**RENDERER.js**

**BLOK 0:** Inicializace proměnných (řádky 1–9)

console.log('ANCHORS:', ANCHORS);

let incidentLog = [];

let incidents = [];

let prevInRed = false;

let playbackSpeed = 1;

let timer = null;

let followBall = false;

Účel:

* Inicializace globálních stavových proměnných používaných v animaci a detekci incidentů.
* incidentLog: pro horní box se statistikami.
* incidents: pro exportní nebo podrobnější záznamy (např. do souboru).
* prevInRed: značí, zda marker byl v zakázané zóně (pomáhá detekovat vstupy/výstupy).
* playbackSpeed: rychlost přehrávání animace.
* timer: identifikátor pro setInterval.
* followBall: určuje, zda má mapa sledovat marker.

**BLOK 1:** Funkce **updateIncidentBoxes()** (řádky 11–27)

function updateIncidentBoxes() {

const countEl = document.getElementById('incident-summary-count');

const listEl = document.getElementById('incident-summary-list');

if (countEl && listEl) {

countEl.textContent = incidentLog.length;

listEl.innerHTML = '';

incidentLog.forEach(inc => {

const inT = new Date(inc.inDate).toLocaleTimeString();

const outT = inc.outDate

? new Date(inc.outDate).toLocaleTimeString()

: '<em>aktivní</em>';

const li = document.createElement('li');

li.innerHTML = `<strong>IN:</strong> ${inT}&nbsp;<strong>OUT:</strong> ${outT}&nbsp;<strong>Doba:</strong> ${inc.duration}s`;

listEl.appendChild(li);

});

}

}

Účel:

* Aktualizuje horní panel incidentů na dashboardu.
* Vypisuje počet incidentů (incidentLog.length).
* Vytváří <li> prvky s časem vstupu a výstupu a délkou trvání incidentu.

**BLOK 2:** Funkce haversine() (řádky 29–36)

function haversine(lat1, lon1, lat2, lon2) {

const toRad = deg => deg \* Math.PI / 180;

const R = 6371000;

const dLat = toRad(lat2 - lat1);

const dLon = toRad(lon2 - lon1);

const a = Math.sin(dLat/2) \*\* 2 +

Math.cos(toRad(lat1)) \* Math.cos(toRad(lat2)) \*

Math.sin(dLon/2) \*\* 2;

return R \* 2 \* Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

}

Účel:

* Výpočet vzdálenosti mezi dvěma GPS body pomocí Haversinovy formule.
* Používá se pro výpočet délky incidentu (např. vzdálenost, kterou objekt urazil uvnitř polygonu).

**BLOK 3:** Globální konstanty a nastavení (řádky 38–42)

const DEVICE = 'GH5200';

const SUBJECT\_ID = 'CEPRO0516';

const MAX\_LOGS = 5;

const TIME\_STEP = 100; // 100 ms

Účel:

* Identifikace zařízení a subjektu (může být použito pro pojmenování výstupních souborů).
* MAX\_LOGS: maximální počet záznamů pro log nebo náhled.
* TIME\_STEP: interval přehrávání animace (v milisekundách).

**BLOK 4:** Definice polygonů (řádky 44 a dále)

const smallPoly = turf.polygon([[ [15.0755, 50.0439], ..., [15.0755, 50.0439] ]]);

const segA\_poly = turf.polygon([...]);

const segB\_poly = turf.polygon([...]);

// atd.

Účel:

* Definují oblasti, ve kterých sledujeme vstup/výstup GPS bodů (např. zakázané zóny, segmenty trasy apod.).
* Používají se pro incident detekci s pomocí knihovny turf.js.

**Načítání reálných dat ze souboru (RENDERERDATA.json) pomocí fetch()**

❗Blok nenalezen ve formátu fetch('RENDERERDATA.json') ani RENDERERDATA v analyzovaném kódu.

➤ Možnosti**:**

* Kód pro fetch() může být dynamicky vkládán nebo zapsán jinak než fetch("RENDERERDATA.json").
* Možná používáš jiný způsob importu – ANO.data přímo (např. input souboru přes uživatelské rozhraní).

