ДОДАТОК А

Алгоритм знайдення найближчої та наступної зупинки для симуляції руху транспорту у transportation-info-service

```
    def find_closest_stop(
        self, vehicle_lat: float, vehicle_lon: float, route_id: str
    ) -> Tuple[Optional[Stop], Optional[Stop]]:
        """Find the closest current and next stops for a vehicle on a route."""

        stops = self.get_route_stops_in_order(route_id)

        if not stops:
            return None, None

        # Find the closest stop to the vehicle's current position
        closest_stop = None
        min_distance = float("inf")

        for stop in stops:
            distance = self._calculate_distance(
                vehicle_lat,
                vehicle_lon,
                stop.stop_lat,
                stop.stop_lon,
            )
            if distance < min_distance:
                min_distance = distance
                closest_stop = stop

        if not closest_stop:
            return None, None

        # Find the next stop in sequence
        next_stop = None
        for stop in stops:
            if stop.stop_sequence > closest_stop.stop_sequence:
                next_stop = stop
                break

        # If no next stop found, wrap around to the first stop
        if not next_stop and stops:
            next_stop = stops[0]

        return closest_stop, next_stop
```

ДОДАТОК Б

React хук, який підписується на оновлення позицій транспортних засобів на конкретному маршруті за допомогою протоколу WebSocket.

```
      "use client";

import { useEffect, useRef, useState } from "react";
import { VehicleResponse } from "@/types";

export const useVehicleWebSocket = (routeId: string | null) => {
  const [vehicles, setVehicles] = useState<VehicleResponse[]>([]);
  const wsRef = useRef<WebSocket | null>(null);

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

    const ws = new WebSocket(`ws://localhost:8002/ws/${routeId}`);
    wsRef.current = ws;
    console.log("Opening WebSocket connection", routeId);

    ws.onmessage = (event) => {
      try {
        const data: VehicleResponse = JSON.parse(event.data);
        setVehicles((prev) => {
          const   updated   =   prev.filter((v)   =>   v.vehicle_id   !==
data.vehicle_id);
          return [...updated, data];
        });
      } catch (err) {
        console.error("Failed to parse vehicle update:", err);
      }
    };

    return () => {
      console.log("Closing WebSocket connection", routeId);
      ws.close();
    };
  }, [routeId]);

  return vehicles;
};
```

ДОДАТОК В

Тест перевірки функціоналу сервісу симуляції руху транспортних засобів

```
from datetime import datetime, timedelta
from unittest.mock import Mock, patch

from pytest_cases import fixture, parametrize
from service.config import VEHICLE_STOP_DURATION_SECONDS
from service.models import Vehicle
from service.services.position_calculator import StopId, VehicleState

@fixture
def sample_vehicle():
    return Vehicle(
        vehicle_id="1",
        route_id="route1",
        latitude=0.0,
        longitude=0.0,
    )


@fixture
def sample_vehicle_state():
    return VehicleState(
        current_stop_id=StopId("stop1"),
        next_stop_id=StopId("stop2"),
        current_position=(0.0, 0.0),
        next_stop_position=(1.0, 1.0),
        start_time=datetime.now(),
        arrival_time=datetime.now() + timedelta(seconds=60),
        is_at_stop=True,
        stop_start_time=datetime.now(),
    )


@parametrize(
    "is_at_stop,stop_duration,expected_is_at_stop",
    [
        (True, VEHICLE_STOP_DURATION_SECONDS - 1, True),  # Still at stop
        (True, VEHICLE_STOP_DURATION_SECONDS + 1, False),  # Time to move
        (False, 0, False),  # Already moving
    ],
)
def test_advance_vehicle_state(
    vehicle_position_service,
    sample_vehicle,
    sample_vehicle_state,
    is_at_stop,
    stop_duration,
    expected_is_at_stop,
):
    # Setup
    current_time = datetime.now()
    sample_vehicle_state.is_at_stop = is_at_stop
    sample_vehicle_state.stop_start_time = current_time - timedelta(
```

```
        seconds=stop_duration
    )

    # Mock the stop service to return some stops
    mock_current_stop = Mock(stop_id="stop3", stop_lat=2.0, stop_lon=2.0)
    mock_next_stop = Mock(stop_id="stop4", stop_lat=3.0, stop_lon=3.0)
    with patch.object(
        vehicle_position_service.stop_service,
        "find_closest_stop",
        return_value=(mock_current_stop, mock_next_stop),
    ):
        # Mock the vehicle service update method
        with patch.object(
            vehicle_position_service.vehicle_service,
            "update_vehicle_position",
        ) as mock_update_position:
            # Act
            result = vehicle_position_service.advance_vehicle_state(
                sample_vehicle, sample_vehicle_state, current_time
            )

            # Assert
            assert result.is_at_stop == expected_is_at_stop

            if not is_at_stop or stop_duration >=
VEHICLE_STOP_DURATION_SECONDS:
                # Should have updated vehicle position
                mock_update_position.assert_called_once()
            else:
                # Should not have updated vehicle position
                mock_update_position.assert_not_called()


def test_advance_vehicle_state_at_stop_transition(
    vehicle_position_service, sample_vehicle, sample_vehicle_state
):
    # Setup
    current_time = datetime.now()
    sample_vehicle_state.is_at_stop = True
    sample_vehicle_state.stop_start_time = current_time - timedelta(
        seconds=VEHICLE_STOP_DURATION_SECONDS + 1
    )

    # Mock the stop service to return some stops
    mock_current_stop = Mock(stop_id="stop3", stop_lat=2.0, stop_lon=2.0)
    mock_next_stop = Mock(stop_id="stop4", stop_lat=3.0, stop_lon=3.0)
    with patch.object(
        vehicle_position_service.stop_service,
        "find_closest_stop",
        return_value=(mock_current_stop, mock_next_stop),
    ):
        # Act
        result = vehicle_position_service.advance_vehicle_state(
            sample_vehicle, sample_vehicle_state, current_time
        )

        # Assert
        assert not result.is_at_stop
```

```
assert result.current_stop_id == StopId("stop3")
assert result.next_stop_id == StopId("stop4")
assert result.next_stop_position == (3.0, 3.0)
assert result.start_time == current_time
assert result.stop_start_time is None
```

