# Інтерфейс для користувача

Команда Гамма

# 1. Встановлення і користування

## 1.1 Встановлення

Для роботи програми потрібна одна папка – gamma. Її можна розмістити в будь-якій директорії, головне – не змінювати розташування файлів всередині папки.

Інтерфейс оформлений у вигляді HTML сторінки, тому має запускатися будь-якому комп'ютері. На мобільних пристроях не вийде використати цю програму, адже там зовсім інший протокол для передачі даних через USB порт
(детальніше про це: https://stackoverflow.com/questions/11011515/how-to-read-and-write-data-to-com-serial-ports-in-android).

## 1.2 Робота з програмою

Щоб запустити програму, потрібно відкрити в браузері файл index.html. Тоді підключити приймач через USB COM порт до комп'ютера. В програмі потрібно вибрати потрібне підключення. Після активації відкривається мапа.

Коли приходять координати від приймача, на мапі з'являється точка. Коли прийде наступна координата, попередня точка зникне і з'явиться нова; між ними буде намальована лінія.

Якщо користувач активував сигнал SOS, спливає сповіщення в нижньому правому куті і змінюється іконка маркера. Натиснувши на нього, мапа відцентрується на тих координатах, де останній раз був переданий сигнал SOS.

При будь-якому збої потрібно перезавантажити сторінку.

## 2. Архітектура програми

Вся програма розміщення в папці **gamma**.

**index.html** – головний файл, веб-сторінка.

**scripts** – файли JavaScript, в яких виконуються основні алгоритми

script.js – головний файл скриптів, відповідає за всі алгоритми на сторінці

leaflet.js – системний файл мапи; дуже небажано змінювати. Зазвичай, його підключають в файлі index.html з посиланням на сервер, але в нашому випадку його довелося завантажити та підключити локально.

**styles** – папка стилів сторінки

leaflet.css – системний файл мапи; дуже небажано змінювати. Аналогічно до leaflet.js

style.scss – всі стилі елементів сторінки. При компіляції з формату SCSS в стандартний CSS утворюється два файли: style.css та style.css.map

**tiles** – тут зберігаються зображення мапи для різних рівнів приближення. Вони використовуються в файлі script.js, де, зазвичай, йде посилання на сервер. Але у нас воно перероблено під локальні потреби.

**img** – тут зберігаються маркери пристрою, які відображаються на мапі.

**user-data** – папка для майбутніх напрацювань, тут будуть записуватися дані користувача

**zones.json** – файл, в якому зберігаються координати меж геозон.

## 3. Код

### 3.1 index.html

Основний файл, який задає розмітку сторінки. Тут ми її налаштовуємо і підключаємо всі необхідні файли.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    Назва вкладки
    <title>Координати</title>

    Підключення стилів
    <link rel="stylesheet" href="styles/style.css">

    Підключення системних стилів мапи
    <link href="styles/leaflet.css" rel="stylesheet"/>

    Підключення іконки вкладки
    <link rel="shortcut icon" href="img/tourist_icon.png" type="image/x-icon">

</head>

<body>

    Екран на всю сторінку, який підключає пристрій
    <div id="portConnectScreen">
        <h1>Натисніть, щоб підключити пристрій</h1>
    </div>

    Мапа
    <div id="osm-map"></div>

    Сповіщення, яке потім з’явиться. Воно змінює текст кожен раз
    <p id="alert">Тестове сповіщення</p>

    Підключення системних скриптів мапи
    <script src="scripts/leaflet.js"></script>

    Підключення наших скриптів
    <script src="scripts/script.js"></script>

</body>
</html>
```

3.2 script.js

Головний скрипт сторінки index.html – задає всі алгоритми.