Na základě analýzy mezi řádky 396–500 zde přináším detailní popis bloku animace markeru na mapě, který využívá data ze souboru RENDERERDATA1.js:

**🟦** BLOK: Animace markeru na mapě podle ***RENDERERDATA1***

📌 Umístění:

* Začátek: Řádek 396 (načtení dat z ***RENDERERDATA1***)
* Konec: cca Řádek 499+ (konec hlavního těla setInterval, pokračuje i dále)

**🔧** Účel:

Tento blok animuje pohyb markeru (kuličky) podle záznamu GPS dat (lat, lng, timestamp) z proměnné RENDERERDATA1, která je načtena přímo ze skriptu (bez fetch). Pro každý snímek animace:

* posune index idx vpřed/zpět podle rychlosti (playbackSpeed),
* aktualizuje pozici markeru na mapě,
* detekuje, zda došlo k incidentu (vstup do polygonu),
* vypočítává rychlost a typ pohybu (např. stání, běh, sprint),
* podle potřeby posouvá mapu (map.panTo).

| **Fáze** | **Řádky** | **Popis** |
| --- | --- | --- |
| **Načtení dat** | 396–411 | Validuje, filtruje a transformuje RENDERERDATA1 na pole data s objekty obsahujícími lat, lng, timestamp a turf.point. |
| **Inicializace** | 433–434 | let idx = 0; – výchozí index pro přehrávání. |
| **Animace (setInterval)** | 435–499+ | Hlavní smyčka každých 100 ms (nebo TIME\_STEP), postupně posouvá index, získává data a aktualizuje mapu a stav. |
| ➤ **Rychlost a typ pohybu** | 448–458 | Využívá Haversin a časový rozdíl k odhadu rychlosti a určuje typ pohybu (např. „pomalá chůze“, „běh“). |
| ➤ **Posun markeru** | 460 | marker.setLatLng([lat, lng]) – aktualizace pozice kuličky. |
| ➤ **Změna barev kotev** | 463 | Volá updateAnchorColors() pro zabarvení bodů na mapě. |
| ➤ **Detekce incidentů** | 466 | Volá checkIncidents() s aktuálním bodem. |
| ➤ **Detekce zóny** | 469–475 | Pomocí turf.booleanPointInPolygon() kontroluje, zda se bod nachází v zelené (bezpečné) nebo červené (nebezpečné) zóně. |
| ➤ **Přesun mapy (panování)** | 478–480 | map.panTo(...) – automaticky sleduje marker při animaci. |

**Typický výsledek:**

* Animovaná kulička se pohybuje po mapě podle záznamů.
* Zaznamenávají se incidenty (např. vstup do červené zóny).
* Mění se rychlost a texty na obrazovce.
* Data lze exportovat nebo zobrazit ve výpisu.

Pokud chceš, mohu teď podrobně popsat i:

* funkci checkIncidents(),
* nebo updateAnchorColors(),
* případně vytvořit kuchařku k tomu, jak změnit tento blok, např. pro přidání pauzy, omezení na konkrétní úseky, apod.

**🟦 2. Animace markeru na mapě (L.marker)**

❗ **Nebyly nalezeny žádné řádky s L.marker nebo L.Marker** – je možné, že marker je vytvořen nepřímo, nebo je obalen ve funkci.

📌 Doporučuji prohledat funkci animace (pravděpodobně definována jako startAnimation() nebo animate() apod.). Mohu jej vyhledat ručně, pokud potvrdíš.

**🟦 3. Detekce vstupu do polygonů pomocí turf.booleanPointInPolygon()**

📍 **Výskyt**: Řádky **242–246, 249** a **488–494**

**🔧 Účel:**

* Kontrola, zda se GPS bod nachází uvnitř polygonu (např. zakázaná zóna).
* Používá se k detekci začátku a konce incidentu.

**🔗 Propojeno s:**

* Incident logging (blok 4)
* Polygonové konstanty (smallPoly, segA\_poly atd.)
* Funkce animace (která periodicky testuje pozici)

**🟦 4. Záznam incidentů (incidentLog, incidents)**

📍 **Výskyt**: Významné řádky: **236, 254, 260, 312, 377, 381, 552, 571, 574, 676, 678, 682, 691, 695, 699, 713**

**🔧 Účel:**

* incidentLog – seznam pro **zobrazení** ve webovém UI.
* incidents – seznam pro **export** dat.
* Zapisují se při změně stavu (vstup/výstup do polygonu).

**🔗 Propojeno s:**

* Funkcí detekce polygonu (booleanPointInPolygon)
* DOM aktualizací boxů (funkce updateIncidentBoxes())
* Funkce exportIncidents() nebo downloadJSON().

**🟦 5. Funkce pro export JSON a manipulaci s DOM (tlačítka, logy)**

📍 **Výskyt**: Roztroušeno mezi řádky **23, 220, 269, 294, 308, 312, 318, 324, 382, 629, 662, 682, 684, 687, 699, 701, 704**

**🔧 Účel:**

* Tlačítka pro ovládání animace, exportu logů.
* Vytváření a klikání na „neviditelný“ <a download> element pro export JSON.
* Manipulace s DOM (přidání záznamů do seznamu, změna textů v boxech).