ДОДАТОК Г

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 6/6/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІз-21-1_Гузєєв_Д_М.pdf
Автор Науковий керівник / Експерт
Гузєєв Дмитро Михайловичдоц. Кравець Н.С./Нечволод В.Ю.
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.58% 2.58% КП 1

1.31% 1.31% КЦ

25
Довжина фрази для коефіцієнта подібності 2

8617
Кількість слів

66712
Кількість символів

ДОДАТОК Д

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Веб-сервіс для відстеження руху міського транспорту

Гузєєв Д.М.,група ПЗПІз-21-1
Керівник: доцент Кравець Н.С.

4 червня 2025

Рисунок Д.1 – Слайд презентації 1 (рисунок виконаний самостійно)

Мета роботи

- Аналіз існуючих рішень у сфері відстеження руху громадського транспорту
- Дослідження методів розробки системи відстеження руху ТЗ
- Проектування та реалізація ПО з використанням досліджених методів
- Дизайн інтерфейсу користувача і тестування системи
- Аналіз результатів роботи

Актуальність роботи

- Відсутність уніфікованого інтерфейсу для відстеження руху ТЗ
- Зростання попиту на real-time системи
- Необхідність сучасного інтерфейсу до транспортної інфраструктури міста

Рисунок Д.2 – Слайд презентації 2 (рисунок виконаний самостійно)

Аналіз проблеми та існуючих рішень

Рішення	Переваги	Недоліки
Google Maps	Висока швидкість роботи; інтуїтивно зрозумілий інтерфейс; інтеграція з іншими сервісами Google	Високе споживання ресурсів пристрою; не завжди точно показує місцезнаходження зупинок, особливо в невеликих містах
Easyway	Велике покриття міст України; можливість додавання коментарів та оцінок для зупинок	інтерфейс часто зависає, загалом незручний і повільний; обмежена функціональність для планування складних маршрутів з пересадками
Moovit	глобальне покриття; потужні алгоритми для планування маршрутів; підтримка різних видів транспорту, включаючи велосипеди та самокати	обмежене покриття в українських містах; у безкоштовній версії обмежені можливості і реклама

Рисунок Д.3 – Слайд презентації 3 (рисунок виконаний самостійно)

Постановка задачі та опис системи

- Фрагментація систем**:** багато різних видів систем та форматів API для надання інформації про транспортні мережі
- Відсутність стандартизації: часто транспортні оператори використовують власні формати даних та протоколи
- Наслідки для користувачів**:**
  - неточна інформація про час прибуття;
  - відсутність єдиного інтерфейсу для всіх видів транспорту;
  - складність планування маршрутів.
- Система-рішення має бути масштабуємою і гнучкою
- Веб-додаток з інтуїтивним інтерфейсом для відстеження транспорту
- Real-time відображення позицій транспортних засобів на карті

Рисунок Д.4 – Слайд презентації 4 (рисунок виконаний самостійно)

Вибір технологій розробки

- PyCharm Professional Edition
- Next.js dev tools
- Visual Studio Code
- Chrome browser v137.0
- DBeaver
- Docker, docker-compose
- macOS 15 Sequoia

Рисунок Д.5 – Слайд презентації 5 (рисунок виконаний самостійно)

