Frontend — Components

Location: src/lib/components/

Components are reusable UI building blocks used across multiple routes. Each is a self-contained Svelte file. They communicate with the backend via Tauri invoke() and receive live rover data via Tauri events listened to with listener().

- video.svelte + double_video.svelte — Video Components
- model_scene.svelte + model_viewer.svelte + model_debug.ts — 3D Model Viewer
- task_completion.svelte — Task Completion
- map.svelte — Map
- costmap.svelte — Costmap
- imu.svelte — IMU
- sampling_locations.svelte + SampleField.svelte — Sampling Locations
- interest_locations.svelte — Interest Locations
- navigation_plan.svelte — Navigation Plan
- probes.svelte — Probes
- maintenance_tasks.svelte — Maintenance Panel
- map.svelte + navigation_plan.svelte + interest_locations.svelte — Map & Navigation Components

video.svelte + double_video.svelte — Video Components

video.svelte

> **TODO:** still being developed, going to change

The basic single-camera display component. Accepts a camera object from `state.svelte.js` and renders it as an `<img>` tag pointing at the MJPEG stream URL. Supports two optional modes passed as the `mode` prop:

- **`measure` mode** — enables pixel-clicking for stereo measurement. The operator clicks two points across two camera feeds; the component calls `invoke("request_measurement")` with the pixel coordinates from both cameras and returns the result via an `onmeasurement` callback.
- **`pick` mode** — enables a pick-up interaction for probe/rock collection. The `pick()` function is a stub ready for the actual rover arm command to be wired in.

When no mode is set the component is a plain passive video display.

double_video.svelte

Displays two camera feeds simultaneously with a picture-in-picture layout. The primary feed fills the frame; the secondary feed appears as a smaller overlay in the bottom-right corner. Clicking the overlay swaps the two feeds, making the secondary feed the primary and vice versa.

Both feeds show a `⚠ SIGNAL LOST` overlay banner when their `stale` flag is `true` (set by the camera health listener in `state.svelte.js`). The secondary feed shows a smaller version of the same warning.

Props: `camera1`, `camera2` — camera objects from `state.svelte.js`.

model_scene.svelte + model_viewer.svelte + model_debug.ts — 3D Model Viewer

These three files together form the 3D model display system used on the dashboard.

model_scene.svelte

The outer container and lifecycle manager. Handles: delayed initialisation (100ms timer to ensure the DOM is ready before the WebGL canvas is created), WebGL context loss detection, error state with a retry button, and a resize nudge that forces the canvas to reflow correctly after mount.

Uses `model_debug.ts` to persist error state across the component lifecycle using `localStorage` — if the model failed to load in the previous render, the error state is restored immediately on mount to avoid a flash of broken content.

model_viewer.svelte

The inner Three.js / Threlte scene. Loads the GLB model file by calling `invoke("load_model")`, which returns the raw bytes of the file. The bytes are parsed directly in the browser using `GLTFLoader.parse()` — no HTTP request is made.

After loading, all mesh materials are replaced with a uniform `MeshStandardMaterial` in the Roboteam's purple brand colour (`#5A1C74`). The camera is automatically fitted to the model's bounding box so any model file will be centred and fill the view regardless of its original scale.

The model auto-rotates slowly using `OrbitControls` with `autoRotate`. User rotation, zoom, and pan are disabled — the view is fixed. A scale animation eases the model from 0 to 1 on load using Threlte's `useTask`.

model_debug.ts

A small utility module with two functions: `setLoadFailed(bool)` and `wasLoadFailed(): bool`. These read and write a `localStorage` key to persist the model error state across component re-renders.

task_completion.svelte — Task Completion

Displays the history of completed tasks read from the `tasks/` app data directory. On mount it lists all task files and deserialises each JSON file into a `Task` object.

Task list

Each task is shown as a card with its name, number, completion time, and finish timestamp. Clicking a card opens a detail modal. A delete button removes the task file and all associated sample images.

Task detail modal

Shows full task metadata and a list of all attached `Sample` objects with their location, coordinates, measurement, weight, and image paths. Image paths are clickable links that open the image viewer modal.

Image viewer modal

Loads the before and after sample images using `appDataDir()` + `convertFileSrc()` and displays them side by side.

map.svelte — Map

> **TODO:** We still don't have the map format, function is subject to change

Displays a static map that the operator imports. The map is stored in the `maps/` app data directory and loaded using Tauri's asset protocol (`convertFileSrc`).

Map selection flow

On mount, the component checks the `displayedMap` store. If a map is already selected (from a previous navigation within the session) it loads it directly. If not, it lists available map files and presents a selection UI. If exactly one map file exists it is auto-selected and confirmed without operator interaction.

