# ESWF — ERP-система управління автотранспортом

# 📦 ESWF — ERP-система управління автотранспортом

## 🔧 Загальні відомості

* - Структура інтерфейсу:
* - Header (верхня панель)
* - Main Area:
* - Sidebar (зліва) — розділи
* - ContentArea (по центру) — форми/довідники
* - ChatPanel (справа) — чат-панель з GPT
* - FooterTabs — вкладки внизу
* - \*\*Назва проєкту:\*\* ESWF (Enterprise Software for Fleet)
* - \*\*Призначення:\*\* управління автопарком, логістикою, обліком водіїв, маршрутами, витратами
* - \*\*Архітектура:\*\*
* - Frontend: React + Ant Design
* - Backend: Node.js + Express
* - База даних: SQLite (із планом переходу на PostgreSQL)

## 🗂 Основні модулі

### 1. \*\*Довідники (DirectoryList + EditForm)\*\*

* - Автомобілі (з брендами і моделями)
* - Водії
* - Маршрути
* - Пункти призначення
* - Номенклатура
* - Контрагенти
* - Організації
* - Валюти
* - Склади

### 2. \*\*Путеві листи (Documents)\*\*

* - Путеві листи
* - Облік витрат пального

### 3. \*\*Сервісні модулі\*\*

* - Обмін з BAF
* - Видалення об'єктів

### 4. \*\*Реєстри (Registers)\*\*

#### 📘 Леджери (залишки)

* - \*\*Cash Ledger\*\* – залишки грошових коштів за рахунками
* - \*\*Inventory Ledger\*\* – залишки товарів по складах

#### 📒 Журнали (рухи)

* - \*\*Cash Journal\*\* – рухи грошових коштів
* - \*\*Inventory Journal\*\* – рухи товарів

## 📁 Структура довідника "Пункти призначення"

### 📍 `location\_point`

* - Населені пункти України та інших країн
* - Поля: id, name, region, country, latitude, longitude

### 🏗 `transport\_hub`

* - Транспортні вузли (порти, аеропорти, станції)
* - Поля: id, name, type, location\_point\_id (FK), code, latitude, longitude

### 📦 `delivery\_point`

* - Адреси доставки (неточні або точні координати)
* - Поля: id, location\_point\_id (FK), address, note, latitude, longitude

## 🔌 API (поточна логіка доступу)

* - `GET /api/location-points` — всі населені пункти
* - `GET /api/transport-hubs?location\_id=1` — вузли міста
* - `GET /api/delivery-points?location\_id=1` — адреси доставки в місті

## 🧠 Принципи

* - Вся географічна логіка обʼєднується в абстракцію `destination`
* - Один населений пункт може мати багато хабів і багато адрес доставки
* - Імпорт довідників можливий з відкритих джерел (OSM, GeoNames, КОАТУУ)

## 💰 Облік залишків і рухів

### 📒 Таблиця `cash\_journal` – журнал рухів коштів

* - `id` — PK
* - `organization\_id` — FK → organizations
* - `account\_id` — FK → chart\_of\_accounts
* - `currency\_id` — FK → currencies
* - `movement\_type` — 'in' / 'out'
* - `amount` — сума операції
* - `document\_type` — тип документа (наприклад, incomingPayments)
* - `document\_id` — ID документа
* - `movement\_date` — дата операції
* - `created\_at` — створено
* - `comment` — примітка

### 📘 Таблиця `cash\_ledger` – залишки по рахунках

* - `id` — PK
* - `organization\_id` — FK → organizations
* - `account\_id` — FK → chart\_of\_accounts
* - `currency\_id` — FK → currencies
* - `balance` — поточний залишок
* - `last\_update` — дата останнього оновлення

### 🧠 Механізм

* - Кожна операція додається до `cash\_journal`
* - Одночасно оновлюється `cash\_ledger`
* - Швидке отримання залишку: `SELECT balance FROM cash\_ledger WHERE account\_id = ...`

### 5. \*\*Чат підприємства (AI Chat)\*\*

#### 📥 Реалізація:

* - \*\*Фронтенд:\*\* компонент `ChatPanel.jsx`, який виводить історію та поле вводу.
* - \*\*Бекенд:\*\* Django endpoint `/api/chat/`, який відправляє запити до OpenAI GPT-3.5.
* - \*\*Передача prompt:\*\* GPT отримує системний prompt із логістичним контекстом.

#### 🧠 Мета:

* - Надати користувачам можливість задавати питання у стилі:

\_“Яка собівартість перевезення 8 т піску з Києва до Одеси без зворотного рейсу?”\_

* - Відповіді адаптовані під контекст ERP.

