










## Testing del Proyecto: Pretty Exam



### Tests unitarios:




#### 1. Test normalizeSearchTerm.js:

La función testeada tiene como propósito normalizar los términos de búsqueda , es decir, remover acentos y quitar mayúsculas, esto para uniformizar el texto para búsquedas case-insensitive y accent-insensitive, es decir, que sin importar de si escribimos "cual" en los resultados también aparezcan "Cuál" .

Para testear esta función se utilizó el archivo ubicado en normalizeSearchTerm.test.js que realiza cuatro test , en ese mismo directorio podremos ver la función en el archivo normalizeSearchTerm.js. El primer test que prueba el caso base, donde se le pasa una cadena con sus letras en mayúscula e intercaladas también (Mayúsculas y Minúsculas) . Los casos probados con sus salidas esperadas fueron: 'HOLA' → 'hola' y 'HoLa' → 'hola' .

Por otro lado, el segundo caso se probaron los casos en los que debe quitar las tildes y acentos de las palabras , donde se le pasaron diversas cadenas con acentos y tildes para probar esto, al igual estas pruebas contenían mayúsculas y minúsculas. Los casos probados con sus salidas esperadas fueron: 'Cuál' → 'cual', 'matemáticas' → 'matematicas', 'niño' → 'nino' y 'corazón' → 'corazon' .

Por otro lado, en el tercer caso se probaron los casos especiales y edge cases , esto con el fin de que la función no contenga errores cuando reciba cosas funcionalmente no esperadas o previstas. Los casos probados con sus salidas esperadas fueron los siguientes: String vacío: '' → '', Solo espacios: ' ' → '', Valor null: null → '' y Valor undefined: undefined → '' .

Y por último, el último caso se probó la limpieza de espacios de esta función  , con el fin de que también haga un buen sanitizado de esta parte. Los casos probados con sus resultados esperados fueron los siguientes: Espacios laterales: ' hola ' → 'hola' y Caracteres de escape: '\t\nhola\n' → 'hola' .

## 2. Test QuestionFactory.js - Factory Method Tests

La función testada tiene como propósito implementar el patrón Factory 🏭 para crear diferentes tipos de preguntas (selección múltiple y verdadero/falso) con validación integrada 🛡️, esto para garantizar la creación correcta de las preguntas según el tipo de pregunta especificado y que se valide que cumplan con las reglas de negocio establecidas para el tipo de pregunta 📋✅. Para testear esta función se utilizó el archivo ubicado en Proyecto\pretty-exam\src\factories que realiza once test distribuidos en tres grupos principales, donde cada uno hace test de una funcionalidad diferente 🧰📁.

El primer grupo prueba el Factory Method 🧰, enfocándose en validar la funcionalidad principal del patrón Factory que es la creación correcta de instancias según el tipo de pregunta especificado 🧩, verificando que el método estático createQuestion() retorna el tipo de objeto correcto con propiedades inicializadas adecuadamente y manteniendo la integridad de los datos pasados como parámetros 📁🔍. Los casos probados con sus salidas esperadas fueron: crear pregunta de selección múltiple con tipo 'multiple\_choice' → instancia de 'MultipleChoiceQuestion', crear pregunta verdadero/falso con tipo 'true\_false' → instancia de 'TrueFalseQuestion', y crear pregunta con tipo inválido 'invalid' → lanzar error 'Unknown question type: invalid' !.

---

## 3. Test QuestionFactory.js - MultipleChoiceQuestion Tests

La función testada tiene como propósito validar las reglas de negocio específicas para preguntas de selección múltiple 🎯 dentro del patrón Factory, garantizando que las preguntas cumplan con los criterios de calidad establecidos para las preguntas de selección múltiple 📁. Para testear esta función se utilizó el mismo archivo mencionado en la sección anterior, pues este contiene estas validaciones enfocándose en el segundo grupo de tests.

Se probaron los casos de validación para MultipleChoiceQuestion, concentrándose en verificar que la validación identifique correctamente preguntas mal formadas como sin opciones suficientes, o sin respuestas correctas, o con datos faltantes, y también que asegure que sólo preguntas que cumplan con los criterios sean aceptadas ✅🛡️. Los casos probados con sus salidas esperadas fueron: pregunta con datos válidos (texto, categoría y

opciones correctas) → `validate()` retorna `true`, pregunta sin opciones (array vacío) → lanzar error 'At least 2 options required', pregunta con menos de 2 opciones (solo 1 opción) → lanzar error 'At least 2 options required', pregunta sin respuesta correcta (todas las opciones marcadas como falsas) → lanzar error 'At least one correct option required', y formato API (serialización correcta) → objeto con propiedades 'text', 'category\_id' y 'options' con valores esperados 🌐.

---

#### 4. Test QuestionFactory.js - TrueFalseQuestion Tests

La función testeada tiene como propósito validar las reglas específicas para preguntas de verdadero/falso ⚖️ dentro del sistema Factory, asegurando que mantengan las restricciones necesarias para garantizar la calidad de las preguntas de este tipo 📏, para testear esta función se utilizó el tercer grupo de tests del archivo `QuestionFactory.test.js`.

Se probaron los casos de validación para `TrueFalseQuestion`, validando que estas preguntas mantengan exactamente dos opciones (Verdadero/Falso) y que al menos una esté marcada como correcta 🎯, preservando lo requerido para este tipo de preguntas. Los casos probados con sus salidas esperadas fueron los siguientes: pregunta con datos válidos (texto y opciones verdadero/falso) → `validate()` retorna `true` ✅, pregunta sin respuesta correcta (ambas opciones marcadas como falsas) → lanzar error 'Must select True or False' ❌, y formato API (serialización correcta) → objeto con propiedades 'text', 'category\_id' y 'options' con exactamente 2 opciones 📄. Todos los test validaron también la estructura y comportamiento de los métodos `toAPIFormat()` 🧩, asegurando que la serialización sea correcta para su uso en la API 📦.

