



МІНІСТЕРСТВО  
ОСВІТИ І НАУКИ  
УКРАЇНИ



ХАРКІВСЬКИЙ  
НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ  
РАДІОЕЛЕКТРОНІКИ

# Програмна система для порятунку та реабілітації диких тварин з використанням геолокаційних сервісів та QR-кодів

Аранжий Руслан Володимирович, ПЗПІ-22-6  
Керівник: ст.викл. кафедри ПІ Олександр Олійник



15 червня 2025

# Мета роботи

Метою нашого комплексного курсового проєкту є створення програмного застосунку, який стане ефективним інструментом для організації процесу порятунку та реабілітації диких тварин.

Проблема збереження дикої природи з кожним роком стає дедалі гострішою. Причинами цього є як глобальні фактори — зміна клімату, руйнування природних середовищ, браконьєрство, — так і локальні, зокрема війна в Україні.

Ще одна проблема — це недостатня поінформованість громадян. Люди часто не знають, як правильно діяти, коли знаходять травмовану тварину, і це може призвести до додаткової шкоди.

Тому розробка такого застосунку є не просто актуальною, а необхідною. Він допоможе врятувати життя багатьом тваринам, поліпшити взаємодію між волонтерами та фахівцями, а також підвищити рівень екологічної свідомості в суспільстві.

# Аналіз проблеми (аналіз існуючих рішень)

Назва	Основний функціонал	Переваги	Недоліки
Домівка Врятованих Тварин	Допомога тваринам з притулків, фінансова підтримка, отримання оновлення та новин про тварин	Можливість допомогти тваринам, які знаходяться у притулках, інтеграція з платіжними сервісами, перегляд списку тварин та отримання оновлення про них	Немає можливості повідомити про знайдених тварин, відсутня інтеграція з соц. мережами, немає можливості перегляду детальної історії та інформації про тварин
Petsi.App	Допомога у пошуку загублених тварин	Можливість повідомлення про знайдену/загублену тварину, інтеграція з соціальними мережами	Немає можливості допомогти тваринам або волонтерам, немає перегляду детальної інформації про тварин
UaAnimals	Порятунок та допомога тваринам	Можливість повідомлення про тварин, можливість допомоги фінансово, інтеграція з соц. мережами, наявність звітності про тварин та фінансові витрати	Немає перегляду історії та детальної інформації про тварин, про тварин є можливість повідомити тільки через соціальні мережі

# Постановка задачі та опис системи

У сфері порятунку та реабілітації диких тварин відсутні централізовані цифрові інструменти для управління даними та координації дій

Розробка та впровадження спеціалізованого програмного застосунку має забезпечити:

- Централізовану базу даних для обліку врятованих тварин із фотознімками, описом стану, медичною історією та етапами реабілітації.
- Автоматизований моніторинг стану тварини, що дозволить фіксувати кожний етап лікування, зміни в стані, графік процедур та адаптацію в реабілітаційному центрі.
- Інструменти координації між учасниками процесу (ветеринари, волонтери), які спростять оперативний обмін повідомленнями.

# Вибір технологій розробки

У розробці програмного забезпечення було використано клієнт-серверну архітектуру, яка забезпечує чітке розділення клієнтської частини та серверної логіки. Для взаємодії між ними застосовується REST API поверх HTTP-протоколу, а формат обміну даними - JSON.

Мова програмування: Python, Фреймворк: FastAPI



База даних: MariaDB



Сховище файлів: MinIO (S3-сумісне об'єктне сховище)



Кешування: Redis



Веб-сервер: Nginx



Docker Compose



# Архітектура створенного програмного забезпечення

Архітектура програмного забезпечення побудована на модульному підході,

де кожен компонент виконує чітко визначену роль у загальній системі.

Database Client - Компонент, який забезпечує зв'язок із базою даних.

Storage - Файлове сховище (MinIO), у якому зберігаються медіафайли.

Auth - Модуль авторизації та автентифікації.

