Entrega 06

Sistema Académico de Formación Empresarial - SAFE

Elaborado por:
Error 404

Integrantes:
Bermudez Guaqueta Tomas Alejandro
Cely Infante Daniel Alfonso
Gracia Pinto Daniel Alonso
Herrera Novoa David Alejandro

Docente:
Oscar Eduardo Alvarez Rodriguez
Ingeniería de Software I

UNIVERSIDAD NACIONAL DE COLOMBIA
Bogotá, 2025

Herramientas Utilizadas

Django (django.test.TestCase): Esta es la herramienta principal para las pruebas que interactúan con la base de datos.

- Es una versión especializada de unittest proporcionada por Django.
- **Por qué es crucial:** Automáticamente crea una base de datos de pruebas temporal (aislada de la base de datos de desarrollo postgres_db) por cada ejecución.
- Cada método de prueba (ej. test_usernameIsUnique) se ejecuta dentro de una transacción de base de datos, asegurando que cada prueba comience con una base de datos limpia y no sea afectada por las pruebas anteriores.

Python (unittest.TestCase): Has usado inteligentemente la biblioteca estándar de Python para tu clase TestPasswordValidator.

- **Por qué es una buena práctica:** Estas funciones de validación de contraseñas (como has_uppercase, has_digit, etc.) son lógica pura de Python. No necesitan conectarse a la base de datos.
- Al usar unittest.TestCase en lugar del de Django, estas pruebas se ejecutan más rápido porque no necesitan construir y destruir una base de datos de prueba.

get_user_model(): Esta es una función de utilidad de Django. Es la mejor práctica para referenciar al modelo de usuario (sea el estándar de Django o uno personalizado), haciendo el código de pruebas robusto y mantenible.

TestPasswordValidator

🚀 Funcionalidades que valida

Esta clase asegura que las reglas de negocio para una contraseña segura se apliquen correctamente. Cada prueba se enfoca en una regla atómica:

- **test_is_enough_length:** Confirma que la contraseña cumple con la longitud mínima requerida (8 caracteres).
- **test_has_uppercase:** Confirma que la contraseña contiene al menos una letra mayúscula.
- **test_has_lowercase:** Confirma que la contraseña contiene al menos una letra minúscula.
- **test_has_digit:** Confirma que la contraseña contiene al menos un número.
- **test_has_no_spaces:** Confirma que la contraseña no contiene espacios en blanco.
- **test_is_valid_password:** Es la prueba de integración que valida la función principal `is_valid_password`, combinando todas las reglas anteriores para dar un veredicto final.

🚧 Casos límite y de borde

Se cubren los siguientes casos:

- **Cadena vacía (""):** Se prueba la entrada más corta posible en `test_is_enough_length`.
- **Contraseñas "puras":** Se prueban contraseñas solo con minúsculas ("islowercase") o solo mayúsculas ("ISUPPERCASE") para asegurar que no den falsos positivos.
- **Violación de una sola regla:** Casos como "Haventdigit" prueban sistemáticamente cada regla.
- **Espacios en blanco:** El caso "have space blank" es un excelente caso de borde.
- **Casos de éxito:** Se incluyen "Password1" y "ValidPass1" para asegurar que el "camino feliz" funcione.

Tests QuestionUploadForm

🚀 Funcionalidades que valida

Esta clase asegura que el formulario de carga de exámenes (`QuestionUploadForm`) aplica correctamente todas sus reglas de validación. Cada método verifica una regla de negocio o campo requerido:

- **test_upload_valid_txt_file:** Confirma el "camino feliz", probando que el formulario es válido cuando todos los datos (Curso, dificultad y archivo .txt) son correctos.
- **test_reject_invalid_extension_file:** Verifica que el validador rechaza un archivo con extensión incorrecta (ej. `.png`) y confirma que el mensaje de error específico está presente.
- **test_form_is_invalid_if_no_file_is_sent:** Asegura que el campo `file` es obligatorio.
- **test_form_is_invalid_if_no_course_selected:** Asegura que el campo `course` es obligatorio.
- **test_form_is_invalid_if_no_difficulty_selected:** Asegura que el campo `difficulty` es obligatorio.

🚧 Casos límite y de borde

Se cubren los siguientes casos de validación:

- **Violación de Campo por Campo:** Las pruebas `test_form_is_invalid_...` aíslan perfectamente los campos requeridos, probando que el formulario falla si solo falta un campo obligatorio a la vez.
- **Validación de Extensión (Caja Negra):** El test `test_reject_invalid_extension_file` prueba el caso de borde de un usuario subiendo un tipo de archivo no permitido.

