*RÚBRICA DE EVALUACIÓN DE ENTREGAS DEL PROYECTO FINAL*

Una **rúbrica** es un conjunto de criterios y normas que sirven para evaluar el nivel de desempeño en una tarea. Es una herramienta de calificación que simplifica la tarea de corrección y permite lograr una evaluación más objetiva y transparente tanto para el evaluador como el evaluado.

Adicionalmente, la rúbrica ayuda a igualar y alinear los criterios de evaluación entre los diferentes docentes y tutores.

**En resumen, la rúbrica:**

1. Define los criterios de evaluación de cada entrega de proyecto final.
2. Describe y detalla los puntos clave que debe tener cada entrega.
3. Facilita y transparenta la corrección de cada trabajo.

*Introducción a las Rúbricas*

Las rúbricas son un esfuerzo por crear un método unificado y homogéneo de evaluar los desafíos, brindando transparencia a las correcciones. Sin embargo es importante aclarar que es un material de referencia y no tanto una guía, ya que en el mismo se sientan las bases de lo que se espera de los desafíos. Hay que tener en cuenta que cada estudiante hace un recorrido único y es por eso que es fundamental que tomen este material como referencia para cada situación.

Dicho eso, se plantean tres estados para clasificar la calidad y el desempeño de los desafío: **bajo, correcto y óptimo.** Se entiende que cada columna, itera sobre la base de la otra, y se asume que por ejemplo, errores de la columna **baja**, no van a ser encontrados en un trabajo **correcto** ni **óptimo**. A su vez, se espera que el estudiante mejore su desafío con cada entrega, por lo que algunas correcciones **óptimas** son *degradadas* a solo **correctas** o quizás **bajas**, para momentos más avanzados de las entregas, denotando el aumento de exigencia con cada nueva clase.

También es importante notar que no todos los estudiantes van a llegar a entregas **óptimas** ya que en esta columna suelen encontrarse características de un trabajo superador. Un trabajo **óptimo** es al que esperamos que todos aspiren, ya que es donde encontramos la mayor cantidad de exigencia. Lo importante es notar que esto no equivale a un trabajo **correcto** sea considerado como erróneo, sino donde hay mucho espacio para la mejora y el feedback.

*ENTREGA DEL PROYECTO FINAL*

**Componentes:**

1. Navbar
2. Catálogo
3. Detalle de producto
4. CartContext
5. CartWidget
6. Formulario de compra

*Entrega final*

* **Formato:** Link a repositorio de git donde se cumplan los objetivos + GIF/Video mostrando la navegabilidad y los flujos básicos de la aplicación y los que el alumno/a considere importantes a remarcar.
* **Objetivo del desafío:** El estudiante deberá cumplir con todos las especificaciones del desafío
* **Condiciones**: El challenge se debe entregar **completo.** Todo lo requerido en el documento del desafío es requerido (no es posible omitir ninguna de las funcionalidades básicas). Lo no requerido o los desafíos extra no serán considerados
* **Modificatorias:** Dado que las solicitudes tienen un sentido, las modificatorias funcionales son des-recomendadas y si se realizaran de todos modos, el docente será quien decidirá a propio y exclusivo criterio si dicha alteración y sus racionales se encuadran y tienen sentido con los criterios de aceptación de la rúbrica actual, pudiendo el mismo solicitar la corrección hacia los requisitos originales.