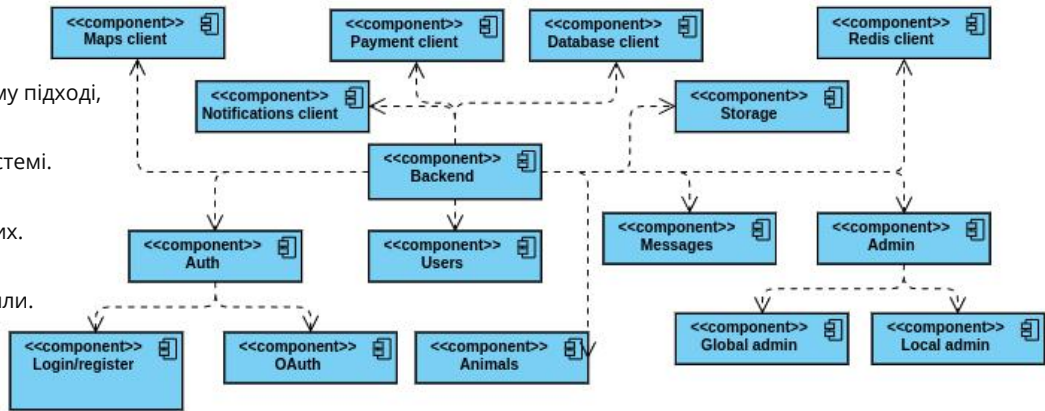
Users - Відповідає за управління профілями користувачів.

Redis Client - Використовується для кешування часто використовуваної інформації, такої як дані про тварин, користувачів.

Notifications Client - Компонент, що відповідає за відправлення push-сповіщень користувачам через FCM (Firebase Cloud Messaging).

Animals - Основний функціональний модуль, що охоплює створення, редагування, перегляд та пошук інформації про тварин.

Ця структура дозволяє легко масштабувати застосунок, додавати нові функції та підтримувати надійну взаємодію між усіма компонентами системи.



# Опис програмного забезпечення, що було використано у дослідженні

Уся розробка проходила у кілька основних етапів:

Аналіз вимог - було проведено дослідження предметної області, аналіз аналогів.

Проектування системи - розроблена архітектура клієнт-серверної моделі, побудована структура бази даних, визначені ключові компоненти та їх взаємодія.

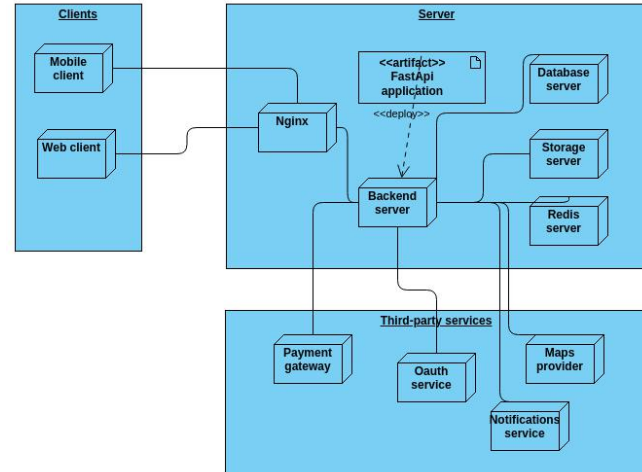
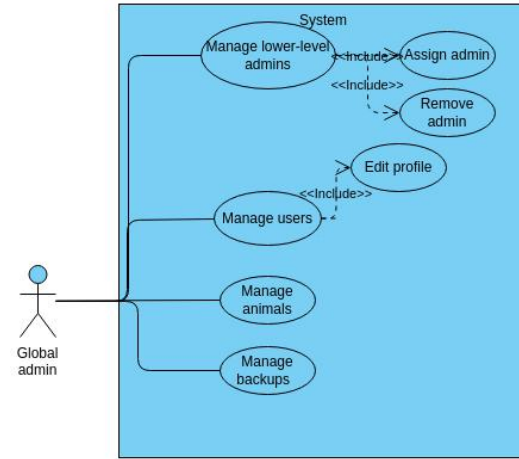
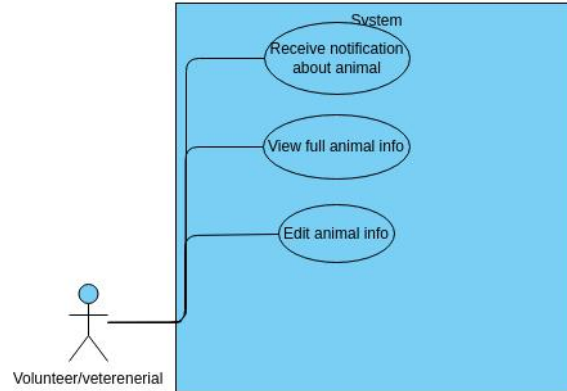
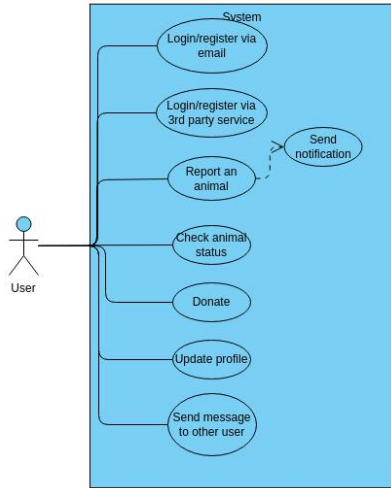
Реалізація функціоналу - покроково реалізовувалися основні модулі - авторизація, облік тварин, система сповіщень, інтеграція та платіжними платформами.