---

#### 5. Test category.validation.js:

El archivo `category.validation.js` 📁 implementa un conjunto de funciones de validación para el modelo de datos `Category` dentro del sistema. Este módulo proporciona validaciones comprehensivas que abarcan desde la verificación de datos individuales de categorías hasta operaciones de validación en lote. Las funciones principales incluyen `validateCategory` para validar datos básicos de una categoría, `validateCategoryUpdate` para actualizaciones parciales, `validateCategoryId` para verificar identificadores,

`validateCategoryNameUniqueness` para garantizar nombres únicos, y `validateBulkCategories` para operaciones masivas.

El módulo sigue un patrón consistente donde cada función de validación retorna un objeto con propiedades `isValid` (booleano) y `errors` (array de strings), proporcionando tanto el resultado de la validación como detalles específicos de los errores encontrados. Las validaciones implementan reglas de negocio específicas, como longitudes mínimas y máximas para nombres, caracteres permitidos mediante expresiones regulares, y verificaciones de unicidad.

Una aclaración importante a tener en cuenta es que los archivos de validaciones junto a los tests se encuentran en la ruta dentro del repositorio "proyecto/pretty-exam/electron/validations/". Esta ruta aplica para los tests 5, 6 y 7.

## **5.1 Documentación de Tests Implementados**

**5.1.1 Tests para `validateCategory`.** La suite de tests para `validateCategory` verifica el comportamiento de la función principal de validación de categorías. El primer test confirma que una categoría válida con nombre "Matemáticas" pase exitosamente la validación, retornando `isValid: true` y un array de errores vacío. Un segundo test valida que nombres con caracteres especiales permitidos, como "Ciencias Naturales - Física (Básica)", sean aceptados correctamente, verificando que la expresión regular implementada permita guiones, paréntesis y espacios.

Los tests de casos negativos verifican el manejo adecuado de datos inválidos. Se comprueba que al pasar `null` como parámetro, la función retorne el error "Los datos de la categoría son requeridos". Para nombres vacíos, se verifica que se genere un error relacionado con el nombre de categoría. El test de longitud mínima confirma que nombres de un solo carácter sean rechazados con el mensaje "al menos 2 caracteres". Finalmente, se valida que caracteres no permitidos como "@#\$%" generen el error "caracteres no permitidos".

**5.1.2 Tests para `validateCategoryUpdate`.** Los tests para `validateCategoryUpdate` verifican la funcionalidad de validación para actualizaciones parciales de categorías. El test positivo confirma que datos de actualización válidos con un nombre nuevo sean aceptados correctamente. El test negativo principal verifica que un objeto vacío `{}` sea rechazado con el error "al menos un campo", asegurando que las actualizaciones contengan al menos un campo modificable.

**5.1.3 Tests para validateCategoryId.** La validación de identificadores de categoría se prueba verificando que identificadores válidos como el número 1 pasen la validación exitosamente. El test negativo confirma que identificadores inválidos como -1 sean rechazados con el error "número entero positivo", garantizando que solo se acepten identificadores que sean números enteros positivos.

**5.1.4 Tests para validateCategoryNameUniqueness.** Esta suite de tests verifica la funcionalidad de unicidad de nombres de categorías. El test principal confirma que un nombre como "Historia" sea validado como único cuando no existe en el array de categorías existentes [{category\_id: 1, name: 'Matemáticas'}, {category\_id: 2, name: 'Ciencias'}]. El test de duplicidad verifica que intentar usar "Matemáticas" cuando ya existe genere el error "Ya existe una categoría".

Un test especializado verifica que durante actualizaciones, una categoría pueda mantener su propio nombre. Al proporcionar el parámetro excludeId: 1, se confirma que "Matemáticas" sea válido para la categoría con ID 1, incluso si ya existe una categoría con ese nombre, permitiendo actualizaciones que no cambien el nombre.


**5.1.5 Tests para validateBulkCategories.** Los tests de validación en lote verifican operaciones con múltiples categorías simultáneamente. El test positivo confirma que un array de categorías válidas [{name: 'Matemáticas'}, {name: 'Ciencias'}, {name: 'Historia'}] pase la validación completamente. Los tests negativos verifican el manejo de casos edge: se confirma que pasar un string en lugar de un array genere el error "array de categorías", y que un array vacío [] sea rechazado con "al menos una categoría".

El test más complejo verifica la detección de duplicados dentro del mismo lote de operación. Al intentar crear categorías con nombres duplicados [{name: 'Matemáticas'}, {name: 'Matemáticas'}], se confirma que la función genere el error "duplicados", previniendo la creación de categorías con nombres idénticos en una sola operación.

## 5.2 Cobertura y Robustez de las Validaciones


Los tests implementados proporcionan una cobertura comprehensiva de los casos de uso principales y casos edge del módulo de validación. Cada función es probada tanto con datos válidos como inválidos, verificando que las validaciones implementen correctamente las reglas de negocio establecidas. La estructura consistente de los tests, utilizando la sintaxis expect().toBe() y expect().toContain(), garantiza verificaciones precisas tanto de valores booleanos como de contenido específico de mensajes de error.

