  + **@ModelAttribute(“nombreObjeto”):** esta anotación se coloca delante de un parámetro de un método. Esta anotación indica que el objeto mapeado desde un formulario de la vista se va a recoger en el parámetro en cuestión.
  + **@PathVariable(“nombreVariable”):** esta anotación se coloca delante de un parámetro de un método. Cuando necesitamos mandar un valor en la misma ruta url, se añade dentro de las anotaciones @GetMapping, @PostMapping o @RequestMapping (ej: @GetMapping(“/vuelo/{id\_vuelo}”) donde id\_vuelo es el identificador de un vuelo que se le pasa al controlador por parámetro y se indica con llaves {}). Para recuperar ese valor de la url se coloca esta anotación delante del parámetro en cuestión.
  + **@RequestParam(“nombreParametro”):** esta anotación se coloca delante de un parámetro de un método. Sirve para recoger los valores de los parámetros que se envían en la url. Normalmente estos parámetros Spring los oculta en la url pero se envían igualmente. La forma de recoger estos valores es colocando esta anotación delante del parámetro del método en cuestión con el nombre del parámetro en la url. (EJ de parámetro de request: /destinos/buscar?**ciudad=xxxx**&**fecha=yyyy**).
* **Dao:** aquí se guardan las clases de tipo “dao “ o “repo” que son las encargadas de realizar las consultas a la base de datos. En el caso de JPA con Spring Data heredamos de la clase JpaRepository<Clase, ClaseID> que nos otorga métodos de persistencia y de acceso a datos básicos. Tenemos dos anotaciones importantes:
  + **@Repository:** le indica al contexto de Spring que se trata de una clase de repositorio de datos y que debe contemplarla en el contexto.
  + **@Query**("SELECT u FROM User u WHERE u.status = 1"): cuando en un método JPA no se puede especificar lo que se necesita utilizando la potencia de JPA usando el nombre del método se usa esta anotación para realizar la consulta a mano con HQL.
* **DTO:** es un paquete donde se almacenarán Java Beans que se van a utilizar para la transmisión de datos entre la vista y el controlador y el controlado y el servicio. DTO significa Data Transfer Object. Estas clases se modificarán según la necesidad de comunicación entre componentes.
* **Model:** en otras ocasiones este paquete de llama domain, por ejemplo. En él se incluyen todos los pojos JPA que se mapean a las tablas de base de datos. (ver apuntes JPA para las anotaciones)
* **Service:** en el se guardan todas las clases e interfaces de servicio. Por ello, en el paquete service se guardarán todas las interfaces y en el paquete “impl” de su interior todas las implementaciones de dichas interfaces. Sólo hay una anotación concreta:
  + **@Service:** le indica al contexto de Spring que se trata de una clase de servicio y que debe contemplarla en el contexto.
* **Utils:** en ella se van a incluir clases para englobar métodos reutilizables para cierto tratamiento de datos básicos o lógicas. Por ejempo, DateUtils, para el tratamiento con objetos de tipo Date. Las funciones que se suelen repetir a lo largo del código se tratan de “homogeneizar” y se incluyen en una clase útil. Esta clase simplemente sigue unas normas de buenas prácticas y en Spring se le añade la anotación:
  + **@Component:** señala a spring que se trata de una clase que debe incluir en el contexto como componente.

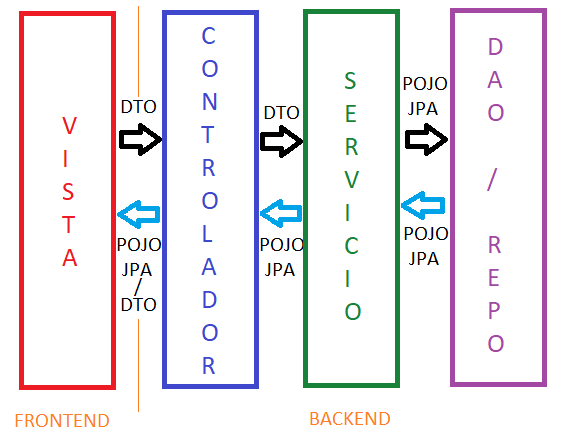
**src.main.resources:** aquí se incluyen todos los ficheros de configuración de la aplicación. Normalmente: persistence.xml, log4j.xml, ficheros properties, application.yml de spring boot…

**src.main.webaap:** aquí se incluyen todos los ficheros que conciernen al entorno web. Spring en sus versiones más altas elimina la necesidad del fichero web.xml. Todo lo que se incluye en la carpeta WEB-INF no es visible para el navegador o cliente. Por ello se suelen guardar los jsp en esta carpeta, para que para ver una vista sea obligatorio lanzar una petición por el controlador.

**MVC o Modelo Vista Controlador:** se trata de un modelo de arquitectura web muy extendido que busca separar la vista del modelo incluyendo el concepto de controlador. De esta manera la vista sólo sirve de input y output de datos con el controlador, eliminando al cliente o el navegador la posibilidad de acceder a la lógica o el acceso a datos de la aplicación. Posteriormente el controlador es el encargado de lanzar las lógicas para que se persistan los datos en la base de datos o recuperarlos para que se muestren en la vista.

Existen muchos modelos de arquitectura para MVC pero los más básicos incluyen los conceptos de controller, dao o repo y service. Lo que se busca con esto es que cada parte del backend cumpla con su cometido:

* **Controller:** se encarga de recibir las peticiones y de gestionar el manejo de datos con la vista y los servicios. Aquí se trata de evitar lo máximo posible la definición de lógicas en los controladores para abstraer a la vista de “lo que nuestra aplicación debe hacer” y así añadir seguridad al código. También es el encargado de realizar las llamadas y las peticiones a los servicios para la ejecución de la lógica y las peticiones de datos.
* **Service:** esta capa se encuentra entre el controlador y los dao o repos. Se encarga de ejecutar las lógicas de la aplicación y las transformaciones de datos para enviárselas al controlador, que posteriormente se lo enviará a la vista, o para enviárselas a los dao o repos y realicen la persistencia de datos. Normalmente los servicios tienen más acceso a las lógicas y a la base de datos, por ello se maneja siempre con una interfaz, de manera que cuando el controlador use las clases de servicios no tenga visibilidad de lo que hacen los métodos.
* **Dao o repo:** es la capa final del flujo y es la que se encarga de recuperar información de la base de datos o de persistir dicha información. Esta capa no debe llevar ninguna lógica.

****