Тестування - модульне та інтеграційне тестування усіх компонентів.

Для створення ефективної, надійної та масштабованої системи було обрано сучасні технології:

Python - Основна мова програмування для бекенду. FastAPI - Легкий і високопродуктивний веб-фреймворк для побудови REST API. Забезпечує просту інтеграцію зі схемами OpenAPI, підтримку асинхронних запитів та автоматичну генерацію документації.

# Дизайн системы





# Дизайн системи

Під час проєктування системи було використано ряд сучасних підходів і методів:

Модульне проєктування - уся система поділена на модулі (автентифікація, користувачі, тварини, сповіщення тощо), що дозволяє легко масштабувати, оновлювати чи замінювати окремі частини без впливу на всю систему.

REST-архітектура - вся взаємодія між клієнтом та сервером організована у вигляді RESTful API — з чіткими HTTP-запитами до конкретних ресурсів (тварини, користувачі, пожертви тощо).

Модель "Клієнт-Сервер"

Архітектура базується на розділенні обов'язків між клієнтом (інтерфейс користувача) і сервером (логіка, обробка даних, зберігання інформації).

# Приклад реалізації

```
@router.post( path: "", response_model=AnimalReportInfo)
async def create_animal_report(user: JwtMaybeAuthUserDep, data: CreateAnimalReportsRequest, bg: BackgroundTasks):
    location = await GeoPoint.get_near(data.latitude, data.longitude)
    if location is None:
        location = await GeoPoint.create(name=None, latitude=data.latitude, longitude=data.longitude)

    async with in_transaction():
        animal_created = False
        if data.animal_id is not None:
            if (animal := await Animal.get_or_none(id=data.animal_id)) is None:
                raise CustomMessageException( messages: "Unknown animal!", status_code: 404)
            elif data.name is not None and data.breed is not None:
                animal = await Animal.create(
                    name=data.name, breed=data.breed, status=AnimalStatus.FOUND, current_location=location,
                )
                animal_created = True
            else:
                raise CustomMessageException( messages: "You need to specify either animal id or name and breed!", status_code: 400)

        report = await AnimalReport.create(reported_by=user, animal=animal, notes=data.notes, location=location)
        if data.media_ids:
            media = await Media.filter(id__in=data.media_ids, uploaded_by=user, status=MediaStatus.UPLOADED)
            await report.media.add(*media)
            await animal.medias.add(*media)
        if not animal_created:
            await Cache.delete_obj(animal)

        await AnimalUpdate.create(animal=animal, type=AnimalUpdateType.REPORT, animal_report=report)

    bg.add_task(_send_notification_task, *args: report)

    return await report.to_json()
```