## 6. Test exam.validation.js:

El archivo exam.validation.js  constituye un módulo integral de validación para el modelo de datos Exam dentro del sistema de gestión de exámenes. Este módulo implementa un conjunto robusto de funciones de validación que abarcan desde la verificación de datos básicos de exámenes hasta validaciones complejas que involucran la asociación con preguntas. Las funciones principales incluyen `validateExam` para validar datos completos de un examen, `validateExamUpdate` para actualizaciones parciales, `validateExamId` para verificar identificadores, y `validateExamWithQuestions` para validar exámenes con preguntas asociadas.

El módulo establece reglas de negocio específicas que garantizan la integridad de los datos del examen. Para el nombre del examen, se requiere un mínimo de 3 caracteres y un máximo de 200, mientras que la descripción opcional puede contener hasta 1000 caracteres. La duración del examen debe estar comprendida entre 5 minutos y 1440 minutos (24 horas), implementando límites prácticos para el sistema educativo. Todas las funciones mantienen consistencia en su estructura de retorno, proporcionando objetos con propiedades `isValid` y `errors` para facilitar el manejo de resultados de validación.


### 6.1 Documentación de Tests Implementados

**6.1.1 Tests para `validateExam`** . La suite de tests para `validateExam` verifica exhaustivamente la funcionalidad de validación principal del módulo. El primer test confirma que un examen completamente válido con todos los campos opcionales poblados pase la validación exitosamente. Este test utiliza datos como `{name: 'Examen de Matemáticas', description: 'Examen de álgebra básica', duration_minutes: 60}`, verificando que la función retorne `isValid: true` y un array de errores vacío.

Un test especializado valida el comportamiento con campos opcionales nulos, confirmando que un examen con `{name: 'Test', description: null, duration_minutes: null}` sea considerado válido. Esto demuestra que el sistema maneja adecuadamente los valores nulos para campos no requeridos, permitiendo flexibilidad en la creación de exámenes.


Los tests de validación negativa verifican el manejo robusto de datos inválidos. Se confirma que pasar `null` como parámetro genere el error "Los datos del examen son

requeridos". Para nombres inválidos, se verifica que nombres vacíos y nombres demasiado cortos como "Ab" sean rechazados con mensajes específicos sobre longitud mínima. Los tests de duración verifican que valores negativos como -5 generen el error "mayor a 0 minutos", mientras que duraciones excesivas como 2000 minutos sean rechazadas con el mensaje "1440 minutos".


**6.1.2 Tests para validateExamUpdate** . Los tests para `validateExamUpdate` validan la funcionalidad especializada de actualizaciones parciales de exámenes. Esta función implementa una lógica compleja que permite actualizar campos individuales sin requerir una validación completa del objeto examen. El primer test confirma que una actualización simple del nombre `{name: 'Nombre actualizado'}` sea procesada correctamente, retornando una validación exitosa.

Un test particularmente importante verifica que actualizaciones que no incluyen el nombre, como `{duration_minutes: 90}`, sean validadas usando lógica parcial específica. Este comportamiento es crucial porque permite modificar campos opcionales sin proporcionar el nombre requerido, implementando una validación contextual según los campos presentes en la actualización.

El test de validación negativa confirma que objetos vacíos `{}` sean rechazados con el error "al menos un campo", asegurando que las operaciones de actualización contengan modificaciones reales y no sean operaciones vacías que podrían indicar errores en la lógica de aplicación.

**6.1.3 Tests para validateExamId** . La validación de identificadores de examen se verifica mediante tests que cubren diversos tipos de entrada. El test principal confirma que identificadores numéricos válidos como 1 pasen la validación exitosamente. Un test adicional verifica que identificadores proporcionados como strings numéricos, como '5', sean convertidos y validados correctamente, demostrando la robustez de la función ante diferentes tipos de datos de entrada.

Los tests negativos verifican el manejo de identificadores inválidos. Se confirma que valores negativos como -1 sean rechazados con el error "número entero positivo", mientras que valores nulos generen el error específico "requerido". Esta validación es fundamental para mantener la integridad referencial en operaciones que involucran identificadores de examen.

**6.1.4 Tests para validateExamWithQuestions** . La función `validateExamWithQuestions` implementa validación compleja que combina la validación de

datos del examen con la verificación de preguntas asociadas. El test principal confirma que un examen válido con un array de IDs de preguntas [1, 2, 3] pase todas las validaciones, verificando tanto la integridad del examen como la validez de las asociaciones con preguntas.

El test de array vacío verifica que intentar crear un examen sin preguntas asociadas (`questionIds = []`) sea rechazado con el error "al menos una pregunta". Esta validación implementa una regla de negocio importante que garantiza que todos los exámenes tengan contenido evaluable antes de ser creados en el sistema.

Un test especializado verifica la detección de IDs duplicados en el array de preguntas. Al proporcionar [1, 2, 2, 3], se confirma que la función genere el error "duplicados", previniendo asociaciones erróneas que podrían resultar en comportamientos inesperados durante la presentación del examen. Esta validación utiliza la estructura `Set` para detectar eficientemente duplicados mediante comparación de tamaños.

## 6.2 Cobertura Integral y Validación de Reglas de Negocio

Los tests implementados proporcionan una cobertura comprehensiva que abarca todos los aspectos críticos del módulo de validación de exámenes. Cada función es sometida a pruebas tanto con datos válidos como con múltiples escenarios de datos inválidos, asegurando que las validaciones implementen correctamente las reglas de negocio establecidas. La estructura de los tests utiliza patrones consistentes con `expect().toBe()` para verificaciones booleanas y `expect().toContain()` para validar contenido específico de mensajes de error.