Архітектура створеного ПЗ

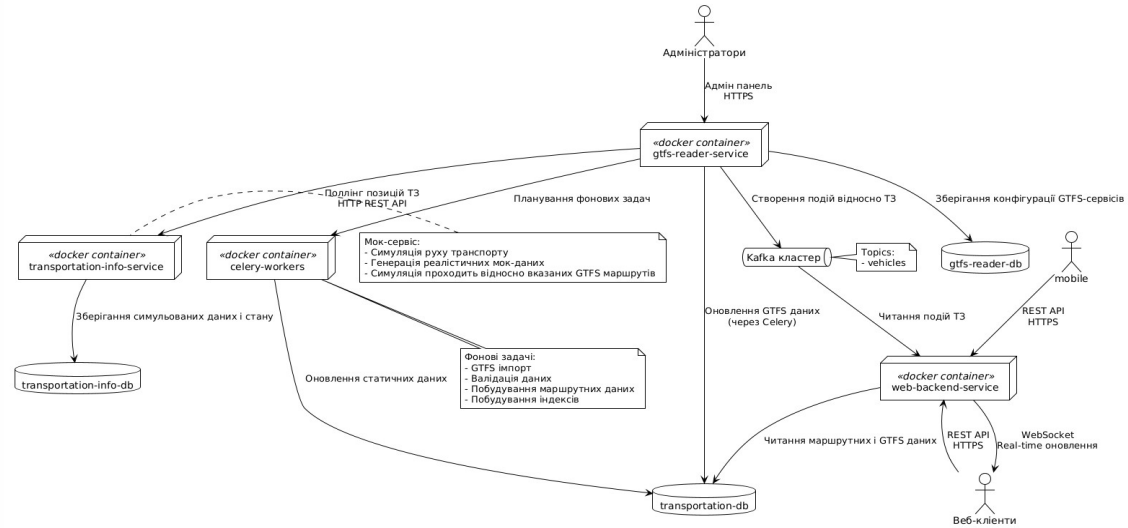

Рисунок Д.6 – Слайд презентації 6 (рисунок виконаний самостійно)

Опис ПЗ, що було використано у дослідженні

- Frontend:
  - TypeScript, React.js, Next.js, react-query, tailwind, daisyui
- Backend:
  - Python 3.12, Fastapi, SQLAlchemy, SQLModel, Pydantic, Mypy, pytest
- База даних:
  - PostgreSQL
  - Аддони для роботи з гео-даними: PostGIS, pgRouting
- Message broker: Apache Kafka
- Distributed task queue: Celery
- Контейнеризація: Docker, Docker-compose

Рисунок Д.7 – Слайд презентації 7 (рисунок виконаний самостійно)

Дизайн системи

Методи

- **Мікросервісна архітектура** для забезпечення масштабованості
- **Event-Driven Architecture** для real-time комунікації
- **Database per Service** для ізоляції даних
- **API Gateway** для централізованого доступу
- **Circuit Breaker** для підвищення надійності

Послідовність

- Аналіз вимог та дослідження існуючих рішень
- Проектування архітектури та вибір технологій
- Розробка мікросервісів (transportation-info, gtfs-reader, web-backend)
- Інтеграція компонентів через Kafka та REST API
- Тестування та оптимізація продуктивності
- Розгортання в контейнерізованому середовищі

Рисунок Д.8 – Слайд презентації 8 (рисунок виконаний самостійно)

Приклад реалізації

```sql
INSERT INTO walking_connections (
    from_stop_id, to_stop_id, walk_time, distance, geom
)
WITH nearby_pairs AS (
    SELECT
        s1.stop_id as from_stop,
        s2.stop_id as to_stop,
        ST_Distance(s1.geometry::geography, s2.geometry::geography) as distance,
        ST_MakeLine(s1.geometry, s2.geometry) as line_geom
    FROM stops s1
    JOIN stops s2 ON s1.stop_id < s2.stop_id  -- Avoid duplicates
    WHERE ST_DWithin(s1.geometry::geography, s2.geometry::geography, 800)  -- 800m max walk
    AND s1.location_type = 0 AND s2.location_type = 0  -- Only actual stops
    AND s1.is_active = true AND s2.is_active = true
)
SELECT
    from_stop,
    to_stop,
    GREATEST(120, distance / 1.4)::INTEGER, -- Min 2 min walk, 1.4 m/s walking speed
    distance,
    line_geom
FROM nearby_pairs
WHERE distance >= 50  -- Don't connect very close stops
AND distance <= 800;
```


Рисунок Д.9 – Слайд презентації 9 (рисунок виконаний самостійно)

Приклад реалізації

```python
async def fetch_vehicles(self, service: Service):
    try:
        logger.info(
            "Fetching vehicles", service_id=service.id, service_name=service.name
        )
        async with aiohttp.ClientSession() as session:
            async with session.get(
                f"{service.api_base}/api/v1/vehicles",
                headers={"Content-Type": "application/json"},
            ) as response:
                vehicles = await response.json()
                logger.info(
                    "Vehicles response", num_vehicles=len(vehicles), service_id=service.id
                )
                for vehicle in vehicles:
                    vehicle_model = VehicleResponse(**vehicle)
                    await self.kafka_producer.send(
                        "vehicles",
                        key=vehicle_model.vehicle_id.encode("utf-8"),
                        value=vehicle_model.model_dump_json().encode("utf-8"),
                    )
        service.vehicles_fetched_at = datetime.now()  # type: ignore
        self.db.commit()
    except Exception as e:
        logger.error("Error fetching vehicles", service=service, error=e)
```


Рисунок Д.10 – Слайд презентації 10 (рисунок виконаний самостійно)