Once a map is confirmed, the full path is constructed using `appDataDir()` and converted to a Tauri asset URL for display. The selected map is written to the `displayedMap` store so other components (and other routes) can access it.

A reload button (⟳) resets the selection and re-lists available files. Mouse coordinates over the map are tracked and displayed, laying the groundwork for click-based waypoint placement.

costmap.svelte — Costmap

TODO: A placeholder component that renders a "Costmap" heading. Intended to display the rover's navigation cost map (obstacle/traversability grid) received from the rover. Not yet implemented.

imu.svelte — IMU

Displays live inertial measurement unit data received from the `imu-update` Tauri event.

Data displayed

Accelerometer — X/Y/Z values in m/s² with a live scrolling sparkline chart showing the last 60 samples per axis. Each axis has a distinct colour (red, purple, green).

Gyroscope — X/Y/Z values in °/s with the same sparkline treatment.

Orientation cube — a CSS 3D cube whose `rotateX/Y/Z` transform is driven by integrating the gyroscope values over time, giving a visual indication of the rover's pitch, roll, and yaw. Euler angles are displayed numerically next to the cube.

Compass — a Canvas-drawn compass rose with tick marks, cardinal labels, and a red needle pointing in the direction derived from the magnetometer X/Y values. The needle and labels adapt to light/dark colour scheme.

Status bar — shows calibration status (✓ Cal / ! Uncal), sensor state (Idle / Operating / Calibrating / Error), any active error code, and the current update rate in Hz.

Performance

Incoming events are batched using `requestAnimationFrame` — a `pending` buffer holds the latest payload and the render only runs on the next animation frame, so high-frequency updates (up to 50Hz) never block the UI thread. The Hz counter counts packets per second independently of renders.

sampling_locations.svelte + SampleField.svelte — Sampling Locations

The main data collection interface for the Science task. Manages a list of `Sample` objects stored in the `samples` Svelte store.

Sample card

Each sample in the list is rendered as a card with an editable location name field and a set of `SampleField` sub-components. The location name field updates `location_name_check` automatically as the operator types.

SampleField.svelte

A reusable row sub-component used inside each sample card. Renders a checkbox (bound to a `checked` prop), a label, the current value, and a `+` button that opens the relevant modal. Used for: Coordinates, Size, Weight, Image Before, Image After.

Modals

Clicking a `+` button opens a modal specific to that field type:

Coordinates — a single button that calls `invoke("request_coordinates")`, receives a `[lat, lon]` tuple from the rover, formats it as a string, and saves it to the sample.

Measurement — shows two camera feeds (arm + front) in `measure` mode. The operator clicks two corresponding points on the two feeds to trigger a stereo measurement via `invoke("request_measurement")`. The returned value is saved to the sample.

Weight — a single button that calls `invoke("request_weight")` and saves the returned gram value to the sample.

Image Before / Image After — shows all three camera feeds. Clicking any feed calls `invoke("save_snapshot")` which captures a JPEG frame from that stream and saves it to the `images/` directory. The filename is `{sample.label}_{before|after}`.

Pick up Rock

> This section is work in progress

A separate overlay accessible from a button at the bottom of the component. Shows all three camera feeds in `pick` mode. The `pick()` function is currently a stub for the rover arm pick-up command.

interest_locations.svelte — Interest Locations

TODO: A placeholder component that renders a "Locations of Interest" heading. Intended to display GPS-tagged points of interest identified during the probing task. Not yet implemented.

navigation_plan.svelte — Navigation Plan

A drag-and-drop ordered list of navigation waypoints using `svelte-dnd-action`. Reads from and writes to the `waypoints`, `startPoint`, and `endPoint` stores in `stores/map.ts`.

The list always starts with a fixed **Starting Point** card and ends with a fixed **End Point** card. Between them the operator can add intermediate waypoints and reorder them by dragging. Each waypoint has a delete button.

The **Add Map File** button opens a native file picker (filtered to JSON, GeoJSON, TXT, JPEG) and calls `invoke("import_map_file")` to copy the selected file into the app's `maps/` directory, where the map component can then find it.

> The **Plan Route** button is a stub ready to be connected to actual route planning logic.

probes.svelte — Probes

Displays the list of probes from the `probes` store. Currently each probe renders as a basic card. A "Pick up probe" button opens an overlay showing all three camera feeds in `pick` mode, using the same pattern as the pick-up overlay in `sampling_locations.svelte`.

> **TODO:** The `pick()` function is a stub.

maintenance_tasks.svelte — Maintenance Panel

TODO: WIP

map.svelte + navigation_plan.svelte + interest_locations.svelte — Map & Navigation Components

map.svelte

The core map display component. Loads a map file from `<appDataDir>/maps/`, renders it in a letterboxed `<img>` element, and overlays interactive pins on top. Accepts a `mode` prop that controls what happens when the operator clicks the map.