La arquitectura de testing demuestra particular atención a casos edge y situaciones límite, como duraciones en los extremos del rango permitido, nombres en los límites de longitud, y combinaciones complejas de campos opcionales. Esta aproximación exhaustiva garantiza que el módulo de validación funcione correctamente en todas las condiciones operativas del sistema, proporcionando confiabilidad y robustez en la gestión de datos de exámenes.

---

## 7. Test `question.validation.js`



El archivo `question.validation.js` constituye un módulo especializado de validación para el modelo de datos `Question`, implementando un sistema robusto de verificación que maneja la complejidad inherente a las preguntas de examen con sus opciones asociadas. Este módulo define cuatro funciones principales: `validateQuestion` para validación completa de preguntas, `validateOptions` para verificación específica de opciones de respuesta, `validateQuestionUpdate` para actualizaciones parciales, y `validateQuestionId` para verificación de identificadores.


La arquitectura del módulo reconoce dos tipos fundamentales de preguntas: `multiple_choice` para preguntas de opción múltiple y `true_false` para preguntas de verdadero/falso. Cada tipo implementa reglas de validación específicas que reflejan sus características únicas. Las preguntas de opción múltiple requieren entre 2 y 6 opciones, mientras que las preguntas de verdadero/falso deben tener exactamente 2 opciones con una sola respuesta correcta. El texto de las preguntas debe contener entre 10 y 1000 caracteres, garantizando claridad sin excesiva verbosidad.

## 7.1 Documentación de Tests Implementados

**7.1.1 Tests para `validateQuestion` ✓.** La suite de tests para `validateQuestion` verifica exhaustivamente la validación de preguntas completas con todos sus componentes. El primer test confirma que una pregunta de opción múltiple válida con estructura completa pase todas las validaciones. Este test utiliza un objeto con `text: 'Esta es una pregunta de prueba con suficientes caracteres'`, `type: 'multiple_choice'`, `category_id: 1`, y opciones válidas, verificando que la función retorne `isValid: true` con un array de errores vacío.


Un test especializado valida preguntas de verdadero/falso, confirmando que la estructura específica `{text: 'Esta es una pregunta de verdadero o falso', type: 'true_false', category_id: null, options: [{text: 'Verdadero', isCorrect: true}, {text: 'Falso', isCorrect: false}]}` sea procesada correctamente. Este test demuestra el manejo adecuado de campos opcionales nulos como `category_id` y la validación específica para preguntas binarias.

Los tests de validación negativa verifican el manejo robusto de datos inválidos. Se confirma que pasar `null` genere el error "Los datos de la pregunta son requeridos". Para textos inválidos, se verifica que textos vacíos y textos insuficientemente descriptivos como "Corto" sean rechazados con mensajes específicos sobre longitud mínima de 10 caracteres. El test de tipo inválido confirma que tipos no reconocidos como `'invalid_type'` generen el error "tipo de pregunta debe ser".

**7.1.2 Tests para validateOptions** . La función `validateOptions` implementa validación especializada para las opciones de respuesta, siendo fundamental para garantizar la integridad estructural de las preguntas. El test principal confirma que opciones válidas de opción múltiple con estructura `[[{text: 'Opción A', isCorrect: true}, {text: 'Opción B', isCorrect: false}, {text: 'Opción C', isCorrect: false}]]` sean procesadas sin errores, retornando un array vacío que indica validación exitosa.


Un test crítico verifica la detección de opciones duplicadas. Al proporcionar opciones con textos idénticos `[[{text: 'Opción A', isCorrect: true}, {text: 'Opción A', isCorrect: false}]]`, se confirma que la función genere el error "mismo texto". Esta validación utiliza normalización de texto mediante `trim().toLowerCase()` para detectar duplicados incluso con diferencias en espacios o capitalización.

El test de opciones sin respuestas correctas verifica que arrays donde todas las opciones tienen `isCorrect: false` sean rechazados con el error "al menos una opción correcta". Un test especializado para preguntas de verdadero/falso confirma que tener múltiples opciones correctas `[[{text: 'Verdadero', isCorrect: true}, {text: 'Falso', isCorrect: true}]]` genere el error "exactamente una opción correcta", implementando la regla de negocio específica para este tipo de pregunta.

**7.1.3 Tests para validateQuestionUpdate** . Los tests para `validateQuestionUpdate` validan la funcionalidad de actualizaciones parciales que permite modificar campos específicos sin requerir validación completa del objeto pregunta. El test principal confirma que una actualización simple del texto `{text: 'Texto actualizado con suficientes caracteres'}` sea procesada correctamente, demostrando que la función puede validar campos individuales manteniendo las mismas reglas que la validación completa.

Esta función implementa lógica condicional que aplica validaciones específicas solo a los campos presentes en `updateData`. Por ejemplo, si solo se actualiza el texto, no se requiere validar el tipo o las opciones, proporcionando flexibilidad operativa crucial para interfaces de usuario que permiten modificaciones granulares.

El test de validación negativa confirma que objetos vacíos `{}` sean rechazados con el error "al menos un campo", asegurando que las operaciones de actualización contengan modificaciones sustanciales y no sean operaciones vacías que podrían indicar errores en la lógica de aplicación o interfaz de usuario.

**7.1.4 Tests para validateQuestionId** . La validación de identificadores de pregunta se verifica mediante tests que cubren los casos esenciales de identificación. El test

principal confirma que identificadores numéricos válidos como `1` pasen la validación exitosamente, retornando `isValid: true` y un array de errores vacío. Esta validación es fundamental para operaciones de consulta, actualización y eliminación que requieren referencias específicas a preguntas existentes.