Приклад реалізації

```
1   const updateFeatures = () => {
2     const source = sourceRef.current;
3     if (!source) return;
4     const existingFeatures = source.getFeatures();
5     const currentIds = new Set(vehicleUpdates.map((v) => v.vehicle_id));
6     existingFeatures.forEach((feature) => {
7       if (!currentIds.has(feature.getId()!.toString())) {
8         source.removeFeature(feature);
9       }
10    });
11    vehicleUpdates.forEach((vehicle) => {
12      const existingFeature = source.getFeatureById(vehicle.vehicle_id);
13      const newPoint = new Point(
14        fromLonLat([vehicle.longitude, vehicle.latitude])
15      );
16      if (existingFeature) {
17 existingFeature.setGeometry(newPoint);
18      } else {
19        const date = new Date(vehicle.timestamp).toLocaleTimeString();
20        const feature = new Feature({
21          name: `Автобус #${vehicle.vehicle_id}.\nОновлено ${date}`,
22          geometry: newPoint,
23        });
24        feature.setId(vehicle.vehicle_id);
25        source.addFeature(feature);
26      }
27    });
28  };
```


Рисунок Д.11 – Слайд презентації 11 (рисунок виконаний самостійно)

Скріншот з додатку

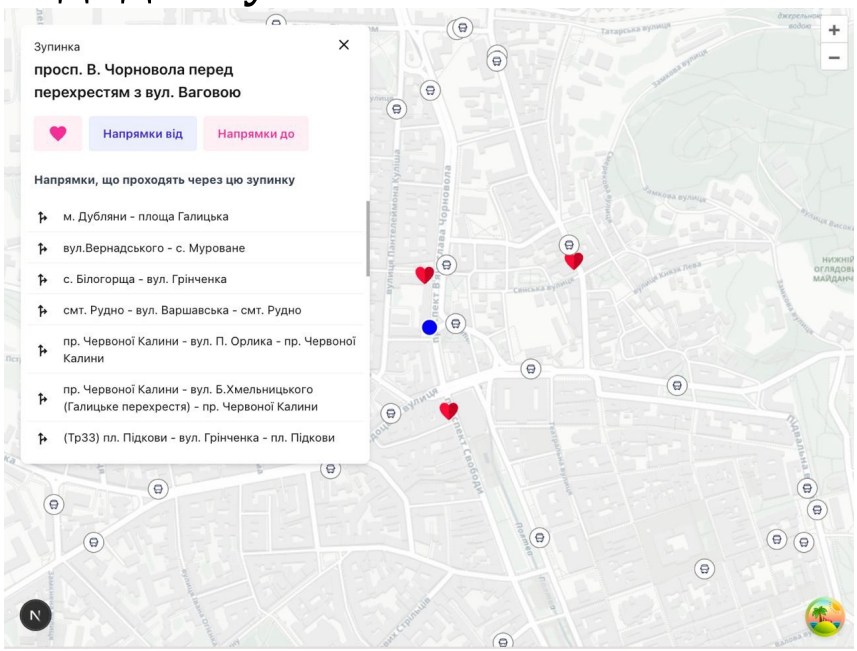

Рисунок Д.12 – Слайд презентації 12 (рисунок виконаний самостійно)

Скріншот з додатку

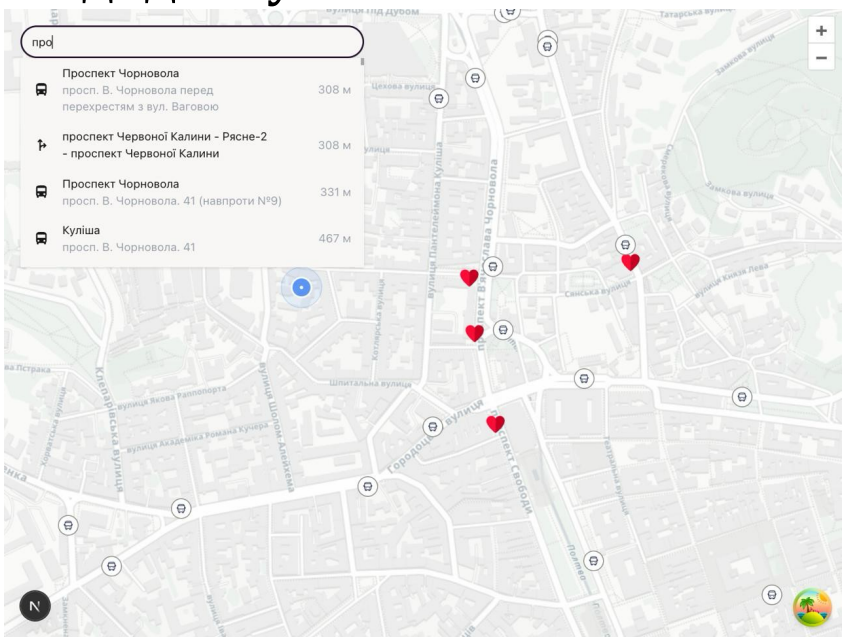

Рисунок Д.13 – Слайд презентації 13 (рисунок виконаний самостійно)