#### 🛠️ Технічні деталі:

* - OpenAI API (`openai==0.28`) використовується для сумісності зі старим SDK.
* - Ключ зберігається в `.env`, підвантажується через `dotenv`.
* - Запити типу `POST /api/chat/` з параметром `message`.

#### ✅ Що реалізовано в рамках інтеграції ESWF + GPT (травень 2025):

* - ✅ Додано endpoint `/api/chat/` на Django з обробкою запитів до GPT.
* - ✅ Впроваджено `semantic\_parser.py`, який розпізнає intent користувача (типу `count\_all`, `get\_field\_by\_name`, `vehicle\_age\_query` тощо).
* - ✅ Підтримка запитів із “корявою” мовою через `normalize\_text()`.
* - ✅ Відповіді будуються на базі даних (`Django ORM`) — напр. “Скільки водіїв чоловіків?”
* - ✅ Додано логіку:
* - `count\_all` — скільки записів у таблиці (наприклад, "скільки машин?")
* - `get\_field\_by\_name` — отримання конкретного поля по імені (наприклад, "телефон водія Максим Мельник")
* - `vehicle\_age\_query` — скільки авто старших за N років
* - `driver\_age\_query` — скільки водіїв старших за N років
* - ✅ Впроваджено універсальну обробку моделей `Driver`, `Vehicle` (усі поля доступні).
* - ✅ Зменшено залежність від GPT: прості запити виконуються без повного GPT prompt.
* - ✅ Оновлено prompt semantic parser'а — враховано назви, синоніми, плутанину мов, орфографію.

#### 📌 Статус реалізації:

✅ Базовий чат функціонує.

🟡 Наступні кроки:

1. \*\*Зберігання історії\*\* у SQLite

2. \*\*Передача внутрішніх даних ERP у prompt\*\*

3. \*\*Обчислення собівартості з використанням логіки системи\*\*

4. \*\*Підтримка ролей і авторизованих відповідей\*\*

5. \*\*Можливість відповіді GPT на основі реальних довідників (через LangChain або вручну)\*\*

---

## 🤖 Очікування від AI (GPT) у проєкті

### 🎯 Мета:

Реалізувати не просто чат, а повноцінного \*\*логістичного помічника\*\*, який:

* - Розуміє природну мову з орфографічними помилками, суржиком, плутаниною мов
* - Має доступ до довідників і таблиць ERP
* - Вміє виконувати аналітичні запити: "Чи варто вже відправляти машину в Доманівку?"
* - Може сам комбінувати інформацію: кількість замовлень, вільні машини, вагу, маршрути
* - У майбутньому — сам запитує додаткові дані (function calling)

---

### ✅ Поточні можливості:

* - Обробка запитів типу:
* - "Скільки водіїв чоловіків?"
* - "Скільки в нас Mercedes?"
* - Відповідь будується на реальних даних із бази через прості запити у Django

---

### 🔜 Очікуваний функціонал:

* - Доступ до замовлень, рейсів, маршрутів, водіїв, машин
* - Вміння GPT самостійно ініціювати запит даних (через function calling)
* - Об’єднання логіки: наприклад, якщо замовлень у Доманівку мало — GPT пропонує об'єднати їх із сусіднім селом
* - Аналіз відстаней: або через внутрішній довідник, або через API (наприклад, OpenRouteService)
* - Універсальна обробка запитів без жорсткого `if` — через правила, шаблони або агента

---

### 🛠 Методи реалізації:

* - Початково: ключові слова → Django робить запит → GPT формує відповідь
* - Далі: function calling (GPT просить дані через API)
* - Ще далі: агент, який керує послідовністю запитів і логікою

---

### 📌 Висновок:

AI має перетворитися з "балакучої форми" на \*\*інструмент логістичного аналізу\*\*, здатний:

* - рахувати,
* - оцінювати маршрути,
* - шукати варіанти,
* - пропонувати оптимальні рішення

— і все це у вигляді простого діалогу з користувачем.

## 🛠️ Поточна реалізація (станом на сьогодні)

* - Усі довідники відкриваються у вкладках (`directoryList`, `directoryItem`, тощо)
* - Підтримка багатовкладковості
* - Вкладки керуються глобальним хук-контекстом `useTabs()`
* - React-компоненти: `App.jsx`, `DirectoryTab.jsx`, `UniversalEditForm.jsx`, `sections.js`, `FooterTabs.jsx`, `Sidebar.jsx`
* - Підключено SQLite через backend API
* - GitHub: https://github.com/yuraNikolaenko/ESWF

1