Los tests negativos verifican el manejo robusto de identificadores inválidos. Se confirma que valores negativos como `-1` sean rechazados con el error "número entero positivo", mientras que valores nulos generen el error "requerido". Esta validación mantiene la integridad referencial del sistema, previniendo operaciones sobre registros inexistentes o con identificadores malformados.

## 7.2 Arquitectura de Validación Compleja y Cobertura Integral 🏗️

El módulo de validación de preguntas implementa una arquitectura sofisticada que maneja la complejidad inherente a preguntas con opciones múltiples y reglas de negocio específicas por tipo. Los tests demuestran particular atención a la validación jerárquica, donde la validación de preguntas incorpora la validación de opciones, creando un sistema de verificación en cascada que garantiza la integridad de todos los componentes.

La cobertura de tests abarca escenarios críticos como validación de unicidad de opciones, verificación de respuestas correctas según el tipo de pregunta, y manejo de actualizaciones parciales. Esta aproximación exhaustiva garantiza que el módulo funcione correctamente en todas las condiciones operativas del sistema educativo, proporcionando confiabilidad en la gestión de contenido evaluativo y manteniendo la calidad pedagógica de los instrumentos de evaluación.

---

## 8. Tests `formatTimeSeconds`:

La función `formatTimeSeconds` se encarga de convertir una cantidad de segundos en un formato legible para la interfaz de usuario, retornando `hh:mm:ss` cuando hay horas completas o `mm:ss` si no las hay. Se utiliza en `src/components/examTimer.jsx` para mostrar el tiempo restante de un examen en curso.

La función maneja entradas de cero segundos devolviendo '0:00' y garantiza el uso de ceros a la izquierda en minutos y segundos cuando corresponde, mejorando la legibilidad del temporizador.

## 8.1 Documentación de Tests Implementados

### 8.1.1 Tests para `formatTimeSeconds`

La suite de pruebas valida la conversión precisa de segundos a los formatos requeridos. Los tests confirman que valores iguales o superiores a 3600 segundos, como 3661, se convierten correctamente a '1:01:01'. También se verifica que valores inferiores a una hora, como 59 y 125 segundos, retornen '0:59' y '2:05' respectivamente.

Además, se cubre el caso límite donde 0 segundos devuelve '0:00', asegurando consistencia para valores nulos o de inicio.

---

## 9. Tests `formatTimeString`:

La función `formatTimeString` toma una cadena de fecha (en formato ISO o similar) y devuelve únicamente la hora en formato HH:mm, adaptada a la localización española. Se utiliza en `src/pages/examHistory.jsx` para mostrar a qué hora fue tomado un examen.

Si el valor de entrada es vacío o `null`, devuelve una cadena vacía, previniendo errores en la visualización.

## 9.1 Documentación de Tests Implementados

### 9.1.1 Tests para `formatTimeString`

Los tests comprueban que las cadenas de fecha válidas se conviertan correctamente a horas.

Entradas como '2025-07-19T15:30:00' y '2025-07-19T05:07:00' producen '15:30' y '05:07' respectivamente.

Casos especiales verifican que entradas vacías o `null` devuelvan "", evitando fallos cuando no hay datos de hora.

## 10. Tests `formatMinutes`

La función `formatMinutes` convierte cantidades de minutos a un formato `h:mm`. Es utilizada en `src/pages/examHistory.jsx` para mostrar la duración total que un usuario invierte en resolver un examen.

Si el valor es `0`, devuelve `'0:00'`, y para valores no numéricos o inválidos como `undefined`, `null` o cadenas, retorna `'N/A'`, garantizando consistencia en la interfaz.

### 10.1 Documentación de Tests Implementados

#### 10.1.1 Tests para `formatMinutes`

Los tests verifican que valores válidos como `61` y `125` minutos se conviertan correctamente a `'1:01'` y `'2:05'`.

Se incluyen validaciones para el caso de `0` minutos, que debe devolver `'0:00'`. Se confirman respuestas `'N/A'` para valores inválidos (`'abc'`, `undefined`, `null`), asegurando robustez ante datos inconsistentes.

---

## 11. Tests `formatDate`

La función `formatDate` transforma cadenas de fecha (`YYYY-MM-DD` o ISO) a una representación larga en español, incluyendo día de la semana, día del mes, mes y año. Es utilizada en `src/pages/examHistory.jsx` para mostrar en qué día fue tomado cada intento de examen. Si la cadena es vacía o `null`, retorna `'N/A'`, manteniendo coherencia con el resto de utilidades.

### 11.1 Documentación de Tests Implementados

#### 11.1.1 Tests para `formatDate`

Las pruebas confirman que fechas como `'2025-07-19'` se conviertan correctamente a

un formato largo, coincidiendo con expresiones como "sábado 19 de julio de 2025". También verifican que valores vacíos o `null` devuelvan 'N/A', garantizando un manejo seguro de entradas inválidas.

---

## 12. Tests para `boldMarkdownToHtml`

La suite de tests para `boldMarkdownToHtml` asegura que la conversión de Markdown a HTML funcione correctamente y cubra casos comunes y límites:

- **Conversión estándar de negrilla:** Se verifica que entradas como 'Esto es `**importante**`' se conviertan en 'Esto es `<b>importante</b>`'. Casos con múltiples bloques, como `'**Hola** **Mundo**'`, producen `'<b>Hola</b> <b>Mundo</b>'`.
- **Texto sin formato Markdown:** Se confirma que cadenas sin `**` (por ejemplo, 'Sin formato') se devuelvan intactas, sin alteraciones.
- **Valores vacíos o nulos:** Se valida que entradas vacías (`''`), `null` o `undefined` se devuelvan tal cual, evitando errores o transformaciones no deseadas.
- **Conversión de múltiples negritas en la misma cadena:** Entradas como `'**uno** y **dos**'` se convierten correctamente a `'<b>uno</b> y <b>dos</b>'`.
- **Compatibilidad con saltos de línea:** Se prueba que cadenas con saltos de línea, como `'línea1\n**línea2**'`, preserven el salto (`\n`) y conviertan solo la porción en negrita, produciendo `'línea1\n<b>línea2</b>'`.