```
Якщо пристрій має надто малі розміри (менше 300 пікселів), то видаємо помиклу:
if (window.matchMedia("(min-width: 300px)").matches) {
    connectDevice()

} else {
    let errorText = document.createElement("p");
    errorText.innerHTML = "Window size error - open page on wider device."
    errorText.style = "text-align: center; color: black; margin: 0; padding: 0;
width: 70%;"
    document.body.style = "width: 100vw; height: 100vh; display: flex; align-items:
center; justify-content: center;";
    document.body.innerHTML = ""
    document.body.append(errorText);
}

Основний колір, який ми всюди використовуємо. Винесли його у змінну, щоб потім було
легше змінити
var primeColor = "#aa0000";

Ці змінні будуть потрібні потім, але задати їх варто на найвищому рівні програми
var lastMarker;
var lastMarkerObject;
var lastCOMPortValue;

Максимальний зум на мапі (потрібно, щоб не вибивало помилку, коли не знаходить
потрібний зум в папці tiles).
var maxAllowedZoom = 17;

Підключення мапи (в т.ч. посилання на tiles) і центрування її на якихось координатах
var map = L.map(document.getElementById("osm-map"), { maxZoom: maxAllowedZoom });
L.tileLayer("tiles/{z}/{x}/{y}.png").addTo(map);
map.setView([49.834956, 24.014456], 14);

Задаємо вигляд іконок: звичайна та сигнал SOS. Тут вказується посилання на картинки,
які лежать в папці img. Вони в форматі SVG (векторний формат), тому потрібно
використовувати відповідний редактор, якщо ви забажаєте змінити.
var markerIcon = L.icon({
    iconUrl: "img/marker.svg",
    iconSize: [25, 25],
    iconAnchor: [12.5, 12.5],
});
var alertIcon = L.icon({
    iconUrl: "img/alert_marker.svg",
    iconSize: [25, 25],
    iconAnchor: [12.5, 12.5],
```

```
});
var currentIcon = markerIcon;

Підключення пристрою
function connectDevice() {
    Додаємо для нашого екрану подію, щоб, коли на нього натиснуть, спрацював код нижче
    document.getElementById("portConnectScreen").addEventListener("click", async() => {
        Підключення пристрою
        const port = await navigator.serial.requestPort();
        await port.open({ baudRate: 9600 });
        document.getElementById("portConnectScreen").style = "top: 100%;";
        console.log("Port connected");

        Викликаємо функцію, яка дістає з файлу координати геозон
        getUserData()

        Скрипт, який розшифровує і записує у змінну decodedValue інформацію, яка прийшла з порта
        const reader = port.readable.getReader();
        while (true) {
            const { value, done } = await reader.read();
            if (done) {
                reader.releaseLock();
                break;
            }

            const decoder = new TextDecoder();
            const decodedValue = decoder.decode(value);

            Виокремлюємо потрібні нам координати через спеціальну функцію (нижче)
            var x = sliceDataString(decodedValue)[0];
            var y = sliceDataString(decodedValue)[1];
            var currentTime = sliceDataString(decodedValue)[2];
            var SOS = sliceDataString(decodedValue)[3];

            Робимо перевірку, чи координати нормального формату. Якщо все добре, малюємо точку. Якщо ні – повторна перевірка, але з об’єднанням попереднього пакету з порта (часто буває, що інформація автоматично розбивається на кілька пакетів: зазвичай, два). Якщо і в цей раз не пройшла перевірка, тоді просто ігноруємо.
            if (checkCoordinatesFormat(x, y)) {
                callSOS(SOS, currentTime, [x, y]);
                drawNewPoint(x, y, currentTime, SOS);
            } else {
                var twoCOMPortPacks = lastCOMPortValue + decodedValue;
                var x = sliceDataString(twoCOMPortPacks)[0];
                var y = sliceDataString(twoCOMPortPacks)[1];
```

```
                var currentTime = sliceDataString(twoCOMPortPacks)[2];
                var SOS = sliceDataString(twoCOMPortPacks)[3];

                if (checkCoordinatesFormat(x, y) && decodedValue != "\n") {
                    console.log(`Joined packs ${lastCOMPortValue} +
${decodedValue}`);
                    callSOS(SOS, currentTime, [x, y]);
                    drawNewPoint(x, y, currentTime, SOS);
                } else {
                    console.log(`Ignored data: ${decodedValue}`);
                }
            }

            lastCOMPortValue = decodedValue;
        }
    })
}

Перевірка формату координат
function checkCoordinatesFormat(x, y) {
    if (x.length == 9 && /^\d+$/.test(x * Math.pow(10, 6)) &&
        y.length == 9 && /^\d+$/.test(y * Math.pow(10, 6))) {
        return true;
    }
}