- **Validación de Mensaje de Error:** El mismo test anterior verifica que el texto específico del error ("Solo se permiten archivos...") esté presente en `form.errors['file']`, garantizando feedback claro al usuario.
- **Caso de Éxito:** Se incluye `test_upload_valid_txt_file` para asegurar que el formulario 100% correcto funcione.

TestUniqueUsername

🚀 Funcionalidades que valida

Esta prueba valida la restricción de la base de datos de que el campo `username` en el modelo `User` sea único. El flujo de la prueba es:

1. **Given:** Se crea un `usuario_unico`.
2. **When:** Se intenta crear un segundo usuario con el mismo `username`.
3. **Then:** Se verifica que el sistema lanza un `IntegrityError`.
4. **And:** Se verifica que el conteo de usuarios sigue siendo 1.
5. **Finally:** Se verifica el "camino feliz" asegurando que un `usuario_distinto` SÍ se puede crear.

🚧 Casos límite y de borde

- **Violación de Restricción UNIQUE:** Este es el caso límite principal. La prueba valida que el sistema maneja este error de BD de forma controlada (`assertRaises(IntegrityError)`).
- **Rollback de Transacción:** El uso de `transaction.atomic()` asegura que la transacción fallida se revierte correctamente sin corromper el estado de la prueba.
- **No Falsos Positivos:** Al probar que "usuario_distinto" sí funciona, se asegura que la restricción no esté rota y rechazando todas las entradas.

TestUniqueEmail

🚀 Funcionalidades que valida

Esta clase asegura que la lógica de verificación de existencia de email (`unique_email`) funcione correctamente:

- **test_unique_email_true:** Confirma que `unique_email` retorna `True` cuando el email consultado sí existe en la base de datos. Se crea un usuario en `setUpTestData` y se invoca la función con ese mismo email.

- **test_unique_email_false:** Confirma que `unique_email` retorna `False` cuando el email consultado no existe. Se usa un correo sintético que no corresponde a ningún registro.

🚧 Casos límite y de borde

Se cubren los siguientes casos:

- **Email existente:** Se verifica que un email exactamente igual al almacenado sea reconocido como existente (`True`).
- **Email inexistente:** Se prueba un correo no registrado, asegurando que la función no genere errores y retorne `False`.
- **Aislamiento de la lógica de negocio:** Se comprueba que `unique_email` depende únicamente del estado de la DB de pruebas y no de datos externos, favoreciendo la repetibilidad.

TestParseEvaluacion

🚀 Funcionalidades que valida

Esta clase valida la función `parse_evaluacion`, que interpreta un archivo de texto plano de evaluaciones.

- **test_valido:** Verifica que, dado un texto con formato correcto:
  - Se construya la lista de preguntas con la cantidad esperada.
  - Cada pregunta tenga el `id` y `texto` correctos.
  - Cada pregunta contenga el número correcto de opciones.
  - Cada opción tenga su `id`, `texto` y `es_correcta` con los valores esperados.
- **test_invalido:** Comprueba que `parse_evaluacion` lance un `ValueError` cuando el texto no respeta el formato. Los textos inválidos probados incluyen:
  - Una opción que aparece sin una pregunta previa.
  - Una pregunta que no tiene ninguna opción marcada como correcta.
  - Una línea de pregunta que no respeta la sintaxis (ej. sin el separador `|`).

🚧 Casos límite y de borde

Se consideran los siguientes casos:

- **Opción sin pregunta asociada:** Un archivo que empieza directamente con una línea `O:`. Se verifica que se rechace con `ValueError`.
- **Pregunta sin opción correcta:** Se define una pregunta donde todas las opciones tienen el flag `0` (incorrecta). El test confirma que se lanza una excepción.

- **Línea de pregunta mal formada:** Se incluye una línea `Q:` que no contiene el separador `|`. El test valida que este error de sintaxis se trate como entrada inválida.
- **Integridad de la estructura resultante:** Se comprueba que la estructura devuelta en el caso válido tenga la forma esperada (lista de diccionarios).

TestIsTxtFile

🚀 Funcionalidades que valida

Esta clase valida la función `is_txt_file`, encargada de comprobar que el archivo subido tenga la extensión `.txt`.

- **test_is_txt_file:** Verifica distintos escenarios de nombres de archivo:
  - Archivos con extensión `.txt` en minúsculas y mayúsculas (ej. `evaluacion.txt` y `evaluacion.TXT`) deben ser aceptados.
  - Archivos con otras extensiones (ej. `evaluacion.pdf`) deben ser rechazados.
  - Archivos sin extensión y valores que no son objetos de archivo también deben ser inválidos.