Mode	Click behaviour	Pins shown
`navigation`	Adds a `PinnedCoord` to the `pinnedCoords` store	Unassigned pins + start/waypoint/end markers
`science`	Adds an `InterestLocation` to `scienceLocations`	Science location pins
`probing`	Adds an `InterestLocation` to `probingLocations`	Probing location pins

Map loading — On mount, if `displayedMap` store already holds a filename it is opened immediately; otherwise the component calls `list_task_files("maps")` and shows a selection modal. If only one map file exists it is auto-selected. The reload button (⟳) clears all state and returns to the selection modal.

3D formats (`.obj`, `.las`, `.laz`, `.e57`) trigger a `render_map` invoke before display, showing a spinner while the backend works. Plain image formats are loaded directly via `convertFileSrc`. After rendering, the `_preview.png` path is used for all subsequent display.

Coordinate geometry — `getRenderedRect()` computes the actual rendered rectangle of the image inside its element (accounting for letterboxing / `object-fit` behaviour). `eventToImgPixel()` converts a raw `MouseEvent` into a pixel coordinate within the PNG, with Y flipped so the origin is bottom-left. `worldToCSSPos()` is the inverse — takes a world-space `(x, y)` in metres and returns a `left/top` percentage suitable for absolute positioning an overlay element on top of the image. Both functions short-circuit to `null` when `mapMeta` is unavailable.

Mouse interaction — `onMouseMove` calls `eventToImgPixel` and then invokes `pixel_to_world` to display a live coordinate overlay in the bottom-left corner (pixel position + world metres + "Click to pin"

hint). The overlay disappears when the cursor leaves the image.

GPS marker — Listens for `gps-update` Tauri events on mount. The payload's `longitude`/`latitude` fields are reused directly as map-space X/Y metres. When a valid GPS position is in the store, a directional arrow marker (▶) is rendered at the corresponding map position, rotated by `heading` via a CSS custom property `--heading`.

Pinned coordinate list — In `navigation` mode, a sidebar panel lists all `pinnedCoords` with copy-to-clipboard (□□) and remove (×) buttons. Hovering a row highlights the corresponding pin on the map, and vice versa, via `hoveredPinId`.

navigation_plan.svelte

A sidebar panel for building a navigation route from map-pinned coordinates. Displays a structured plan of start point → ordered waypoints → end point, and surfaces any `pinnedCoords` that haven't yet been assigned a role.

Route structure — The plan always shows three sections in order: a Start card, a draggable Waypoints list, and an End card. Unset slots show "Not set". Hovering any card cross-highlights its corresponding pin on the map via the `hoveredNavId` store (shared with `map.svelte`).

Waypoint reordering — The waypoint list uses `svelte-dnd-action` (`dndzone`) for drag-and-drop reordering. Both `consider` and `finalize` events write the new order directly to the `waypoints` store.

Promoting pinned coords — Each `PinnedCoord` from the map appears in a "Pinned from map" section at the bottom of the list. Three buttons let the operator promote a pin to Start, add it as a new Waypoint, or set it as End. In all cases the pin is removed from `pinnedCoords` after promotion.

Destructive actions — Removing a waypoint and clearing the entire plan both trigger a native `confirm()` dialog before proceeding.

Map import — The "+ Add Map File" button opens a file picker (via `@tauri-apps/plugin-dialog`) filtered to `.json`, `.geojson`, `.txt`, `.jpeg`, `.obj`, `.las`, `.laz`, `.e57`, then calls `invoke("import_map_file")` to copy the chosen file into the app's maps directory. The "▶ Plan Route" button is present but not yet wired to a backend command.

interest_locations.svelte

A generic, reusable sidebar list for named locations of interest. Used by both the Science and Probing task panels, which pass in their respective stores (`scienceLocations` / `probingLocations` and the matching hovered-id store) as props.

Props: `locations` — a `Writable<InterestLocation[]>` store; `hoveredId` — a `Writable<string | null>` store shared with `map.svelte` for cross-highlighting.

Each location row shows its auto-generated name and its `(x, y)` coordinates in metres. Hovering a row sets `hoveredId`, which causes the corresponding pin on the map to highlight. Two actions are available per row:

- **Rename (✏)** — Switches the name label to an inline `<input>`. The edit is committed on blur or Enter; if the input is left empty the original name is kept. Only one location can be in edit mode at a time (`editingId` state).
- **Remove (×)** — Removes the location from the store immediately with no confirmation.

When the list is empty a hint instructs the operator to click the map to add a location.