**🔗 Propojeno s:**

* incidentLog / incidents
* updateIncidentBoxes()
* Záznam a export dat (stažení .json, CSV, apod.)

| **Blok** | **Řádky** | **Účel** | **Navazuje na** |
| --- | --- | --- | --- |
| Polygon detekce | 242–246, 488–494 | Zjištění, zda GPS bod je v polygonu | turf.js, GPS data, incidentLog |
| Incident logika | 236–713 (roztroušené) | Vytváření a správa záznamů | Polygon check, DOM, export |
| DOM & export | 23, 220–704 (roztroušené) | Zobrazení a export dat | incidentLog, tlačítka |
| Animace a fetch | ❓ nutno dohledat | Načítání dat a pohyb markeru | GPS data, turf, DOM |

**1. Funkce checkIncidents()**

📍 **Začíná na řádku: 239**

**🔧 Účel:**

* Detekuje, zda GPS bod vstoupil do „nebezpečné zóny“ (červený polygon).
* Sleduje, kdy marker vstoupil a kdy vystoupil.
* Vytváří incidenty s časem, pozicí a trváním.

**🔗 Propojeno s:**

* incidentLog, incidents
* booleanPointInPolygon() (pro polygon test)
* Hlavní smyčkou animace (setInterval)
* Panelem se záznamy (updateIncidentBoxes())

**🟦 2. Funkce updateAnchorColors()**

📍 **Začíná na řádku: 421**

**🔧 Účel:**

* Aktualizuje barvu nebo vizuální reprezentaci „kotev“ na mapě v závislosti na blízkosti GPS pozice.
* Slouží k vizuální orientaci – např. zvýraznění nejbližšího bodu nebo aktivního úseku.

**🔗 Propojeno s:**

* Funkcí animace (každý snímek volá updateAnchorColors(current position))

**🟦 3. document.getElementById('map-wrapper')?.appendChild(ballInfoPanel)**

📍 **Řádek: 719**

**🔧 Účel:**

* Přidává prvek ballInfoPanel (pravděpodobně informační box) přímo do mapového kontejneru.
* Použití ?. (optional chaining) zajišťuje, že se kód nesesype, pokud element neexistuje.

**❗ Proč je to důležité:**

* Umožňuje **připojení dynamického UI** (např. popupu, tooltipu) do mapy bez reloadu.
* Zvyšuje **modularitu** – mapa funguje i bez tohoto boxu (např. při testování).

**🟦 4. window.addEventListener("DOMContentLoaded", () => { ... })**

📍 **Začíná na řádku: 619**

**🔧 Účel:**

* Tento blok zajišťuje, že veškerý DOM je **plně načtený**, než se k němu přistupuje pomocí getElementById().
* Uvnitř tohoto bloku jsou **navázány všechny posluchače (eventy)** a spuštěna inicializace (např. loadAndStartAnimation()).

**❗ Proč je to důležité:**

* Bez tohoto by document.getElementById(...) vracel null, pokud by se skript spustil před načtením HTML.
* Je to standardní, bezpečný způsob **navazování interaktivních prvků na HTML**.

**🟦 5. Tlačítka pro ovládání přehrávání**

📍 **Řádky: 621–625**

document.getElementById("startBtn")?.addEventListener("click", () => { playbackSpeed = 1; updateSpeed(); });

document.getElementById("pauseBtn")?.addEventListener("click", () => { playbackSpeed = 0; updateSpeed(); });

document.getElementById("fasterBtn")?.addEventListener("click", () => { playbackSpeed = 2; updateSpeed(); });

document.getElementById("slowerBtn")?.addEventListener("click", () => { playbackSpeed = -1; updateSpeed(); });

document.getElementById("stopBtn")?.addEventListener("click", () => { location.reload(); });

**Účel:**

* **Start**: nastaví přehrávání na rychlost 1.
* **Pause**: zastaví přehrávání (0).
* **Faster / Slower**: změní rychlost.
* **Stop**: restartuje aplikaci (reload stránky).

**🔗 Propojeno s:**

* Globální proměnnou playbackSpeed (řídí směr a tempo animace).
* Funkcí updateSpeed() (pravděpodobně aktualizuje stav UI).
* Animací (setInterval), která každých X ms kontroluje playbackSpeed.

**❗ Proč je to důležité:**

* **Uživatelská kontrola** animace v reálném čase.
* Nezávislost na backendu – všechno ovládání probíhá na klientovi.
* ?.addEventListener zajistí bezpečnost i pokud tlačítko neexistuje (např. při testování).

| **Blok/Funkce** | **Kde** | **Propojeno s** | **Funkce** |
| --- | --- | --- | --- |
| checkIncidents() | ř. 239 | booleanPointInPolygon, incidentLog | Detekce vstupu do polygonů |
| updateAnchorColors() | ř. 421 | marker animace | Vizualizace blízkosti kotev |
| DOM loaded + append panel | ř. 619–719 | všechny UI elementy | Inicializace po načtení |
| Tlačítka (start/pause/...) | ř. 621–625 | playbackSpeed, updateSpeed() | Ovládání přehrávání |

celá HTML struktura dokumentu byla prohlížečem načtena a sestavena do tzv. Document Object Modelu (DOM stromu), a tedy je připravená k manipulaci pomocí JavaScriptu.

