# ДОДАТОК А

## Приклад коду підключення до MongoDB

```
async def connect_to_db():
    global client, db
    try:
        client = AsyncIOMotorClient("mongodb://localhost:27017")
        db = client["drone_db"]
        print("✅ Подключение к MongoDB установлено")
        return db
    except Exception as e:
        print(f"❌ Ошибка подключения к MongoDB: {str(e)}")
        raise

async def close_db_connection():
    if client:
        client.close()
        print("❌ Подключение к MongoDB закрыто")

async def get_drones_collection():
    if db is None:
        await connect_to_db()
    return db["drones"]

async def get_users_collection():
    if db is None:
        await connect_to_db()
    return db["users"]

async def record_flight_history(drone_id: str, position: Optional[dict], status:
str):
    collection = await get_drones_collection()
    await db["flight_history"].insert_one({
        "drone_id": drone_id,
        "position": position,
        "status": status,
        "timestamp": datetime.utcnow()
    })
```

# ДОДАТОК Б

## Приклад WebSocket взаємодії

```
async def handle_websocket(websocket: WebSocket):
    await websocket.accept()
    drone_manager.clients.append(websocket)
    logger.info("Новий WebSocket клієнт підключено")

    try:
        while True:
            data = await websocket.receive_text()

            try:
                msg = json.loads(data)
                logger.debug(f"📥 WebSocket повідомлення: {json.dumps(msg,
indent=2)}")

                if msg["type"] == "get_drones":
                    await websocket.send_json({
                        "type": "drones_update",
                        "drones": list(drone_manager.drones.values())
                    })

                elif msg["type"] == "add_drone":
                    drone = msg.get("drone", {})
                    drone_data = {
                        "start": drone.get("start",
drone_manager.base_location),
                        "end": drone.get("end", drone_manager.base_location),
                        "addressStart": drone.get("addressStart", "База"),
                        "addressEnd": drone.get("addressEnd", "База"),
                        "useAI": drone.get("useAI", True),
                        "path": drone.get("path")
                    }

                    drone_id = await drone_manager.add_drone(drone_data)
                    await broadcast_drones()
                    await websocket.send_json({
                        "status": "success",
                        "drone_id": drone_id,
                        "message": "Дрон успішно додано"
                    })

                elif msg["type"] == "start_drone":
                    if await drone_manager.update_drone_status(msg["drone_id"],
"moving"):
                        await broadcast_drones()

                elif msg["type"] == "pause_drone":
                    if await drone_manager.update_drone_status(msg["drone_id"],
"paused"):
                        await broadcast_drones()

                elif msg["type"] == "remove_drone":
                    if await drone_manager.remove_drone(msg["drone_id"]):
                        await broadcast_drones()
```

```
                elif msg["type"] == "clear_drones":
                    await drone_manager.clear_drones()
                    await broadcast_drones()

                elif msg["type"] == "drone_completed":
                    await drone_manager.update_drone_status(msg["drone_id"],
"completed")
                    await broadcast_drones()

                elif msg["type"] == "chat_command":
                    user_input = msg.get("text", "").strip()
                    if not user_input:
                        raise ValueError("Порожня команда")

                    command = await ask_ollama(user_input)
                    commands_to_run = command.get("multiple_commands",
[command])
                    results = []

                    for cmd in commands_to_run:
                        action = cmd.get("action")
                        drones = cmd.get("drones", [])
                        target = cmd.get("target", "")
                        formation = cmd.get("type", "")

                        if action == "formation":
                            target_coords = await
drone_manager.geocode_address(target) if target else drone_manager.base_location
                            for i, _ in enumerate(drones):
                                offset = [
                                    math.cos(i * 2 * math.pi / 3) * 0.005,
                                    math.sin(i * 2 * math.pi / 3) * 0.005
                                ] if formation == "triangle" else [0, 0]

                                drone_data = {
                                    "start": [
                                        drone_manager.base_location[0] +
offset[0],
                                        drone_manager.base_location[1] +
offset[1]
                                    ],
                                    "end": target_coords,
                                    "addressStart": "База",
                                    "addressEnd": target or "База"
                                }

                                drone_id = await
drone_manager.add_drone(drone_data)
                                await
drone_manager.update_drone_status(drone_id, "moving")
                                results.append(drone_id)

                        elif action == "move":
                            target = cmd.get("target") or cmd.get("destination")
                            target_coords = await
drone_manager.geocode_address(target) if target else drone_manager.base_location

                            avoid_zones = cmd.get("avoid_zones", [])

                            if not drone_manager.drones:
```

```
                                drone_data = {
                                    "start": drone_manager.base_location,
                                    "end": target_coords,
                                    "addressStart": "База",
                                    "addressEnd": target or "База",
                                    "useAI": True,
                                    "avoid_zones": avoid_zones
                                }
                                drone_id = await
drone_manager.add_drone(drone_data)
                                await
drone_manager.update_drone_status(drone_id, "moving")
                                results.append(drone_id)
                            else:
                                for drone_id, drone in
drone_manager.drones.items():
                                    drone["end"] = target_coords
                                    drone["addressEnd"] = target
                                    drone["useAI"] = True


                                    new_path = await
drone_manager.generate_route(
                                        drone["current"],
                                        target_coords,
                                        zones=weather_state.weather_zones,
                                        avoid_zones=avoid_zones
                                    )

                                    drone["path"] = new_path
                                    await
drone_manager.update_drone_status(drone_id, "moving")
                                    results.append(drone_id)

                        await broadcast_drones()
                        await websocket.send_json({
                            "type": "chat_result",
                            "status": "success",
                            "message": f"Команди виконано. Оновлено дронів:
{len(results)}",
                            "drone_ids": results
                        })

                    else:
                        await websocket.send_json({
                            "status": "error",
                            "message": f"Невідомий тип повідомлення: {msg['type']}"
                        })

                except Exception as e:
                    logger.error(f"❌ Помилка обробки повідомлення: {e}")
                    await websocket.send_json({
                        "status": "error",
                        "message": str(e)
                    })

        except WebSocketDisconnect:
            logger.info("🔌 WebSocket відключено")
        finally:
            if websocket in drone_manager.clients:
                drone_manager.clients.remove(websocket)
```

