1. En la entrega de servicios tratar de solucionar el siguiente problema con las categorías: En el caso de que se añada un nuevo idioma al sistema, inicialmente todas las categorías tienen ese nuevo idioma a nulo, lo que puede dar error.
2. Restricción de patrón de email en servicios.
3. Un mensaje solo debe eliminarse de la base de datos cuando no esté referenciado en ninguna carpeta de ningún actor.
4. ¿Un handyWorker podría solicitar más de una vez una tarea? NO HAY CLASE ASOCIACIÓN. Podría interpretarse como entidad ya que puede cambiar el estado a REJECTED y volverlo a solicitar, cambiando así el estado y por lo tanto en ese caso sería una entidad. Pero esto depende de la interpretación de los requisitos.
5. Tras finalizar cada uno de los métodos anotados con @Test, se hace internamente y de manera automática un rollback en la base de datos. Tener esto en cuenta para los tests: Dentro de una misma clase de test, cuando acaba un método test y pasa al siguiente, en este último método de test la base de datos está como si estuviera recién creada, sin ningún cambio.
6. Si recuperamos un objeto de la base de datos y le aplicamos un “setAtributo”, automáticamente este objeto es actualizado en la base de datos, aunque no hayamos hecho ningún “save”. Esto no pasa cuando creamos un objeto nuevo y, antes de hacer “save”, inicializamos sus atributos con “setAtributo”.
7. En el caso de las composiciones como Curriculum – NoSeQueRecord, podemos crear el curriculum y todos sus record y hacer “save” solo de curriculum. Cuando se hace el save de curriculum también se hace de todos sus récords. También pasa lo mismo al borrar un curriculum, si se elimina un curriculum se eliminan automáticamente todos sus récords.
8. La etiqueta @Transactional que se pone en los servicios es la de la librería javax.transaction.Transactional.
9. La etiqueta @Transactional que se pone al crear los test es la de la librería org.springframework.transaction.annotation.Transactional.
10. Consultas dashboard que devuelven las 3 primeras entradas: Usar Pageable en el tipo de variable que devuelve
11. Posibles causas del error “Load Application Context”:
    1. Los métodos constructores de los servicios deben ser públicos.
    2. Alguna query de algún repositorio está mal construida.
12. Solo se pueden actualizar las Applications cuyo status sea “PENDING”.
13. Si un Application es actualizado a “ACCEPTED”, creditCard debe ser distinto de null. Esta regla de negocio se puede poner de forma simplificada con el siguiente assert: “Assert.isTrue(status != "ACCEPTED" || application.creditCard != null)”.
14. Cuando un Application es aceptado, el resto de Application del mismo FixUpTask deben de rechazarse de forma automática.
15. No se pueden borrar las FixUpTasks que tengan al menos un Application.
16. Intentar programar el tema de banear cuenta con el atributo UserAccount::isAccountActivated.
17. Revisar la visibilidad de todos los métodos:
    1. Los métodos que vamos a usar directamente en los controladores: public.
    2. Los métodos que vayamos a reutilizar entre distintos servicios pero que no se usen directamente en controladores: protected.
    3. El resto private.
18. **No** ejecutar nunca todos los tests de golpe haciendo click derecho en la carpeta services > Run as Junit Test. Si se hace de esta forma algunos tests que están bien pueden dar error y algunos tests que están mal pueden dar como correctos.
19. Para hacer los mockups, si hay una vista en la que puedan acceder todos los actores del sistema, la url que le llega a la vista sería borrando el rol de la url, por ejemplo: **announcement/list.do,**  en vez de poner **announcement/administrator/list.do.**