Функція отримання реального часу в потрібно нам форматі
function getCurrentTime() {
    var currentDate = new Date();
    return currentDate.getHours() +
        ":" +
        currentDate.getMinutes() +
        ":" +
        currentDate.getSeconds() +
        " " +
        currentDate.getDate() +
        "." +
        (currentDate.getMonth() + 1) +
        "." +
        currentDate.getFullYear();
}

Витягаємо зі стрічки, яка прийшла з порта, потрібну нам інформацію
function sliceDataString(str) {
    let x = str.slice(0, 9);
    let y = str.slice(10, 19);
    let SOS = str.slice(20, 21);
    let currentTime = getCurrentTime();
```

```
    return [x, y, currentTime, SOS];
}

Виведення сигналу SOS
function callSOS(SOS, time, coords) {
    if (SOS == 1) {
        Якщо є SOS
        console.log(`SOS: ${time}`)
        Виводимо сповіщення
        document.getElementById("alert").style.bottom = "15px";
        document.getElementById("alert").innerHTML = `SOS!`;
        Центруємо мапу
        document.getElementById("alert").addEventListener("click", () => {
            map.flyTo(coords, maxAllowedZoom);
        })
        І міняємо маркер
        currentIcon = alertIcon;
    }
    if (SOS == 0) {
        А якщо нема SOS, то ховаємо іконку
        document.getElementById("alert").style.bottom = "-100%";
        currentIcon = markerIcon;
    }

}

Малювання маркеру на мапі
function drawNewPoint(x, y, currentTime, SOS) {
    Для розробника – виводимо координати в консоль
    console.log(`%c${currentTime} ${x}; ${y} - ${SOS}`, 'background: #900000; color:
#fff');

    Малюємо новий маркер
    var newMarker = L.latLng(x, y);

    Малювання лінії між двома точками
    if (lastMarkerObject != undefined && lastMarker != undefined) {
        var markerLine = new L.Polyline([newMarker, lastMarker], {
            color: primeColor,
            weight: 3,
            opacity: 0.8,
            smoothFactor: 1,
        });
        markerLine.addTo(map);

        Видаляємо стару іконку
        map.removeLayer(lastMarkerObject);
    }
```

```
    Додаємо новий маркер
    var newMarkerObject = L.marker(newMarker, { icon: currentIcon });
    newMarkerObject.addTo(map);

    lastMarker = newMarker;
    lastMarkerObject = newMarkerObject;
}

Функція, яка витягає з файлу координати геозон
async function getUserData() {
    const url = "../user-data/zones.json";
    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error(`Response status: ${response.status}`);
        }

        const json = await response.json();
        for (line in json) {
            Одразу малюємо ці зони – викликаємо функцію
            drawPolygon(json[`${line}`])
        }

    } catch (error) {
        console.error(error.message);
    }
}

Малювання геозон
function drawPolygon(polyCoords) {
    var polydata = [];
    for (i in polyCoords) {
        polydata.push(polyCoords[`${i}`]);
    }
    L.polygon(polydata, { color: primeColor }).addTo(map);
}
```

В цьому коді геозони мають суто декоративний вигляд. В майбутньому потрібно зробити функціонал, який додасть до них події.

### 3.3 style.scss

Файл стилів немає сенсу пояснювати, адже там лише декорації сторінки.

# 4. Додаткові ресурси

## 4.1 Leaflet

Мапа, яка використовується в програмі, створена на основі системи leaflet. Вона дозволяє малювати маркери на мапі по координатах та багато іншого.
Детальніше за посиланням: https://leafletjs.com/

## 4.2 Maperative

Щоб використовувати мапи оф-лайн, потрібно їх експортувати. На мою думку, найзручнішою програмою для цього буде Maperative.
Детальніше за посиланням: http://maperitive.net/