# ДОДАТОК В

## ПРИКЛАД ПРОМТУ ДЛЯ LLM МОДЕЛІ

```
def optimize_waypoints_with_context(origin, destination, avoid_zones,
raw_waypoints):
    prompt = f"""
    Ти навігаційний агент для дронів.

    Побудуй оптимальний маршрут від точки {origin} до {destination}, уникаючи
погодні зони:
    {avoid_zones}.

    Поточний маршрут (Google): {raw_waypoints}

    Поверни лише список координат у форматі:
    [{{"lat": ..., "lng": ...}}, ...]

    ЖОДНА точка маршруту не повинна потрапляти всередину жодної з зон
avoid_zones:
    [{ "lat": ..., "lng": ..., "radius": ... }].

    Облітай зони, дотримуючись оптимальності маршруту, але з абсолютним
униканням.

    """

    try:
        loop = asyncio.get_event_loop()
    except RuntimeError:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)

    try:
        response = loop.run_until_complete(ask_ollama(prompt))
    except Exception as e:
        logging.warning(f"[👹] LLM помилка: {e}")
        return raw_waypoints

    # Очікуємо, що response вже буде у вигляді списку координат
    if isinstance(response, list) and all("lat" in wp and "lng" in wp for wp in
response):
        return response
    else:
        logging.warning(f"[👹] Невалідна відповідь від AI: {response}")
        return raw_waypoints

def filter_invalid_points(route_points, avoid_zones):
    valid = []
    for point in route_points:
        p = Point(point["lng"], point["lat"])
        in_zone = False
        for zone in avoid_zones:
            center = Point(zone["lng"], zone["lat"])
            radius_deg = zone["radius"] / 111000.0
            if p.distance(center) < radius_deg:
                in_zone = True
                break
```

```
            if not in_zone:
                valid.append(point)
        return valid

def build_route_with_waypoints(origin, destination, avoid_zones,
mistral_model=None):
        logger = logging.getLogger(__name__)
        base_url = "https://maps.googleapis.com/maps/api/directions/json"
        params = {
            "origin": f"{origin[0]},{origin[1]}",
            "destination": f"{destination[0]},{destination[1]}",
            "key": GOOGLE_MAPS_API_KEY,
            "mode": "driving"
        }

        # Отримуємо основний маршрут
        try:
            response = requests.get(base_url, params=params)
            data = response.json()

            if data.get("status") != "OK":
                raise ValueError(f"Google Directions API error:
{data.get('status')}")

            route = []
            for leg in data["routes"][0]["legs"]:
                for step in leg["steps"]:
                    end_loc = step.get("end_location", {})
                    if "lat" in end_loc and "lng" in end_loc:
                        route.append((end_loc["lat"], end_loc["lng"]))
                    else:
                        logger.warning(f"[⚠] Невірний формат end_location:
{end_loc}")
        except Exception as e:
            logger.exception(f"[❌] Помилка під час запиту до Google Maps API:
{e}")
            return []

        # Якщо є обхідні зони, рахуємо точки обходу
        waypoints = compute_waypoints_to_avoid_zones(origin, destination,
avoid_zones)

        if waypoints:
            try:
                if mistral_model:
                    waypoints = mistral_model.optimize_waypoints(waypoints)

                logger.info(f"[🛰] Відправляємо оновлений маршрут з waypoint'ами:
{waypoints}")
                params["waypoints"] = '|'.join([f"via:{wp['lat']},{wp['lng']}" for
wp in waypoints])

                response = requests.get(base_url, params=params)
                data = response.json()
                if mistral_model and waypoints:
                    try:
                        optimized_waypoints =
mistral_model.optimize_waypoints_with_context(
                            origin=origin,
                            destination=destination,
                            avoid_zones=avoid_zones,
```

```
                        raw_waypoints=waypoints
                    )
                    waypoints = optimized_waypoints
                except Exception as e:
                    logger.warning(f"[🤘] Не вдалося оптимізувати через AI:
{e}")
            if data.get("status") != "OK":
                raise ValueError(f"Google Directions API error (updated):
{data.get('status')}")

            route = []
            for leg in data["routes"][0]["legs"]:
                for step in leg["steps"]:
                    end_loc = step.get("end_location", {})
                    if "lat" in end_loc and "lng" in end_loc:
                        route.append((end_loc["lat"], end_loc["lng"]))
        except Exception as e:
            logger.exception(f"[❌] Помилка обробки маршрутів з обхідними
точками: {e}")

    return [{"lat": lat, "lng": lng} for lat, lng in route]
```