Інтерфейс користувача

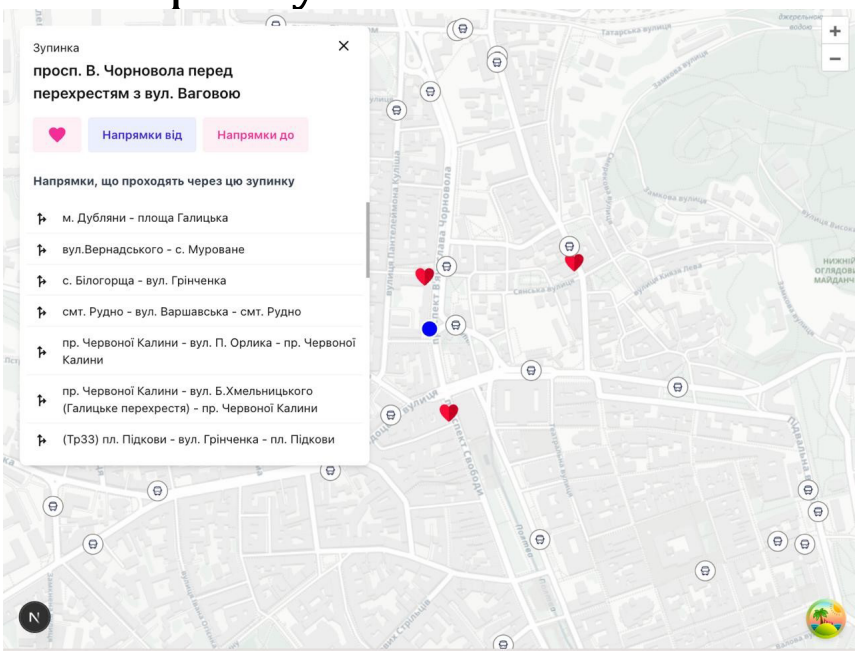

Рисунок Д.14 – Слайд презентації 14 (рисунок виконаний самостійно)

Інтерфейс користувача

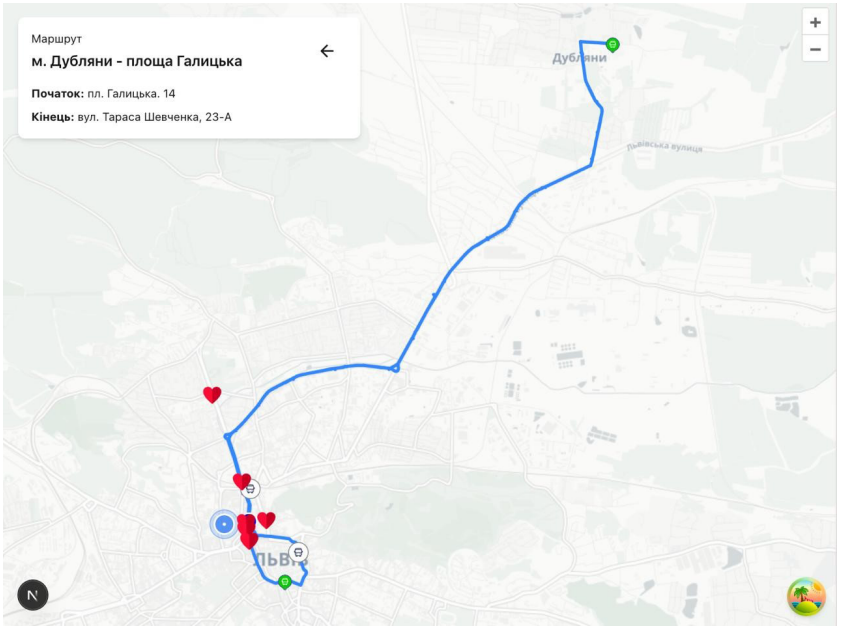

Рисунок Д.15 – Слайд презентації 15 (рисунок виконаний самостійно)

Інтерфейс користувача

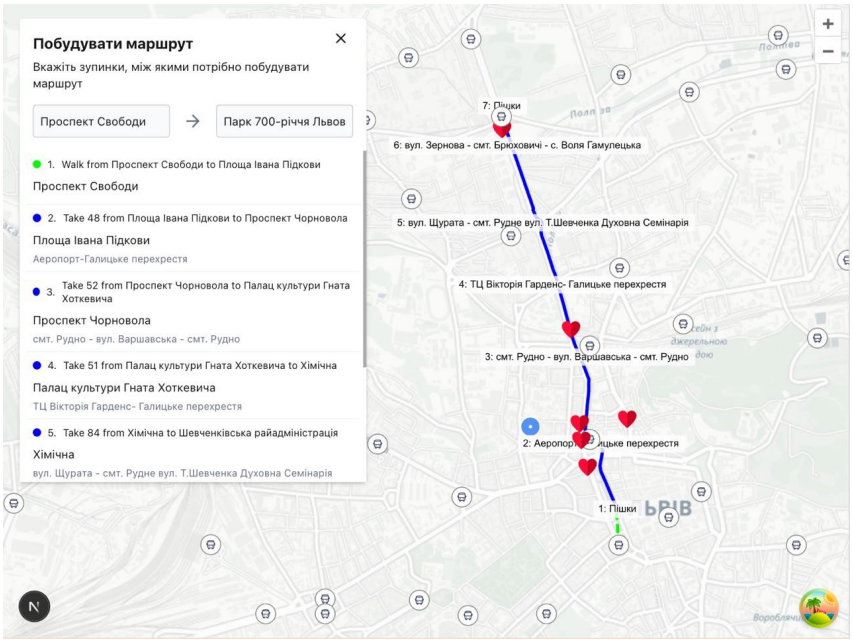

Рисунок Д.16 – Слайд презентації 16 (рисунок виконаний самостійно)