# Приклад реалізації

```
async def _send_notification_task(report: AnimalReport) -> None:
    animal = report.animal
    location = report.location

    point = location.point
    point_wkb = point.to_sql_wkb_bin().hex()
    radius_m = 25000
    radius = radius_m / 111320
    before_time = int((datetime.now(UTC) - timedelta(days=14)).timestamp())

    sessions = await Session.raw(f"""
        SELECT
            'session'.nonce, 'session'.active, 'session'.user_id, 'session'.created_at, 'session'.location,
            'session'.location_time, 'session'.fcm_token, 'session'.id, 'session'.fcm_token_time,
            ST_Distance_Sphere('session'.location, x'{point_wkb}') 'dist'
        FROM 'session'
        LEFT OUTER JOIN 'user' 'session_user' ON 'session_user'.id='session'.user_id
        WHERE {mbr_contains_sql(point, radius, point_field: 'location')}
            AND 'session'.location_time > FROM_UNIXTIME({before_time})
            AND 'session'.fcm_token IS NOT NULL
            AND 'session_user'.role IN ({UserRole.VET.value}, {UserRole.VOLUNTEER.value})
            HAVING 'dist' < {radius_m}
    """)

    for session in sessions: # pragma: no cover
        try:
            await FCM.send_notification(
                "New animal needs your help!",
                f"Name: {animal.name}\nBreed: {animal.breed}\nNotes: {report.notes}",
                device_token=session.fcm_token,
            )
        except Exception as e:
            logger.opt(exception=e).warning(
                f"Failed to send notification to session {session.id} ({session.fcm_token!r})"
            )
```

# Приклад реалізації

```
@router.post(path="/google/mobile-callback", response_model=LoginResponse)
async def google_auth_mobile_callback(data: GoogleIdOAuthData):
    try:
        id_info = await verify_oauth2_token(data.id_token, config.oauth_google_client_id)
    except ValueError:
        raise CustomMessageException("Failed to verify token.")

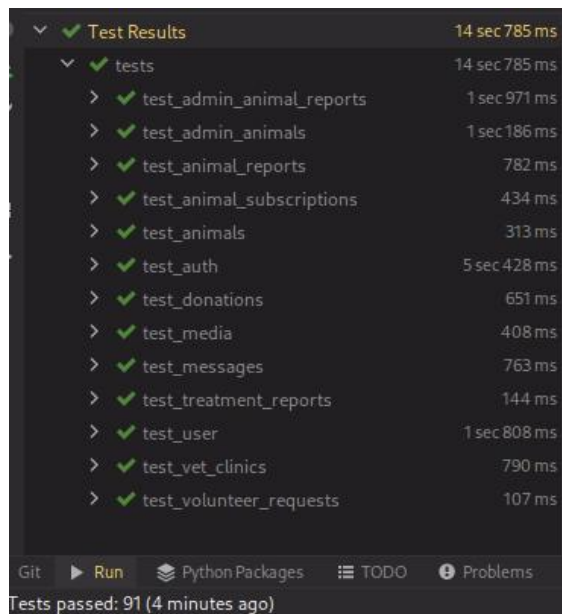
    existing_auth = await ExternalAuth.get_or_none(type=ExtAuthType.GOOGLE, external_id=id_info["sub"]).select_related("user")
    if existing_auth is not None:
        existing_auth.access_token = "id"
        existing_auth.refresh_token = "id"
        existing_auth.token_expires_at = 0
        await existing_auth.save(update_fields=["access_token", "refresh_token", "token_expires_at"])

    if existing_auth is None:
        user = await User.get_or_none(email=id_info["email"])
        if user is None:
            user = await User.create(
                email=id_info["email"],
                first_name=id_info["given_name"],
                last_name=id_info["family_name"],
            )
            await ExternalAuth.create(
                user=user,
                type=ExtAuthType.GOOGLE,
                external_id=id_info["sub"],
                access_token="id",
                refresh_token="id",
                token_expires_at=0,
            )

        elif existing_auth is not None:
            user = existing_auth.user
        else: # pragma: no cover
            raise RuntimeError("Unreachable")

    session = await Session.create(user=user, active=True)
    return {
        "token": session.to_jwt(),
        "expires_at": int(time() + config.jwt_ttl),
    }
```