🚧 Casos límite y de borde

- **Extensión en mayúsculas:** Se incluye `evaluacion.TXT` para comprobar que la validación no dependa de la capitalización.
- **Archivos sin extensión:** Se prueba un nombre como `evaluacion` (sin `.txt`) para asegurarse de que no sea válido.
- **Entrada no archivo:** Se usa un valor como `"no_es_archivo"` para confirmar que la función maneja entradas inesperadas (sin atributo `.name`) retornando `False` en lugar de una excepción.

ChangeRoleTests (RF_3)

🚀 Funcionalidades que valida

El test case `ChangeRoleTests` verifica el servicio `change_role` (núcleo del RF_3). Las situaciones evaluadas son:

- **Cambio de rol exitoso:** El Analista TH puede cambiar el rol de un usuario. El servicio retorna `True`.

- **Validación del dominio de roles:** Intentar asignar un rol inexistente o `None` genera un `ValueError`, garantizando que solo se acepten roles definidos.
- **Registro de auditoría:** Se genera un `RoleChangeLog` incluso si el nuevo rol es igual al anterior, cumpliendo el requisito de trazabilidad.
- **Prohibición de auto-modificación:** Un Analista TH no puede cambiar su propio rol. El sistema lanza `PermissionDenied`.
- **Usuarios sin rol previo:** Un usuario sin rol inicial puede recibir uno válido, cubriendo escenarios de cuentas incompletas.
- **Restricción para roles no autorizados:** Cualquier usuario que no sea Analista TH recibe `PermissionDenied` al intentar cambiar un rol.
- **Múltiples cambios, múltiples logs:** Cada cambio genera una entrada independiente en `RoleChangeLog`, asegurando consistencia histórica.

🚧 Casos límite y de borde

Se consideran los siguientes casos:

- Asignación de roles inexistentes.
- Asignación de rol nulo.
- Auto-asignación bloqueada.
- Usuario sin rol previo.
- Cambios repetidos.
- Intento de gestión por usuarios sin permisos.

LearningPathAccessTests (RF_5)

🚀 Funcionalidades que valida

Evalúa la función `get_paths_for_user(user)`, que determina qué rutas puede ver un usuario según su rol.

- **Colaborador:**
  - Solo accede a las rutas en las que tiene una inscripción válida.
  - No recibe rutas que no le pertenecen.
  - No recibe duplicados si tiene inscripciones repetidas.
  - Recibe lista vacía si no tiene inscripciones.
- **Supervisor:**
  - Accede únicamente a las rutas donde al menos un miembro de su equipo tiene inscripción activa.
  - Si ningún miembro está inscrito, la lista es vacía.
- **Analista TH:**
  - Puede ver todas las rutas, independientemente de inscripciones.

🚧 Casos límite y de borde

- Colaborador sin inscripciones.
- Supervisor sin equipo.
- Rutas sin inscripciones activas.
- Inscripciones duplicadas.

GetCoursesForUserTests (RF_5)

🚀 Funcionalidades que valida

Evalúa el servicio `get_courses_for_user(user)`, que corresponde al acceso a cursos sin filtrar por ruta.

- **Analista TH:** Solo ve cursos en estado `ACTIVE`, nunca borradores, respetando que la administración no accede a contenidos incompletos.
- **Supervisor:** Ve todos los cursos donde al menos un miembro de su equipo esté inscrito. Recibe lista vacía si su equipo no tiene inscripciones.
- **Colaborador:** Accede exclusivamente a cursos en los que él mismo está inscrito. Si no tiene inscripciones, la lista es vacía.

🚧 Casos límite y de borde

(El documento original no detalla casos de borde para este test case)

GetCoursesInPathTests (RF_5)

🚀 Funcionalidades que valida

Evalúa la función `get_courses_in_learning_path_for_user(user, path)`, validando el comportamiento del RF_5 dentro de una ruta específica.

- **Analista TH:** Accede a todos los cursos activos vinculados a la ruta, independientemente de inscripciones.
- **Supervisor:** Solo ve cursos en los que algún miembro de su equipo tiene inscripción activa *y* el curso pertenece a la ruta.
- **Colaborador:** Debe cumplir dos condiciones: el curso pertenece a la ruta *y* él tiene una inscripción activa en ese curso. Si cualquiera falla, el curso no aparece.

🚧 Casos límite y de borde

(El documento original no detalla casos de borde para este test case)

GetContentsForUserTests (RF_5)

🚀 Funcionalidades que valida

Evalúa `get_contents_for_user_in_course(user, course)`, validando la unidad mínima: el contenido pedagógico.