Тестування

ID	Опис	Передумови	Кроки тесту	Очікуємий результат	Пріоритет	Результат	Оточення	Дата
1	Побудування маршруту	Основний екран додатку відкритий	1. Обрати одну з зупинок на карті 2. Натиснути «побудувати маршрут від» 3. Натиснути на поле пошуку, і ввести назву кінцевої зупинки. 4. Вибрати її	На мапі відображається маршрут яким можна скористатись щоб проїхати від початкової зупинки до кінцевої	Високий	Пройшов	Chrome 137.0	31.05.2025
2	Коректне відображення ТЗ на мапі	Основний екран додатку відкритий	1. В пошуку знайти один з маршрутів і вибрати його 2. Почекати 5-10 секунд	На мапі мають з'явитися індикатори рухаючихся ТЗ з інформацію про дату оновлення	Високий	Пройшов	Chrome 137.0	31.05.2025

Рисунок Д.17 – Слайд презентації 17 (рисунок виконаний самостійно)

Підсумки

Корисність отриманих результатів

- Створено єдину платформу для відстеження всіх видів транспорту
- Забезпечено real-time оновлення позицій транспортних засобів
- Підвищено зручність планування поїздок для пасажирів

Можливості використання

- Інтеграція з існуючими транспортними системами
- Використання транспортними операторами для моніторингу
- Основа для розробки мобільних додатків
- Впровадження в системи «розумного міста»

Рисунок Д.18 – Слайд презентації 18 (рисунок виконаний самостійно)

Перспективи розвитку

- Використання Machine Learning алгоритмів для прогнозування часу прибуття транспорту до зупинок
- Сповіщення про затримки, пробки на дорогах
- Окремий мобільний додаток
- Підтримка інших форматів даних
- Додати CI/CD у репозиторій
- Інтеграція з системами оплати

Рисунок Д.19 – Слайд презентації 19 (рисунок виконаний самостійно)

ДОДАТОК Е

Специфікація ПЗ

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра програмної інженерії

СПЕЦИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Веб-сервісу для відстеження руху міського транспорту

Харків

2025

ЗМІСТ

1 Вступ 20
1.1 Огляд продукту 20
1.2 Мета 20
1.3 Межі 20
1.4 Посилання 21
1.5 Означення та абревіатури 21
2 Загальний опис 23
2.1 Перспективи розвитку 23
2.2 Функції продукту 23
2.3 Характеристики користувачів 24
2.4 Загальні обмеження 24
2.5 Припущення в залежності 25
3 Специфікація вимог 26
3.1 Вимоги до зовнішніх інтерфейсів 26
3.1.1 Інтерфейс користувача 26
3.1.2 Програмний інтерфейс 26
3.1.4 Комунікаційний протокол 27
3.1.5 Обмеження пам’яті 27
3.1.6 Операції 27
3.1.7 Функції продукту 28
3.1.8 Припущення й залежності 28
3.2 Властивості програмного продукту 28
3.3 Атрибути програмного продукту 29
3.3.1 Надійність 29
3.3.2 Доступність 29
3.3.3 Безпека 29
3.3.4 Супроводжуваність 30
3.3.5 Переносимість 30

3.3.6 Продуктивність 30
3.4 Вимоги бази даних 30
3.5 Інші вимоги 31
4 Додаткові матеріали 32

1 ВСТУП

1.1 Огляд продукту

Програмний застосунок для відстеження руху міського транспорту призначений для надання користувачам актуальної інформації про місцезнаходження, маршрути та розклад громадського транспорту у режимі реального часу. Система інтегрує дані з різних GTFS-систем (General Transit Feed Specification), що дозволяє об’єднувати інформацію про транспортні маршрути різних міст у єдиному інтерфейсі. Програмний застосунок складається з веб-додатку, бекенд-сервісу, бази даних та додаткових мікросервісів для обробки та агрегації даних.

1.2 Мета

Метою розробки є створення масштабованого програмного застосунку, який забезпечує зручний доступ до інформації про рух міського транспорту, дозволяє користувачам швидко будувати оптимальні маршрути з урахуванням пересадок та часу очікування, а також надає інструменти для міських адміністрацій та транспортних компаній для моніторингу ефективності транспортної мережі. Завдяки уніфікованому інтерфейсу та можливості підключення різних GTFS-джерел, система може бути адаптована для будь-якого міста без значних змін у базовій архітектурі

1.3 Межі

Програмний застосунок призначений для використання у веб-браузерах на сучасних платформах (Chrome, Firefox, Safari, Edge, Opera) з підтримкою WebSocket для оновлення інформації в реальному часі. Система не передбачає обов’язкової авторизації користувачів, не зберігає персональні дані та не реалізує функції бронювання чи оплати проїзду. Застосунок може інтегруватися з різноманітними GTFS-системами, але не підтримує автоматичну інтеграцію з

іншими стандартами обміну даними про громадський транспорт без додаткових доопрацювань.

1.4 Посилання

- GTFS Specification — https://developers.google.com/transit/gtfs;
- PostgreSQL — https://www.postgresql.org/.
- FastAPI — https://fastapi.tiangolo.com/;
- React — https://react.dev/;
- pgRouting — https://pgrouting.org/;
- PostGIS — https://postgis.net/.