🔍 Co je DOM?

* DOM je interní reprezentace HTML dokumentu v paměti prohlížeče.
* HTML prvky (např. <div>, <button>, <ul>, <svg>) jsou převedeny na uzly stromu.
* JavaScript pak přes document.getElementById() nebo querySelector() přistupuje právě k těmto uzlům.

📌 Co tedy znamená „DOM je plně načtený“?

✅ V praxi to znamená:

* Všechny elementy zapsané v HTML kódu (nebo načtené do <body>) už existují v paměti.
* Můžeš je bezpečně hledat, měnit jejich vlastnosti, přidávat event listenery (např. click, hover).
* Např. toto už je bezpečné:

document.getElementById("startBtn").addEventListener("click", startPlayback);

❌ Pokud bys volal tento řádek předčasně, tedy před načtením celého <body>, getElementById() by vrátil null a následná práce s tímto null by vyvolala chybu.

🧠 Proč se používá DOMContentLoaded?

window.addEventListener("DOMContentLoaded", () => {

// V tuto chvíli je DOM připraven

});

| Stav | Kdy nastává | Popis |
| --- | --- | --- |
| DOMContentLoaded | Jakmile je HTML načteno a zparsováno | Obrázky a externí styly ještě nemusí být hotové |
| load | Když je celá stránka včetně obrázků načtena | Zpomalené, vhodné pro poslední kroky |
| defer | Pokud JS soubor má defer, spustí se automaticky po načtení DOM | Moderní způsob bez ručního DOMContentLoaded |

„DOM je plně načtený“ znamená:

* HTML bylo převedeno do objektové struktury.
* JavaScript už může bezpečně manipulovat s prvky.
* Používá se DOMContentLoaded, defer, nebo vložení <script> na konec <body>.

DOM (Document Object Model) je strukturovaná reprezentace index.html v paměti prohlížeče.

🔍 Co to znamená prakticky?

* Prohlížeč vezme obsah souboru index.html (nebo jakéhokoliv HTML dokumentu),
* zpracuje všechny značky jako <div>, <h1>, <button>, <svg>, <script> atd.,
* a vytvoří tzv. DOM strom – stromovou datovou strukturu, kde každý prvek je objekt a má vlastnosti.

**INDEX.HTML vs RENDERER.JS**

**způsob, jak minimalizovat závislost na složitém index.html (např. v AdminLTE)**

Pokud chceš:

* **co nejméně sahat** do index.html (a jeho variant index2.html, index3.html),
* **zachovat jednoduchost a přehlednost**,
* ale **zachovat dynamické prvky a mapu, tabulky, přepínače, grafy apod.**,

pak doporučuji některou z těchto architektur:

**✅ 1. Vlastní samostatný HTML soubor pro konkrétní komponentu (např. zonovano.html)**

To už správně děláš – a je to **velmi doporučené** řešení.

**Výhody:**

* Nezávislost na složité šabloně (index.html).
* Můžeš načítat jen to, co opravdu potřebuješ (např. mapu, tlačítka).
* Přehlednost – vše co vykresluješ, máš pod kontrolou.

**Jak na to:**

* Mít v zonovano.html jen základní strukturu (mapa, panel, tlačítka).
* Používat komponenty z AdminLTE jen tam, kde dávají smysl (např. card, box, info-box).

**2. Dynamické vkládání obsahu přes JavaScript (tzv. komponenty)**

Místo toho, abys měl všechno v index.html, můžeš například:

const wrapper = document.getElementById("map-wrapper");

const btn = document.createElement("button");

btn.textContent = "Start";

btn.classList.add("btn", "btn-primary");

btn.onclick = () => startPlayback();

wrapper.appendChild(btn);

**Používání querySelector() jen v rámci vlastních zón**

Pokud máš AdminLTE dashboard s více panely (index.html, index2.html, ...), můžeš psát JavaScript takto:

const context = document.querySelector("#my-zone-panel");

const btn = context.querySelector(".export-button");

| **Příznak** | **Doporučená cesta** |
| --- | --- |
| Velké množství šablon (index\*.html) | Používej vlastní HTML (zonovano.html) |
| Dynamicky generované komponenty | Vytvářej elementy v JS (appendChild) |
| Potřebuješ čisté testovací prostředí | Drž se jednoho lehkého HTML |