# ДОДАТОК Г

## ПРИКЛАД КОДУ ВІЗУАЛІЗАЦІЇ МАПИ

```
const Map = ({ drones = [] }) => {
    const { isLoaded: mapLoaded, error: mapError } = useGoogleMaps(['places',
'drawing']);
    const [map, setMap] = useState(null);
    const [drawingManager, setDrawingManager] = useState(null);
    const [selectedWeather, setSelectedWeather] = useState("rain");
    const [weatherInfo, setWeatherInfo] = useState(null);
    const [weatherError, setWeatherError] = useState(null);
    const [weatherZones, setWeatherZones] = useState([]);
    const markersRef = useRef({});
    const routesRef = useRef({});
    const animationsRef = useRef({});
    const polylinesRef = useRef({});

    const animateDrone = useCallback((droneId, path) => {
        if (!path?.length || !map) return;

        if (animationsRef.current[droneId]) {
            cancelAnimationFrame(animationsRef.current[droneId].animationId);
        }

        if (!markersRef.current[droneId]) {
            markersRef.current[droneId] = new window.google.maps.Marker({
                position: path[0],
                map,
                icon: {
                    url: '/camera-drone.png',
                    scaledSize: new window.google.maps.Size(40, 40),
                },
                title: `Дрон ${droneId}`,
                optimized: false
            });
        }

        const duration = 15000;
        const startTimeRef = { value: null };

        const animateStep = (currentTime) => {
            if (!startTimeRef.value) {
                startTimeRef.value = currentTime;
            }

            const elapsed = currentTime - startTimeRef.value;
            const progress = Math.min(elapsed / duration, 1);

            const totalSegments = path.length - 1;
            const exactPosition = progress * totalSegments;
            const segmentIndex = Math.floor(exactPosition);
            const t = exactPosition - segmentIndex;

            if (segmentIndex < totalSegments) {
                const p1 = path[segmentIndex];
                const p2 = path[segmentIndex + 1];
```

```
                const interpolatedLat = p1.lat() + (p2.lat() - p1.lat()) * t;
                const interpolatedLng = p1.lng() + (p2.lng() - p1.lng()) * t;

                markersRef.current[droneId].setPosition(
                    new window.google.maps.LatLng(interpolatedLat,
interpolatedLng)
                );
            }

            if (progress < 1) {
                animationsRef.current[droneId] = {
                    animationId: requestAnimationFrame(animateStep)
                };
            } else {
                delete animationsRef.current[droneId];
                console.log("✅ Анимация завершена для дрона:", droneId);

                if (window.droneSocket && window.droneSocket.readyState ===
WebSocket.OPEN) {
                    console.log("💥 Дрон завершил маршрут, отправка на
сервер:", droneId);
                    window.droneSocket.send(
                        JSON.stringify({
                            type: 'drone_completed',
                            drone_id: droneId
                        })
                    );
                } else {
                    console.warn("⚠️ WebSocket не готов для отправки
drone_completed");
                }
            }
        };

        animationsRef.current[droneId] = {
            animationId: requestAnimationFrame(animateStep)
        };
    }, [map]);

    useEffect(() => {
        if (!mapLoaded) return;

        const mapInstance = new
window.google.maps.Map(document.getElementById('map'), {
            center: { lat: 50.4501, lng: 30.5234 },
            zoom: 13
        });
        setMap(mapInstance);

        return () => {
