GRUPO DE TRABAJO G2-1

aSIGNATURA DISEÑO Y PRUEBAS 2, TERCER CURSO DEL GRADO INGENIERÍA INFORMÁTICA, INGENIERÍA DEL SOFTWARE  UNIVERSIDAD DE SEVILLA

ACME-VOLAR

INFORME RELATIVO A PRUEBAS Y SEGUIMIENTO DEL PROYECTO

# ÍNDICE

[ÍNDICE 1](#_Toc36493895)

[INTRODUCCIÓN 2](#_Toc36493896)

[ADVERSIDADES E INCONVENIENTES 3](#_Toc36493897)

[ESTRATEGIA DE PRUEBAS Y RESULTADOS 5](#_Toc36493898)

# INTRODUCCIÓN

El presente documento pertenece al proyecto de la asignatura desarrollado a cabo por el equipo de desarrollo compuesto por el grupo G2-1 de la asignatura, en el marco de desarrollo del proyecto “Acme-Volar” perteneciente a la parte práctica de la asignatura Diseño y Pruebas 2 perteneciente al grado de Ingeniería Informática, Ingeniería del Software cursado en la Universidad de Sevilla.

Con este documento, desde el grupo de trabajo, pretendemos dejar constancia, en primera instancia, de los diferentes contratiempos y adversidades a los que se ha enfrentado el equipo a la hora de llevar a cabo la implementación de la aplicación y sus pruebas.

De manera adicional, pretendemos mostrar y dejar constancia de la estrategia a la hora de implementar y realizar las pruebas y resultados de estas, para que sea considerada a la hora de analizar sus resultados a posteriori.

# ADVERSIDADES E INCONVENIENTES

Durante la elaboración de las tareas asociadas al proyecto, los diferentes miembros del grupo se enfrentaron a problemas o adversidades durante la implementación de la aplicación y sus pruebas, entre las que destacamos las que mostramos a continuación.

En lo relativo a los problemas que aparecieron durante la implementación de tareas relacionadas con la entidad flight: tuvimos problemas con el *Formatter* de *FlightStatusType*, debido a que la anotación *@Component* no se incluyó al implementar dicha clase. A su vez, tuvimos problemas con los *Formatters* de *Plane, Runway* y *RunwayType*, ya que necesitábamos reflejar dichas entidades en selectores y poder elegir un objeto satisfactoriamente para crear un Flight. Se optó por añadir una clase al controlador que insertara estos datos en el modelo. Además, la bidireccionalidad que poseía la entidad con otras entidades como *Plane, FlightStatusType, Airline* y *Runways*, nos obligó a hacer *debug* de todos los métodos, dado que había muchas dependencias insatisfechas. Con respecto a las restricciones de Flight, primero se hizo una serie de validaciones en el mismo controlador. Posteriormente decidimos implementar un validador, aunque el traspaso de validaciones a un contexto diferente dio problemas, aunque finalmente se logró implementar con éxito.

En lo relativo a la implementación de pruebas, relativas a servicio, el único inconveniente fue la captación de excepciones, debido a que no conocemos algunas excepciones. En lo relativo al controlador, en un principio, las anotaciones *@MockBean* y *@Autowired* no dejaban iniciar los *tests*, hasta que conseguimos disponerlas de la forma correcta. Otro problema añadido fueron los métodos "*setup*" que se ejecutaban al principio de cada prueba y por ello, tuvimos que hacer *debug* para ir conociendo los *mocks* que eran necesarios y su correspondiente devolución en las llamadas a los *given*(), en los que tuvimos que inicializar todo lo que nos debería devolver la base de datos. A su vez, el tipo Date de algunas fechas nos obligó a investigar como formar un *String* adecuado para usar en las pruebas de dichas propiedades. Finalmente usamos *Date.from(Instant.parse("yyyy-MM-ddT00:00:00.00Z")*. Posteriormente, intentamos implementar las pruebas del procesamiento de creación que devolvieran la vista de dicha creación, pero descubrimos que, al no persistir en la base de datos, solo podíamos esperar una redirección. Por otro lado, este error vino influido porque al hacer los *tests* de *update* sí que se necesitaba el id de la entidad. En definitiva, el *@Valid* del controlador causó muchos errores dado que no se encontraba los objetos necesarios en el *@BeforeEach* y no encontraba atributos en el modelo, puesto que no se llegaba al proceso de creación o actualización.