# Тестування

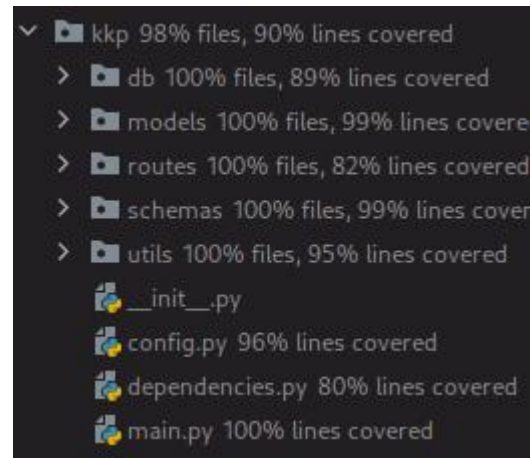


A screenshot of a test runner interface, likely Pytest, showing a list of test results. The interface has a dark theme. At the top, a green checkmark and the text 'Test Results' are followed by the total time '14 sec 785 ms'. Below this, a tree view shows a 'tests' directory with a green checkmark and the same total time. Under 'tests', there is a list of 15 test functions, each with a green checkmark and its execution time. At the bottom, there is a status bar with icons for Git, Run, Python Packages, TODO, and Problems. Below the status bar, it says 'Tests passed: 91 (4 minutes ago)'.

Test Results	14 sec 785 ms
tests	14 sec 785 ms
test_admin_animal_reports	1 sec 971 ms
test_admin_animals	1 sec 186 ms
test_animal_reports	782 ms
test_animal_subscriptions	434 ms
test_animals	313 ms
test_auth	5 sec 428 ms
test_donations	651 ms
test_media	408 ms
test_messages	763 ms
test_treatment_reports	144 ms
test_user	1 sec 808 ms
test_vet_clinics	790 ms
test_volunteer_requests	107 ms

Git Run Python Packages TODO Problems

Tests passed: 91 (4 minutes ago)



A screenshot of a code coverage report, likely generated by pytest-cov. It shows a tree view of a project structure with coverage percentages for files and lines. The root directory 'kkp' has 98% file and 90% line coverage. It contains subdirectories 'db', 'models', 'routes', 'schemas', and 'utils', all with 100% file coverage. The 'utils' directory contains several Python files with varying line coverage: '\_\_init\_\_.py' (not shown), 'config.py' (96%), 'dependencies.py' (80%), and 'main.py' (100%).

kkp	98% files, 90% lines covered
db	100% files, 89% lines covered
models	100% files, 99% lines covered
routes	100% files, 82% lines covered
schemas	100% files, 99% lines covered
utils	100% files, 95% lines covered
__init__.py	
config.py	96% lines covered
dependencies.py	80% lines covered
main.py	100% lines covered

# Підсумки

Розроблене програмне забезпечення є реалістичним з точки зору впровадження: воно побудоване на сучасних та стабільних технологіях, має модульну архітектуру та може бути легко адаптоване під різні потреби. Система вже зараз покриває основні завдання, які стоять перед фахівцями, що займаються порятунком диких тварин — облік, моніторинг, координація дій.

# Підсумки

Програмне забезпечення може бути успішно використано у наступних сферах:

- у реабілітаційних центрах для диких тварин;
- волонтерами та ветеринарами;
- громадськими організаціями, які займаються захистом тварин;
- широкою громадськістю;
- як частина державних або муніципальних ініціатив, пов'язаних із збереженням біорізноманіття та охороною навколишнього середовища.

# Підсумки

Проект має великий потенціал для подальшого розвитку, зокрема:

- AI-модулі для автоматичної діагностики та оцінки стану тварин на основі фото та відео;
- Інтеграція з державними реєстрами, системами GPS-трекерів для моніторингу випущених тварин;
- Розширена система аналітики та побудови звітів для дослідницьких або управлінських потреб;
- Механізми багатомовної підтримки для впровадження в інших країнах;
- Платформа для навчання волонтерів — інтерактивні курси, матеріали, сертифікація;
- Відкрите API, що дозволить іншим організаціям інтегрувати свої сервіси.