*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 INTERMEDIA DEL PROYECTO FINAL*

**Componentes:**

1. Navbar con cart
2. Catálogo
3. Detalle de producto

*Catálogo, detalle y navegabilidad*

* **Formato:** Link a último commit de git donde se cumplan los objetivos + GIF mostrando la navegabilidad
* **Objetivo del desafío:** El estudiante deberá desarrollar la navegabilidad básica de la aplicación, demostrando que la app permite ver el catálogo, y navegar a un detalle

**Incluir:**

* **Readme.md:** El archivo debe estar en el root del proyecto para dar una breve introducción acerca de su proyecto y qué ideas o enfoque eligió para el mismo. Si incluyó dependencias extra por npm (por fuera de las trabajadas en clase), aparte debe hacer un resumen explicando sus decisiones

*RÚBRICAS DE EVALUACIÓN*

*ENTREGA INTERMEDIA DE PROYECTO FINAL*

| **Desafío: Navegabilidad / Componentes** | | | |
| --- | --- | --- | --- |
| Criterios | **Bajo**  Falta más profundización. Es confuso. | **Correcto**  Acorde pero con errores puntuales. | **Óptimo**  Es claro y pertinente. |
| Convenciones | * No respeta camelCase para eventos * No respeta PascalCase para Componentes | * Respeta algunas convenciones pero aunque sea posible detectar algunos buenos usos, todavía no lo hace de manera consistente | * Respeta las convenciones propuestas y lo hace en todas las instancias según lo propuesto |
| 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 logra navegar entre vistas pero las rutas separadas son accesible | * Logra navegar a cada ruta separada y logra navegabilidad de catálogo a detalle y viceversa sin reload | * Logra la navegabilidad completa y usa efectos para evitar desmontar componentes y re-carga al cambiar de categoría o itemId |
| Cmp: 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 | * No olvida implementar 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) |
| Cmp: Catálogo | * No separa el itemlist del container, dividiendo las responsabilidades de la view y de los datos * No implementa promises | * 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 promises |
| Cmp: Detalle | * No separa el detalle del container * No implementa promises * 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 * Genera 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 |