1.5 Означення та абревіатури

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

RPS – Requests per Second

API – Application Programming Interface

AWS – Amazon Web Services

WS – Web Sockets

GTFS – General Transit Feed Specification

RDBMS – Relational Database Management System

ORM – Object-Relational Mapping

AVL - Automated Vehicle Location

SIRI - Service Interface for Real Time Information

NeTEx - Network Timetable Exchange

SSR – Server-Side Rendering

Kafka – Розподілена потокова платформа для обміну повідомленнями

LocalStorage – Механізм веб-браузера для зберігання даних на стороні клієнта

FastAPI – Веб-фреймворк для Python

React – JavaScript-бібліотека для побудови інтерфейсів користувача

GTFS Static – Статична частина GTFS, що містить інформацію про маршрути, зупинки, розклади

GTFS Realtime – Розширення GTFS для передачі даних у реальному часі

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи розвитку

Програмний застосунок для відстеження руху міського транспорту розроблений з урахуванням можливості подальшого розширення функціональності. В майбутньому планується:

- підтримка додаткових стандартів обміну даними (SIRI, NeTEx, TransXChange); для інтеграції з іншими транспортними системами.
- розробка мобільних додатків для iOS та Android;
- авторизація для персоналізації можливостей додатку;
- інтеграція з платіжними системами для можливості придбання квитків онлайн;
- розширення аналітики для міських адміністрацій та операторів транспорту;
- підтримка мультимовності для забезпечення доступності користувачам з різних країн.

2.2 Функції продукту

Основні функції програмного застосунку:

- відображення маршрутів громадського транспорту на інтерактивній мапі;
- пошук оптимального маршруту між двома точками з урахуванням пересадок та часу очікування;
- відстеження руху транспорту в реальному часі за допомогою GTFS Realtime;
- відображення розкладу руху для вибраних зупинок та маршрутів;
- фільтрація транспорту за типом (автобус, трамвай, метро тощо);
- відображення інформації про зупинки (назва, координати, доступні маршрути);

– інтеграція з різними GTFS-джерелами для підтримки багатьох міст;

– інструменти для адміністраторів (моніторинг стану транспорту, аналіз завантаженості маршрутів).

2.3 Характеристики користувачів

Основні категорії користувачів системи:

– звичайні користувачі**:**

– пасажири громадського транспорту, які потребують актуальної інформації про маршрути та розклад;

– не мають спеціальних знань, працюють через веб-інтерфейс;

– адміністратори системи**:**

– мають доступ до аналітичних інструментів та можуть налаштовувати джерела даних;

– розробники**:**

– програмісти, які інтегрують нові джерела даних або розширюють функціональність системи;

– працюють з API та базою даних.

2.4 Загальні обмеження

– **обмежена підтримка стандартів:**

– наразі система підтримує лише GTFS (Static та Realtime), інші стандарти потребують доопрацювання;

– **відсутність мобільних додатків:**

– початкова версія доступна лише через веб-браузер;

– **відсутність авторизації:**

– користувачі не можуть зберігати свої налаштування;

– **відсутність платіжних функцій:**

– неможливо купити квиток або здійснити оплату через систему.

2.5 Припущення в залежності

Припущення, від яких залежить коректна робота системи:

– наявність GTFS-джерел даних:

– система працює лише за умови наявності GTFS-файлів (Static та Realtime) для відповідних міст;

– стабільний інтернет-зв’язок:

– для роботи веб-інтерфейсу та оновлення даних у реальному часі необхідний стабільний доступ до інтернету;

– сумісність браузерів:

– система розрахована на роботу у сучасних веб-браузерах з підтримкою WebSocket;

– сумісність серверного ПЗ:

– для роботи бекенду необхідні актуальні версії PostgreSQL, PostGIS, pgRouting, FastAPI;

– відсутність значних змін у GTFS-стандарті:

– система розрахована на сумісність з поточними версіями GTFS, значні зміни можуть вимагати оновлення коду;

– відсутність DDoS-атак або значних перевантажень:

– система не має вбудованих механізмів захисту від масових атак або перевантажень.

3 СПЕЦИФІКАЦІЯ ВИМОГ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Веб-інтерфейс:

– інтерактивна мапа з відображенням маршрутів, зупинок та рухомого складу;

– пошук маршрутів між двома точками з можливістю вибору часу відправлення;

– відображення розкладу руху на вибраних зупинках;

– фільтрація транспорту за типом (автобус, трамвай, метро тощо);

– відображення інформації про зупинки та маршрути;

– інтуїтивно зрозумілий та адаптований до різних екранів інтерфей.

Сервер:

– стандартні серверні рішення (x86-64 архітектура);

– вимоги до процесора, оперативної пам’яті та сховища даних залежать від обсягу оброблюваної інформації та кількості користувачів.

Клієнт:

– сучасний комп’ютер або мобільний пристрій з підтримкою сучасних веб-браузерів;

– достатня оперативна пам’ять (мінімум 2 ГБ) та пропускна здатність інтернет-з’єднання.

3.1.2 Програмний інтерфейс

API:

– RESTful API для взаємодії з бекендом;

– WebSocket для оновлення даних у реальному часі;

– підтримка JSON для обміну даними.

