La clase Bicicleta tiene un atributo de tipoEstacion, pero la estación con todos sus campos se guarda en mongodb, por lo que el atributo estacion en Bicicleta debería guardar el id de la estación, y tal como está definido, no se va a guardar bien en MySQL.

No definís la asociación entre Bicicleta e Incidencia.

De la misma forma, Estacion tiene una colección de tipo Bicicleta, no es necesario duplicar de esa forma, debería ser una colección de ids de bicicleta, de todas formas, luego no la guardáis en MongoDb, con lo que os sirve de poco.

Mejor etiquetar con @Lob los campos que pueden ser textos largos.

Seguís sin usar el patrón repositorio tal y como se ha visto en la asignatura.

Los servicios deberían lanzar sus propias excepciones.

En MongoDB podríais haber gestionado Estacion e Historico con una colección codificada, de manera que podéis manejar dichas entidades con la colección, en lugar de tipos Document.

Cuando dais de alta la bicicleta, no creáis histórico y supuestamente se estaciona, pero al estacionarla la añadís a la colección de la estación, que no se persiste, y no actualizáis el campo estación de la bici, que de todas formas se guarda mal. Luego, al retirar la bici ya ignoráis la colección de bicis que hay en estación.

Al crear/modificar las entidades estaría bien comprobar los campos para que no sean vacíos.

Habría que comprobar que la bicicleta no está ya estacionada a la hora de estacionarla.

No se debería aparcar una bici en una estación sin espacio.

En lugar de recuperar todos los históricos de la bicicleta (que pueden llegar a ser muchos) e iterar sobre ellos, se podría filtar en la consulta mongodb por aquellos que no tengan fecha de fin.

Para encontrar las bicis estacionadas cerca de una localización dada usáis el operador near pero no veo que hayáis creado un índice de geolocalización sobre el campo ubicacion con lo que no va a funcionar bien. Tenéis el método que crea el índice, pero jamás lo ejecutáis.

Al dar de baja una bicicleta hay que retirarla también.++++++++++

La función que devuelve las estaciones ordenadas por sitios turísticos se refiere a los sitios turísticos ya asignados a la estación (que por cierto no guardáis en mongodb), por lo que no hay que volver a llamar a geonames.

La fecha de baja era más para cuando se da de baja definitiva la bicicleta, pero ok.

No se debería poder crear una incidencia a una bicicleta que ya tiene incidencias abiertas.

Recuperar incidencias abiertas podría ser una consulta JPQL.

El método ServicioIncidencias.gestionarIncidenciaes demasiado genérico, hay que comprobar que una incidencia está en el estado adecuado para cambiarla a otro estado, si se asigna hay que retirarla de la estación, hay que darla de baja definitiva si se resuelve a no reparada, etc. No estáis implementando toda la lógica que se pide para ese método.

La conexión a mongodb debería gestionarse desde el repositorio, además de que tantos los servicios como los repositorios deberían crearse por medio de factorías como se puede ver en los proyectos vistos en clase.

En clase usamos la clase EntityManagerHelperpara gestionar la creación de EntityManager. Sin embargo vosotros usáis la anotación @PersistenceContext con la que le contenedor va a inyectar un entitymanager cuando sea necesario, pero ahora mismo la aplicación es Java SE porque no hay contenedor, y jetty, que se usa en el entregable 3, no es un contenedor JEE completo, con lo que no podemos usar dicha anotación y por eso no la usamos.