### 12.1 Robustez y Aplicación en la Interfaz

La función y sus pruebas aseguran una integración estable con la capa de presentación de la aplicación, permitiendo que contenido generado dinámicamente con formato Markdown pueda renderizarse de manera consistente en HTML sin afectar la estructura del texto (incluyendo saltos de línea y contenido no marcado).

Gracias a su comportamiento defensivo frente a entradas nulas o vacías, evita errores en el flujo de `ai.controller.js` cuando el modelo Gemini produce respuestas sin formato o con valores inesperados, garantizando una visualización limpia y segura en la interfaz.

## Ejecución de los Tests Planteados

```
PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam> npm run tests
> pretty-exam@0.0.0 tests
> node --experimental-vmodules node_modules/jest/bin/jest.js --coverage

(node:12448) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use "node --trace-warnings ..." to show where the warning was created)
PASS electron/validations/question.validation.test.js
PASS src/utills/format.test.js
PASS src/factories/QuestionFactory.test.js
PASS electron/validations/category.validation.test.js
PASS electron/validations/exam.validation.test.js
PASS src/utills/normalizeSearchTerm.test.js
PASS electron/utills/markdown.test.js

-----
File                                % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #s
-----
All files                          80.96     73.17     97.5      80.8
electron/utills                    100       100       100       100
markdown.js                        100       100       100       100
electron/validations               76.85     71.54     95.23     76.56
  category.validation.js           85.71      84      88.88     85.48    30,34,68,83,105-106,154,164-165
  exam.validation.js              78.94     71.42     100      78.94    32,36,43,45,53,60,96-99,108,110,112,114,158,170
  question.validation.js          69.9      66.07     100      69.3    33,37,44,52,58,80-81,89,93,100-101,108,112,114,120,164,168,170,172,179-183,189-190,196-201
src/factories                      93.75     71.42     100       100
  QuestionFactory.js              93.75     71.42     100       100    18-23,48-64
src/utills                        96.96     94.44     100      96.55
  format.js                       96.66     92.85     100      96.15    54
  normalizeSearchTerm.js          100       100       100       100
-----

Test Suites: 7 passed, 7 total
Tests:       81 passed, 81 total
Snapshots:   0 total
Time:        1.135 s
Ran all test suites.
```


## Analizador estático: linter


Los **linters** son herramientas fundamentales en el desarrollo de software porque analizan el código fuente automáticamente para detectar errores, malas prácticas, inconsistencias de estilo y posibles bugs antes de que el código se ejecute. Para el desarrollo del proyecto manteniendo la calidad, legibilidad y consistencia del código del proyecto se utilizó un linter para JavaScript llamado **ESLint**.



ESLint es una herramienta de análisis estático de código JavaScript que permite identificar y reportar patrones problemáticos encontrados en el código. Es altamente configurable y extensible, lo que la convierte en la opción ideal para proyectos que utilizan

múltiples tecnologías como React, Node.js y Electron ⚡. Permite definir en su configuración una serie de reglas de estilo y buenas prácticas, facilitando la detección de errores y la consistencia de buenas prácticas.

El proyecto cuenta con un archivo de configuración con el nombre `.eslintrc.cjs`  el cual permite definir un conjunto de reglas utilizadas durante el análisis. Este linter es compatible para JavaScript, React y Electron que son las herramientas básicas para este proyecto.

A continuación se presenta el contenido del archivo `.eslintrc.cjs`  que contiene la configuración del linter ESLint para la definición de estilo y buenas prácticas para la detección de errores y calidad del código. Esta configuración fue definida en un inicio durante la construcción del proyecto.

```
module.exports = {
  root: true,
  env: { browser: true, es2020: true },
  extends: [
    'standard',
    'plugin:react/recommended',
    'plugin:react-hooks/recommended',
    'plugin:prettier/recommended'
  ],
  ignorePatterns: ['dist', '.eslintrc.cjs'],
  plugins: ['react-refresh'],
  rules: {
    'react-refresh/only-export-components': [
      'warn',
      { allowConstantExport: true },
    ],
    'react/prop-types': 'off',
    'react/react-in-jsx-scope': 'off',
    'prettier/prettier': [
      'error',
      {
        endOfLine: 'auto',
      },
    ],
  ],
  settings: {
    react: {
      version: 'detect',
    },
  },
}
```



### Entornos configurados:

- **browser:** true - Para código que se ejecuta en el navegador
- **es2020:** true - Soporte para características de ECMAScript 2020


### Configuraciones extendidas:

- **standard:** Configuración estándar de JavaScript para mantener consistencia en el estilo
- **plugin:react/recommended:** Reglas recomendadas para proyectos React
- **plugin:react-hooks/recommended:** Reglas específicas para React Hooks
- **plugin:prettier/recommended:** Integración con Prettier para formateo automático

### Plugins utilizados:

- **react-refresh:** Para desarrollo con React y Hot Module Replacement (HMR)
- Se evita la advertencia innecesaria sobre la importación de React en JSX (**react/react-in-jsx-scope**).
- Se integra prettier como verificador de estilo, con énfasis en mantener saltos de línea compatibles según sistema operativo (**endOfLine: 'auto'**).