Por parte de la pareja de trabajo formada por Daniel y Javier, aseguran acerca del *Formatter* en *plane* no sabía que había que hacerlo y perdí muchísimo tiempo en implementarlo - Pruebas sobre el *AirportControllerTests* en el método de show que es positivo, ya que mi compañero Javi y yo no caímos en la cuenta de que se nos olvidó agregar la API sobre el propio *setup().* Acerca de errores más generales, no se tomaron correctamente las restricciones definidas desde un primer momento. Por otra parte, surgió la necesidad de solucionar errores en los seleccionables como por ejemplo en Flight cuando se fallaba algún atributo tardamos mucho tiempo en detectar dicho error. Por otra parte, las historias de usuario se tomaron en un principio superficialmente, cosa que deberíamos habernos tomado con más tranquilidad y haber aclarado con mayor profundidad los casos positivos y negativos, no que durante en el Sprint hemos ido añadiendo dichos casos conforme veíamos más correctos, ya que van surgiendo detalles que dan pie a una nueva interpretación.

La pareja formada por José Manuel y José tuvo problemas al trabajar con las horas de algunas propiedades. Como solución se optó por cambiar las horas que daban conflictos. Con respecto a las pruebas, la creación incorrecta en validación impidió obtener entidades en el modelo.

# ESTRATEGIA DE PRUEBAS Y RESULTADOS

A continuación, pasamos a exponer la estrategia que hemos llevado a cabo para implementar las pruebas destinadas a probar la aplicación para tener constancia de una manera detallada la metodología que se ha utilizado para llevarla a cabo y los resultados obtenidos mediante ella.

Como parte de la estrategia de pruebas de la aplicación en producción, se han llevado a cabo pruebas unitarias para todos los métodos en los controladores y servicios de la aplicación, tanto triviales como no triviales, probando casos positivos y negativos.

De esta manera, se han implementado dos pruebas en los casos positivos, una prueba unitaria y posteriormente una parametrizada que pruebe diferentes casos de tal manera que quede probado su correcto funcionamiento. Además, se han implementado casos de prueba negativos, en los que se prueban casos de datos no permitidos o que no cumplan con las reglas de negocio de la aplicación o con restricciones del sistema para probar que se cumplen y validan los campos correctamente, además de realizar pruebas acerca de la seguridad en lo que respecta a los roles de usuario que pueden ejecutar una acción u otra dentro del sistema.

Por otra parte, también se han realizado pruebas acerca de la corrección en el modelo de las restricciones implementadas nivel de base de datos que complementan a las de niveles superiores de servicio y controlador, incluyendo también pruebas de la API externa.

Con el fin de automatizar el proceso de las pruebas, también se ha utilizado la herramienta Travis CI, a la cual se le fue concedido el acceso y se ha integrado al repositorio para llevar a cabo la ejecución de todas las pruebas cada vez que un miembro del equipo actualiza una rama, devolviendo el resultado y una traza con los detalles de la ejecución de estas, pudiendo visualizar el resultado de manera intuitiva desde la interfaz de Git desde la que se gestiona el repositorio y ramas del proyecto.

De tal manera, tras el desarrollo completo de las pruebas, se ha obtenido un resultado de se han ejecutado, sin contar con aquellas implementadas previamente para la plantilla *PetClinic*, 6 pruebas a nivel de modelo en las que se verifica las restricciones a nivel de Base de Datos, 68 pruebas a nivel de servicio y finalmente, 146 pruebas a nivel de controlador; lo que hacen en total un conjunto de 220 pruebas unitarias y parametrizadas.