            Object.values(markersRef.current).forEach(marker =>
marker.setMap(null));
            Object.values(routesRef.current).forEach(route =>
route.setMap(null));
            Object.values(polylinesRef.current).forEach(polyline =>
polyline.setMap(null));
            Object.values(animationsRef.current).forEach(({ animationId }) =>
cancelAnimationFrame(animationId));
            weatherZones.forEach(circle => circle.setMap(null));
            if (drawingManager) {
```

```
                drawingManager.setMap(null);
            }
        };
    }, [mapLoaded]);

    useEffect(() => {
        if (!map || !mapLoaded) return;

        drones.forEach(drone => {
            if (!drone.path || drone.path.length === 0) return;

            const path = drone.path.map(p => new
window.google.maps.LatLng(p.lat, p.lng));

            if (!polylinesRef.current[drone.id]) {
                polylinesRef.current[drone.id] = new
window.google.maps.Polyline({
                    path,
                    map,
                    strokeColor: '#000000',
                    strokeOpacity: 0.8,
                    strokeWeight: 3,
                });
            }

            if (drone.status === 'moving' && !animationsRef.current[drone.id]) {
                animateDrone(drone.id, path);
            }
        });
    }, [drones, map, mapLoaded, animateDrone]);

    useEffect(() => {
        if (!map || !mapLoaded) return;

        const currentDroneIds = drones.map(d => d.id);

        Object.keys(markersRef.current).forEach(droneId => {
            if (!currentDroneIds.includes(droneId)) {
                markersRef.current[droneId]?.setMap(null);
                delete markersRef.current[droneId];

                routesRef.current[droneId]?.setMap(null);
                delete routesRef.current[droneId];

                polylinesRef.current[droneId]?.setMap(null);
                delete polylinesRef.current[droneId];

                if (animationsRef.current[droneId]) {

cancelAnimationFrame(animationsRef.current[droneId].animationId);
                    delete animationsRef.current[droneId];
                }
            }
        });

        drones.forEach(drone => {
            if (!drone.id || !drone.path || !drone.path.length) return;

            const path = drone.path.map(p => new
window.google.maps.LatLng(p.lat, p.lng));
```

```
            if (!polylinesRef.current[drone.id]) {
                polylinesRef.current[drone.id] = new
window.google.maps.Polyline({
                    path,
                    map,
                    strokeColor: drone.status === 'moving' ? '#00FF00' :
'#FF0000',
                    strokeOpacity: 0.8,
                    strokeWeight: 3,
                });
            }

            if (drone.status === 'moving' && !animationsRef.current[drone.id]) {
                animateDrone(drone.id, path);
            }
        });
    }, [drones, map, mapLoaded, animateDrone]);

    useEffect(() => {
        if (!map || !mapLoaded || drawingManager) return;

        const manager = new window.google.maps.drawing.DrawingManager({
            drawingMode: window.google.maps.drawing.OverlayType.CIRCLE,
            drawingControl: true,
            drawingControlOptions: {
                position: window.google.maps.ControlPosition.TOP_CENTER,
                drawingModes: ['circle'],
            },
            circleOptions: {
                fillColor: '#FF0000',
                fillOpacity: 0.35,
                strokeWeight: 2,
                clickable: false,
                editable: false,
                zIndex: 1,
            },
        });

        manager.setMap(map);
        setDrawingManager(manager);
```