## Agregado

Durante el desarrollo del proyecto, la integración de nuevas funcionalidades y tests unitarios se debió realizar algunas modificaciones al archivo `.eslintrc.cjs`  agregando nuevas reglas y overrides para generar revisiones más específicas sobre archivos de test .js y .jsx usando la


```
overrides: [  
  {  
    files: ['**/*.test.js', '**/*.test.jsx'],  
    env: {  
      jest: true,  
    },  
    rules: {  
      // Reglas específicas para archivos de test  
      'no-console': 'off',  
      'max-lines': 'off',  
      'max-lines-per-function': 'off',  
    },  
  },  
]
```

```

    'prefer-const': 'error',
    'no-var': 'error',
    'no-unused-vars': ['error', { argsIgnorePattern: '^_' }],
    'jest/no-disabled-tests': 'warn',
    'jest/no-focused-tests': 'error',
    'jest/prefer-to-be': 'error',
    'jest/prefer-to-have-length': 'error',
    'jest/valid-expect': 'error',
    'jest/no-identical-title': 'error',
    'jest/prefer-strict-equal': 'warn',
    'jest/no-test-return-statement': 'error',
    'no-magic-numbers': 'off'
  },
},
],
}

```

Se implementó una sección completa de overrides dedicada exclusivamente a archivos de testing, aplicando reglas especializadas para archivos con patrones `**/*.test.js` y `**/*.test.jsx`. Esta modificación introduce el entorno Jest y configura las reglas específicas que brindan flexibilidad para escribir tests efectivos con buenas prácticas de calidad de código. Las reglas implementadas incluyen la permisión de `console.log` para debugging, eliminación de restricciones de longitud de archivo y función (comunes en tests extensos), y la incorporación de reglas específicas de Jest como la prohibición de tests focalizados (`.only`) y la validación de assertions correctas. Esta configuración especial y personalizada mejora el desarrollo de pruebas sin comprometer los estándares de calidad del código de producción.

Dentro del archivo `package.json`  donde se encuentra la información principal del proyecto (nombre, versión, descripción), así como las dependencias del proyecto y los scripts personalizados para ejecutar comandos desde la terminal como es el caso de la ejecución del comando `.eslint` para la automatización del linter desde la raíz del proyecto sobre todo el mismo, y obtener un informe de errores y violación de reglas definidas en la configuración de linter.

```

"scripts": {
  "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
}

```

**"eslint .":** Ejecuta ESLint sobre el directorio actual (.), es decir, analiza todos los archivos del proyecto.

**--ext js,jsx :** Indica que ESLint debe analizar archivos con extensiones .js y .jsx.

**--report-unused-disable-directives :** Hace que ESLint muestre advertencias si existen reglas desactivadas (con comentarios como // eslint-disable) pero que no están siendo usadas en ninguna parte del archivo.

**--max-warnings 0:** Configura ESLint para que no permita advertencias (warnings). Si encuentra una sola advertencia, el comando termina con error.

## Ejecución del linter

Ya seleccionado y configurado el linter para el desarrollo del proyecto es importante saber cómo ejecutar el linter automáticamente para que evalúe todo el proyecto desde la raíz, brindando como resultado un informe sobre las violaciones de las reglas establecidas en la configuración del linter y malas prácticas en el código.

Para ello se ejecuta el comando: **npm run lint** 🔥

A modo de ejemplo se realizó la ejecución del anterior comando durante la etapa de construcción de los test unitarios para notar la efectividad de este. La imagen a continuación deja la ejecución del comando a partir del cual se despliega una lista de todos los archivos del proyecto donde se encontraron errores o incumplimiento de las reglas y buenas prácticas de código.

```

PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam> npm run lint
> pretty-exam@0.0.0 lint
> eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0

C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam\electron\controllers\category.controller.js
  1:20  error  'Question' is defined but never used  no-unused-vars
  7:31  error  Insert ','                             prettier/prettier
 13:17  error  Replace '(data)' with 'data'           prettier/prettier
 16:24  error  Insert ','                             prettier/prettier
 34:1   error  Delete '.....'                      prettier/prettier
 38:1   error  Delete '.....'                      prettier/prettier
 50:17  error  Replace '(id)' with 'id'               prettier/prettier
 52:33  error  Insert ','                             prettier/prettier
 54:1   error  Delete '....'                         prettier/prettier
 58:1   error  Delete '....'                         prettier/prettier
 68:1   error  Delete '....'                         prettier/prettier
 71:4   error  Insert ','                             prettier/prettier
 74:34  error  Insert 'se'                           prettier/prettier

C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam\electron\controllers\question.controller.js
 132:29  error  Insert ','                             prettier/prettier
 139:1   error  Delete '....'                         prettier/prettier
 145:72  error  Insert ','                             prettier/prettier
 155:22  error  Replace 'se.....whereClause,se.....{:[Op.or]:searchConditions}se.....]' with 'whereClause,{:[Op.or]:searchConditions
 163:36  error  Insert ','                             prettier/prettier
 176:39  error  Insert ','                             prettier/prettier
 184:24  error  Replace '(categoryId)' with 'categoryId' prettier/prettier
 191:39  error  Insert ','                             prettier/prettier

C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam\electron\ipcHandlers\category.ipc.js
   6:3   error  Delete '...'                         prettier/prettier
  11:1   error  Delete '...'                         prettier/prettier
  16:1   error  Replace '....' with '...'            prettier/prettier
  21:1   error  Delete '...'                         prettier/prettier
  26:1   error  Delete '...'                         prettier/prettier

