lib.rs — Application Bootstrap

`lib.rs` is where the entire Tauri application is configured and started. It does the following in order:

Managed state registration

Three pieces of state are registered with Tauri's state manager so they can be injected into any command via `State<'_>`:

- `RoverState` — the three rover mode booleans (`drive_manual_mode`, `arm_manual_mode`, `pickup_mode`), all wrapped in `Mutex` so they are safe to read and write from async commands
- `DummyStreamHandle` — holds an optional cancellation flag for the dummy simulator
- `RoverAddress` — holds the port the Rover is sending to

> `RoverState` is subject to change, because the rover might become able to drive and move the arm at the same time

Plugin registration

Three official Tauri plugins are loaded:

Plugin	Purpose
`tauri-plugin-fs`	File system access from the frontend
`tauri-plugin-opener`	Open files/URLs in the OS default application
`tauri-plugin-dialog`	Native file picker and dialog boxes

All plugins must be registered in `lib.rs`, `Cargo.toml` and if they require access to anything in `src-tauri/capabilities/default.json`

> AI will often try to get you to add them to `tauri.conf.json` but that is, as far as I have encountered if, incorrect

Command registration

All Tauri commands are registered here via `tauri::generate_handler!`. This is the complete list of commands callable from the frontend via `invoke()`. If you add a new command in any `commands/` file, it must also be added here or it will not be accessible from the frontend.

Setup (startup sequence)

The `.setup()` closure runs once at launch, before any window is shown. It performs these steps in order:

a) GStreamer plugin path Sets the `GST_PLUGIN_PATH` environment variable so GStreamer can find its plugins on Windows. It will look for them at `C:\gstreamer\1.0\msvc_x86_64\bin`. On Linux it can find the plugins automatically.

b) Storage directory creation Calls `ensure_storage_dirs_internal()` to create the `tasks/`, `images/`, and `maps/` subdirectories under the app data directory if they don't already exist.

c) Cache clearing Calls `clear_cache_on_startup()` to wipe any stale cached files from the previous session.

d) GStreamer streaming server Spawns an async task that runs `commands::gstreamer::stream()` for the lifetime of the app. This starts the three GStreamer pipelines and their corresponding MJPEG HTTP servers.

e) UDP service Creates `UdpService` (binding `0.0.0.0:9000`) synchronously using `block_on`. The socket is extracted before the service is moved into Tauri's state manager, so it can be passed to the listener independently.

f) UDP listener Spawns an async task running `network::listener::run_listener()` with the shared socket. This is the loop that receives, decodes, and forwards all incoming rover packets to the frontend.

g) Controller listener Calls `commands::controller::start_controller_listener()`, which spawns an OS thread to poll for gamepad events.

Revision #8
Created 2026-04-14 13:18:19 UTC by Candela Cimadevilla Gonzalez
Updated 2026-05-05 09:38:10 UTC by Candela Cimadevilla Gonzalez