ANEXO · TAREAS DEL ALUMNADO

TAURI: EMPAQUETAR TU APP

Nombre: _____ Fecha: _____

TAREA 1 · ENVUELVE TU APP (17 MIN)

Individual, en **tu propio proyecto**, en una rama nueva de git llamada `tauri`. **Tauri** pone tu frontend web en una ventana nativa: la dibuja la WebView que ya tiene el sistema operativo y un pequeño programa en Rust la abre. Sigue los seis pasos:

OK	PASO	EXACTAMENTE
☐	**1** · Instala las herramientas.	`npm install -D @sveltejs/adapter-static @tauri-apps/cli` y después `npm install @tauri-apps/api`
☐	**2** · Sin servidor en una ventana: exporta archivos estáticos.	Donde el proyecto diga `adapter-auto` (`svelte.config.js`, o `vite.config.js` en plantillas recientes): importa `@sveltejs/adapter-static` y usa `adapter({ fallback: 'index.html' })`
☐	**3** · Renderiza en la WebView, no en un servidor.	Archivo nuevo `src/routes/+layout.js` con una línea: `export const ssr = false;`
☐	**4** · Un atajo para la CLI.	En `package.json` → `scripts`: `"tauri": "tauri"`
☐	**5** · Crea la mitad nativa.	`npm run tauri init` → nombre de la app, título de la ventana, frontend dist `../build`, dev URL `http://localhost:5173`, dev command `npm run dev`, build command `npm run build`
☐	**6** · Ejecútala en su propia ventana.	`npm run tauri dev`

El primer `tauri dev` compila unos 400 paquetes de Rust (5–10 minutos). En cuanto veas pasar `Compiling...`, empieza la tarea 2.

Si algo falla:

- **Tu app usa el servidor** (`+page.server.js`, una form action o una ruta `+server.js`): no puede existir en una exportación estática. Mueve la carga de datos a `+page.js` o a un `fetch` dentro del componente. Si es un cambio grande, hoy envuelve NoteBox y migra en casa.
- **Puerto ocupado.** Tienes un `npm run dev` olvidado en otra terminal. Ciérralo: Tauri arranca el suyo.
- **`tauri: not found`.** Falta el paso 4, o no ejecutaste `npm install` después del paso 1.
- **Ventana en blanco.** `frontendDist` no es `../build`, o falta `fallback: 'index.html'`.
- **La cámara o el micrófono de la UT5 no hacen nada.** La WebView de Linux no ofrece todo lo que ofrece Chrome. Anótalo en el README como limitación conocida y sigue.

TAREA 2 · ANATOMÍA, MIENTRAS COMPILA (5 MIN)

En parejas. Duplica el archivo de Figma de Classroom y arrastra las seis etiquetas amarillas a las seis casillas del tablero **1 · Anatomy of a Tauri app**. Escribe aquí qué etiqueta va en cada una:

#	LA CASILLA DICE…	ETIQUETA
1	Lo que escribiste en UT3–UT7; después de `npm run build`, HTML, CSS y JS normales.	
2	El motor de navegador que dibuja tu UI; no va dentro de la app.	
3	Un pequeño programa nativo: abre la ventana, puede tocar archivos.	
4	El puente: JavaScript llama a una función de Rust por su nombre y espera la respuesta.	
5	Un archivo con el nombre, la versión, el identificador, los iconos y qué empaquetar.	
6	La herramienta que produce .exe, .msi, .deb, .rpm, .AppImage, .dmg.	

Cuando se abra tu ventana: ¿tu app se ve exactamente igual que en Chrome? Anota dos diferencias.

TAREA 3 · ¿QUÉ VA EN LA CAJA? (5 MIN)

En parejas, tablero **2 · What goes in the box?**: arrastra las 12 cartas a **In the installer** o **Not in the installer**. Marca aquí tu decisión.

CARTA	DENTRO	FUERA	CARTA	DENTRO	FUERA
`node_modules/`	☐	☐	`build/` (HTML, CSS, JS)	☐	☐
Node.js y npm	☐	☐	Los iconos	☐	☐
`src/routes/+page.svelte`	☐	☐	El binario Rust compilado	☐	☐
Rust y cargo	☐	☐	`resources/welcome-notes.json`	☐	☐
Las stories de Storybook	☐	☐	Un archivo de texto LICENSE	☐	☐
WebView2	☐	☐	Las notas del propio usuario	☐	☐

TAREA 4 · LA ETIQUETA DE LA MALETA

En el `src-tauri/tauri.conf.json` de tu app:

OK	HAZ	ESTÁ HECHO CUANDO…
☐	Cambia `identifier` por uno tuyo en forma de dominio al revés, por ejemplo `com.iessonferrer.notebox` (solo letras, dígitos, guiones y puntos).	Ya no pone `com.tauri.dev` (si lo dejas, `tauri build` no arranca).
☐	Pon `version` a `0.1.0`.	Aparece en el nombre del paquete.
☐	Si tu app lee archivos extra, decláralos en `bundle > resources` (rutas relativas a `src-tauri/`).	La app instalada los encuentra, no solo `tauri dev`.

TAREA 5 · ICONO DE LA APP (13 MIN)

OK	HAZ	ESTÁ HECHO CUANDO…
☐	Tablero **3 · App icon**: dibuja el icono de **tu** app en el frame de 1024 × 1024, con el motivo dentro del área segura discontinua. Una forma, dos o tres colores planos, sin texto, transparente fuera de la baldosa.	Se entiende en las vistas previas de 128, 32 y 16 px: tu compañero sabe qué es a 16 px.
☐	Borra el cuadrado discontinuo → selecciona el frame → **Export** → PNG, 1x → guárdalo como `app-icon.png` en la raíz del proyecto.	Existe `./app-icon.png`.
☐	Ejecuta `npm run tauri icon`.	`src-tauri/icons/` tiene `32x32.png`, `128x128.png`, `icon.icns`, `icon.ico`…

Si a 16 px no se entiende, quita detalle; no lo añadas.

TAREA 6 · PRIMER PAQUETE (9 MIN)

OK	HAZ	ESTÁ HECHO CUANDO…
☐	Ejecuta `npm run tauri build -- --bundles deb` (en Windows: `--bundles nsis`). Compila todo otra vez: 5–10 minutos.	Las últimas líneas imprimen la ruta: `src-tauri/target/release/bundle/deb/`.
☐	Copia la ruta y el tamaño del paquete.	Los tienes en la entrega.

ENTREGA

En la tarea de Classroom: (1) el enlace a tu copia del archivo de Figma con los tres tableros hechos; (2) una captura de tu app en su propia ventana, con tu icono; (3) la ruta y el tamaño del paquete, copiados del terminal. Si el build no ha terminado, pega la ruta mañana por la mañana.

PARA CASA

- Termina de envolver tu app si una función de servidor te bloqueó. El paquete tiene que compilar.
- Si puedes, instala el paquete en tu ordenador (doble clic en el `.deb`, o `sudo apt install ./nombre.deb`) y anota una cosa que te sorprenda. La usamos mañana.
- Ejercicio de transferencia (un párrafo): una app muestra sus páginas de ayuda desde archivos Markdown en `static/help/`, y su núcleo Rust lee un `defaults.json` que está junto al código fuente. Tras instalarla, la ayuda funciona pero faltan los valores por defecto. Explica por qué uno viajó y el otro no, y la línea que lo arregla.