*RÚBRICAS DE EVALUACIÓN*

*ENTREGA DEL PROYECTO FINAL*

| **Proyecto Final** | | | |
| --- | --- | --- | --- |
| Criterios | **Bajo**  Falta más profundización. Es confuso. | **Correcto**  Acorde pero con errores puntuales. | **Óptimo**  Es claro y pertinente. |
| Presentación | * No genera docs o tiene docs limitados a dependencias | * Genera docs con dependencias, y alguna breve explicación * Borra el markdown estándar e indicios del script automático de react | * Sobresalta la calidad de redacción * Genera recursos de levantado de proyecto o remarks a considerar * Incluye racionales y consideraciones del producto * Reemplaza otros assets de fábrica generados por CRA |
| Convenciones | * No respeta camelCase para eventos * No respeta PascalCase para Componentes * No mantiene consistencia en el scaffolding (pone context en un component scope o al revés) * Publica node\_modules | * Respeta algunas convenciones pero aunque sea posible detectar algunos buenos usos, todavía no lo hace de manera consistente * Mantiene todos los componentes en distintos directorios con sus respectivos styles (si lo hace distinto deja en claro su approach en el readme) * Puede tener algunos temas leves de consistencia * Puede dejar algunas piezas no utilizadas o comentarios extra | * Respeta todas las convenciones propuestas y lo hace en todas las instancias según lo propuesto * Es consistente con sus declaraciones del readme * Trata de afiliarse a las convenciones del ecosistema de React * Presenta un linting consistente con espaciados que favorezcan la lectura en todo el proyecto * No se deja comentarios, debuggers, o TODO’s olvidados * No deja piezas de código inutilizadas, no referenciadas o sin sentido |
| Container patterns | * No realiza distinciones / sub-clasificaciones y desarrolla obtención y renderings en el mismo archivo | * Divide los patrones, no mezcla data-fetching / data-rendering. * Puede presentar fricciones en el pasaje de props * Puede no tener fallbacks o muestra el componente incompleto antes de tener la información (siempre y cuando no implemente skeletons) * Un re-fetch en el container actualiza el child render | * Agrupa múltiples props de manera entendible y consistente bajo la consideración de respetar sus entidades * Respeta namings a lo largo de la aplicación * No agrega fetches extra dentro de los dummy components / componentes presentacionales. |
| Simpleza | * Agrega librerías en exceso * Agrega librerías que no utiliza * Genera componentes en exceso | * Si bien le dió un uso a las librerías que importó, se podría haber resuelto simplemente sin hacerlo * No agrega features complejos no solicitados | * Sólo usa librerías que aportan valor no lograble de otro modo * Las dependencias incluidas son de proyectos con mantenimiento |
| Navegabilidad | * No puede acceder directamente a las rutas de categorías / item by id * Las navegaciones destruyen el estado del context * No implementa el browser router * Escribe en duro las categorías en los routes o repite links * No respeta el principio de que la home tiene el catálogo | * Logra navegar a cada ruta separada y logra navegabilidad de catálogo a detalle y viceversa sin reload de context o app * Ajusta correctamente los effects con los hooks de react router * Evita dismounts al saltar entre rutas similares item/1 -> item/2 los cuales efectúan re-fetches a firestore * No genera un componente custom para apuntar el categoryId | * Respeta las convenciones y los routings * Puede implementar patrones adicionales * Ejecuta navegaciones programáticas * Utiliza NavLinks para ubicar al usuario y les da un styling que valide que ya está en una ruta * Si ya se encuentra en una ruta y navega a la misma ruta no re-ejecuta llamados * Respeta las convenciones de naming * Separa el key / id de categoría de su respectiva descripción internacionalizada |
| Navbar | * No implementa navbar, o implementa las categorías escritas a mano cada una ó usa hrefs en vez de links | * Implementa el navbar y logra separar los datos del render en alguna variable estática para implementar un categories.map( c => c), pero puede olvidar de aplicar los Links del router | * Implementa los Links y logra navegaciones sin reload mientras usa una variable estática o una **promise** para resolver las categorías (detecta oportunidades basadas en otros conocimientos) |
| Catálogo | * No se visualiza el título o no muestra alguna referencia legible al producto | * Separa correctamente las responsabilidades e implementa las promises, pero todavía no hace un uso eficiente de las props * No filtra por categorías | * Filtra por categorías implementando correctamente el useParams(:catId), y aprovecha un useEeffect para detectar sus cambios de ruta y vuelve a disparar las tareas asincrónicas |
| Detalle | * No separa el detalle del container * Utiliza where() para obtener un único documento * No funciona el counter, no respeta los límites o no emite el evento a su parent onAdd * Faltan más de la mitad de los elementos visuales requeridos | * Separa correctamente las responsabilidades e implementa las promises, pero todavía no hace un uso eficiente de las props * Utiliza .doc() para realizar el fetching de un único documento * Genera algunas optimizaciones tempranas (negativo) * Puede faltar algún dato (usar solo descripcion o solo titulo, ú olvidarse el precio, pero no todos) | * Separa correctamente las responsabilidades * Toda la información solicitada se muestra * Usa loaders * No sobre-optimiza y se mantiene simple |
| CartProvider / CartContext | * Realiza cálculos de precio / stock / otros, fuera del contexto o en componentes, evitando la centralización de la fuente de verdad * Trata de llevar parte del flow de checkout o lógicas del formulario al context * Mueve cosas al Context que no son necesarias o le remueven el sentido de la responsabilidad * Asigna al cart context responsabilidades de fetching de datos | * Lleva las rutinas de cálculos de costos y precio dentro del context * Puede tratar de guardar el precio y las cantidades en algún estado interno del context aunque no es necesario * No entremezcla conceptos con el Context aunque tampoco aprovecha su potencial * Puede presentar alguna fricción menor / sobre-complejización de los procesos de add / remove / clear | * Mantiene las responsabilidades limpias * No lleva cosas de UI al context * No genera subestados para guardar totalizaciones o counts / summary * Puede crear otras variaciones de las sub-funcionalidades y adicionar funciones simples que mantienen la consistencia con el resto |
| CartWidget | * No incluye widget o realiza un widget que no está conectado al contexto * El widget presenta problemas de sincronización | * El widget sincroniza las cantidades correctas * El widget está conectado y sincronizado con los eventos de add en catálogo y “remover” desde el cart * El widget permite navegar al cart | * No hay lógicas de cálculo de cantidades dentro del widget * Puede separar el widget del rendering del icon |
| Cart | * Dificultades de división de responsabilidades: hace todo en un mismo componente (form, escritura en firestore y display) * Si borra todos los items le queda el formulario activo o no brinda feedback volver al catálogo por la carencia de artículos | * Divide responsabilidades a través de eventos / props y múltiples componentes (elementos agregados, form, botón) * Puede presentar algunas redundancias o guardados de estados innecesarios * Puede generar algunos cálculos que deberían ir dentro cart dentro de Cart.js | * Aparte de dividir responsabilidades es capaz de desarrollar un flow claro y reactivo de deshabilitación de campos * Valida correctamente los inputs y sus tipos de datos * No utiliza recursos como alert o nativos del browser manteniendo el feedback en el componente * Después de realizar la compra deja el cart y otros componentes en un estado consistente |