```

Por ejemplo, el linter resalta la importancia de reemplazar o eliminar los paréntesis al indicar una variable. Por ejemplo: *Replace (data) with 'data'*

También deja notar el error provocado por espacios innecesarios como indentación, dado por la instrucción: *Delete '....'* en archivos como [category.controller.js](#), según las definición de prettier como formateador de código para mejorar la legibilidad del código fuente del proyecto.

```

  77:32  error  Replace '(categoryId)' with 'categoryId'  react-hooks/exhaustive-deps
  85:1   error  Delete '.....'                         prettier/prettier
  90:24  error  Replace '(category)' with 'category'      prettier/prettier
 138:40  error  Replace '(categoryId)' with 'categoryId'  prettier/prettier
 148:40  error  Replace '(categoryId)' with 'categoryId'  prettier/prettier
 168:30  error  Insert ','                               prettier/prettier
 175:1   error  Delete '.....'                         prettier/prettier
 198:9   error  Delete '...'                             prettier/prettier
 207:24  error  Insert ','                               prettier/prettier
 210:11  error  Delete '...'                             prettier/prettier
 214:31  error  Insert ','                               prettier/prettier

```

Al final de todo el despliegue de errores y advertencias encontrado por el linter en todo el proyecto se obtuvieron las siguientes estadísticas:

```

X290 problems (286 errors, 4 warnings)
  251 errors and 0 warnings potentially fixable with the '--fix' option.

PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam>

```

Además de los 290 problemas encontrados entre errores y warnings, también resalta la posibilidad de arreglar o corregir un número de problemas de forma automática por el linter ESLint al ejecutar el comando: **npx eslint . --fix**, **npx eslint --fix** ó **npx prettier --write .**:

```

PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam> npx prettier --write .
.prettierrc 54ms
coverage/coverage-final.json 358ms
coverage/lcov-report/base.css 106ms
coverage/lcov-report/block-navigation.js 51ms
coverage/lcov-report/prettify.css 14ms
coverage/lcov-report/prettify.js 193ms
coverage/lcov-report/sorter.js 28ms
electron/config/database.js 8ms (unchanged)
electron/controllers/ai.controller.js 61ms (unchanged)
electron/controllers/category.controller.js 16ms (unchanged)
electron/controllers/exam.controller.js 28ms (unchanged)
electron/controllers/option.controller.js 9ms (unchanged)
electron/controllers/question.controller.js 40ms (unchanged)
electron/controllers/result.controller.js 8ms (unchanged)
electron/controllers/userAnswer.controller.js 6ms (unchanged)
electron/ipcHandlers/ai.ipc.js 5ms (unchanged)
electron/ipcHandlers/category.ipc.js 7ms (unchanged)
electron/ipcHandlers/exam.ipc.js 4ms (unchanged)
electron/ipcHandlers/option.ipc.js 4ms (unchanged)
electron/ipcHandlers/question.ipc.js 6ms (unchanged)
electron/ipcHandlers/result.ipc.js 6ms (unchanged)
electron/ipcHandlers/userAnswer.ipc.js 7ms (unchanged)
electron/main.js 14ms (unchanged)
electron/models/category.model.js 3ms (unchanged)
electron/models/exam.model.js 7ms (unchanged)
electron/models/index.js 5ms (unchanged)
electron/models/option.model.js 4ms (unchanged)
electron/models/question.model.js 7ms (unchanged)
electron/models/result.model.js 6ms (unchanged)
electron/models/userAnswer.model.js 4ms (unchanged)
electron/preload.js 14ms (unchanged)
electron/utils/markdown.js 3ms (unchanged)
electron/utils/markdown.test.js 4ms (unchanged)
electron/validations/category.validation.js 12ms (unchanged)
electron/validations/category.validation.test.js 14ms (unchanged)
electron/validations/exam.validation.js 17ms (unchanged)
electron/validations/exam.validation.test.js 17ms (unchanged)
electron/validations/question.validation.js 11ms (unchanged)
electron/validations/question.validation.test.js 16ms (unchanged)
package-lock.json 110ms (unchanged)

```

```

PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam> npx eslint . --fix

C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam\electron\controllers\category.controller.js
  1:20  error  'Question' is defined but never used  no-unused-vars

C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam\electron\controllers\question.controller.js
  2:38  error  'UserAnswer' is defined but never used  no-unused-vars

X2 problems (2 errors, 0 warnings)

```

Al ejecutar el comando, se corrigieron todos los errores que se podían arreglar automáticamente con el linter. Como se observa en la imagen, quedaron 2 errores debido a

que estos se deben corregir manualmente en los archivos indicados. Al corregir estos errores y ejecutar nuevamente **npm run lint** se obtuvo lo siguiente:

```
PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam> npm run lint
> pretty-exam@0.0.0 lint
> eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0
PS C:\Users\Bellic12\Documents\2025-1_IngeSoft1\Proyecto\pretty-exam>
```

Esto indica que ya no hay más errores de formato ni advertencias.

## Conclusión

Los test unitarios y los linters son herramientas esenciales para garantizar la calidad del software desde las etapas tempranas del desarrollo. Mientras que los linters ayudan a mantener un código limpio, coherente y libre de errores sintácticos o de estilo, los test unitarios aseguran que cada componente del sistema funcione correctamente de forma aislada. Ambos promueven un desarrollo más confiable, facilitan la detección temprana de errores y reducen significativamente el tiempo de depuración y mantenimiento, logrando aplicaciones más escalables.