- **Analista TH:** Ve todos los contenidos del curso en orden, sin restricciones.
- **Supervisor:** Accede a todos los contenidos si un miembro de su equipo está inscrito en el curso. De lo contrario, no ve nada.
- **Colaborador:** Avanza de forma progresiva.
    - Ve la secuencia de contenidos de forma lineal.
    - Al llegar al primer contenido obligatorio (`is_mandatory=True`), la progresión se detiene.
    - Los contenidos posteriores a ese punto no son visibles.

🚧 Casos límite y de borde

- Supervisores sin equipo inscrito.
- Colaborador que solo ve una parte de la secuencia (detenido por un contenido obligatorio).

MaterialTypeInferenceTests

🚀 Funcionalidades que valida

Valida la lógica de inferencia automática del tipo de un `Material` a partir de la extensión del archivo, usando el método `infer_type_from_file()`.

- **test_infiere_tipo_extensiones_soportadas:** Verifica que para extensiones soportadas (.pdf, .jpg, .mp4, .mp3, .txt), se asigne el tipo correcto, independientemente de mayúsculas/minúsculas.
- **test_no_infiere_tipo_si_no_hay_archivo:** Valida que si `file=None`, el `material.type` se mantenga como `None`.
- **test_no_infiere_tipo_si_ya_esta_definido:** Verifica que la inferencia no sobrescriba un `type` previamente establecido (ej. tipo "docx" para un archivo "documento.pdf").
- **test_extensiones_no_soportadas_no_asignan_tipo:** Prueba que extensiones no soportadas (.bin, .xlsx, .zip, .jpeg, etc.) no asignen ningún tipo.
- **test_nombres_sin_extension_no_asignan_tipo:** Asegura que nombres como "README" o "Makefile" no asignen tipo.
- **test_multiples_puntos_en_nombre_archivo:** Valida que solo la última parte después del último punto se considere extensión (ej. "archivo.backup.pdf" infiere "pdf").

🚧 Casos límite y de borde

- Nombres sin archivo (`file=None`).
- Tipos ya definidos que no deben ser sobrescritos.
- Extensiones no soportadas o poco comunes.
- Nombres con varios puntos, espacios y caracteres especiales.
- Nombres sin extensión explícita.

CourseFormValidationTests

🚀 Funcionalidades que valida

Valida las reglas de negocio del `CourseForm`.

- **test_nombre_obligatorio:** Prueba que el formulario sea válido solo cuando el nombre contiene texto real, rechazando `""` o `"  "`. Comprueba que el error se registre en el campo `name`.
- **test_nombre_limite_150_caracteres:** Evalúa longitudes, aceptando 1, 149 y 150 caracteres, pero rechazando 151 y 999.
- **test_duracion_valores_extremos:** Prueba `duration_hours`, aceptando 0, 1, 100 y 9999, pero rechazando valores negativos como -1 y -100.
- **test_nombre_con_caracteres_especiales:** Confirma que el formulario acepta caracteres Unicode (ruso, chino) y emojis (ej. "🚀 Curso Moderno").

🚧 Casos límite y de borde

- Nombre vacío o solo espacios.
- Nombre exactamente en el límite de 150 caracteres.
- Nombres extremadamente largos (999 caracteres).
- Duraciones negativas frente a duraciones grandes pero válidas.
- Uso de caracteres especiales, Unicode y emojis en el nombre.

ContentFormValidationTests

🚀 Funcionalidades que valida

Valida la lógica de validación del `ContentForm`.

- **test_title_obligatorio:** Evalúa valores de título, verificando que el formulario solo sea válido cuando `title` contiene texto significativo (rechaza `""` y `"  "`). Confirma que el error se registre en `title`.

- **test_title_limite_150_caracteres:** Prueba que el límite de 150 caracteres se respete, aceptando 1, 149 y 150, pero rechazando 151 y 999.
- **test_caracteres_especiales_en_titulo:** Verifica que el formulario acepte títulos con símbolos, tildes, otros alfabetos y emojis (ej. "📚 Lectura").
- **test_combinaciones_campos_vacios_y_espacios:** Evalúa combinaciones, confirmando que un `title` válido con una `description` vacía o solo con espacios es un formulario válido.

🚧 Casos límite y de borde

- Títulos vacíos o con solo espacios.
- Títulos en el límite de longitud permitido (150).
- Descripciones vacías o con espacios en combinación con títulos válidos.
- Títulos con caracteres especiales, Unicode y emojis.