Інтеграція:

– підтримка GTFS (Static та Realtime) для імпорту маршрутів, зупинок та даних про рух транспорту;
– можливість додаткових інтеграцій через API сторонніх сервісів.

3.1.4 Комунікаційний протокол

– HTTP/HTTPS:
– для обміну даними між клієнтом та сервером;
– WebSocket:
– для оновлення інформації про рух транспорту в реальному часі;
– GTFS:
– для імпорту та оновлення даних про маршрути, зупинки та розклад.

3.1.5 Обмеження пам’яті

Сервер:

– мінімум 8 ГБ оперативної пам’яті для середнього навантаження;
– рекомендовано 16 ГБ та більше для великих міст або високого трафіку.

Клієнт:

– достатньо стандартної пам’яті сучасного браузера (LocalStorage для кешування даних).

3.1.6 Операції

Основні операції:

– пошук маршрутів;
– відображення інформації про зупинки та маршрути;
– оновлення даних у реальному часі;
– фільтрація транспорту за типом;
– імпорт даних з GTFS-джерел.

Додаткові операції:

– моніторинг стану транспорту для адміністраторів;
– аналітика руху транспорту.

3.1.7 Функції продукту

– відображення маршрутів та зупинок на мапі;
– пошук оптимального маршруту з урахуванням пересадок та часу очікування;
– відстеження руху транспорту в реальному часі;
– відображення розкладу руху на вибраних зупинках;
– фільтрація транспорту за типом;
– імпорт даних з GTFS-джерел;
– моніторинг стану транспорту для адміністраторів.

3.1.8 Припущення й залежності

– наявність GTFS-джерел даних;
– стабільний інтернет-зв’язок для роботи клієнта та сервера;
– сумісність браузерів з WebSocket;
– відсутність значних змін у GTFS-стандарті;
– відсутність DDoS-атак або значних перевантажень системи.

3.2 Властивості програмного продукту

– Масштабованість:
– Система здатна обробляти дані для різних міст та адаптуватися до зростання кількості користувачів;
– Модульність:
– Архітектура дозволяє легко додавати нові функції та інтегрувати нові джерела даних;
– Відкритість:

– Підтримка відкритих стандартів (GTFS) та можливість інтеграції з іншими системами;

– Кросс-платформенність:

– Веб-інтерфейс доступний з будь-якої платформи з сучасним браузером.

3.3 Атрибути програмного продукту

3.3.1 Надійність

– стабільна робота:

– система працює без збоїв у штатному режимі;

– відновлення після збоїв:

– можливість швидкого відновлення роботи після апаратних або програмних збоїв.

3.3.2 Доступність

– відкритий доступ:

– система доступна для всіх користувачів через веб-інтерфейс без обов’язкової реєстрації;

– адаптація до різних пристроїв:

– інтерфейс адаптований для роботи на комп’ютерах, планшетах та смартфонах.

3.3.3 Безпека

– захист даних:

– система не зберігає персональні дані користувачів;

– HTTPS:

– всі дані передаються по захищеному протоколу;

– обмежений доступ до адміністративних функцій:

– адміністративні інструменти доступні лише авторизованим користувачам.

3.3.4 Супроводжуваність

– документований код:

– код системи добре документований для полегшення підтримки та розвитку;

– модульна архітектура:

– дозволяє легко вносити зміни та додавати нові функції;

– типізація:

– присутність типізації на бекенді і на фронтенді дозволяє краще розуміти структуру ПЗ і полегшує внесення змін у кодову базу.

3.3.5 Переносимість

– підтримка різних операційних систем:

– серверна частина працює на Linux та Windows завдяки Docker;

– веб-клієнт:

– доступний з будь-якої платформи з сучасним браузером.

3.3.6 Продуктивність

– швидка обробка запитів:

– оптимізовані алгоритми пошуку маршрутів та обробки даних;

– ефективне використання ресурсів:

– мінімальні вимоги до апаратних ресурсів для клієнта.

3.4 Вимоги бази даних

– тип БД:

– реляційна база даних PostgreSQL з розширеннями PostGIS та pgRouting;

– структура даних:

– таблиці для маршрутів, зупинок, розкладів, рухомого складу, історії руху;

– інтеграція:

- підтримка імпорту даних з GTFS-файлів
- масштабування:
- можливість збільшення обсягу даних без втрати продуктивності;
- резервне копіювання:
- регулярне резервне копіювання даних для забезпечення надійності.

3.5 Інші вимоги

- тестування:
- система повинна проходити регулярне тестування на коректність роботи та безпеку;
- документація:
- наявність інструкцій для користувачів, адміністраторів та розробників;
- підтримка:
- можливість технічної підтримки користувачів та адміністраторів.

4 ДОДАТКОВІ МАТЕРІАЛИ

- GTFS-специфікація: https://developers.google.com/transit/gtfs;
- документація PostgreSQL: https://www.postgresql.org/docs/;
- документація PostGIS: https://postgis.net/documentation/;
- документація pgRouting: https://docs.pgrouting.org/;
- документація FastAPI: https://fastapi.tiangolo.com/;
- документація React: https://react.dev/.