

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної освіти
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Система автоматизованого планування графіків робочих змін
із застосуванням алгоритму Min-Conflicts
(тема)

Виконав:
здобувач _____ другого _____ року навчання,
групи _____ ІТШІпз-24-1

Вероніка МАЛЫЦЕВА
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна

Освітня програма _____ Штучний інтелект

(повна назва освітньої програми)

Керівник ст.викл. Дмитро ВОДЯНИЦЬКИЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної освіти _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Мальцевій Вероніці Валеріївні _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Система автоматизованого планування графіків робочих змін із застосуванням алгоритму Min-Conflicts _____

затверджена наказом університету від 18 травня 2026 р. № 77Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2026 р.

3. Вихідні дані до роботи _____ Відомості про методи та алгоритми автоматизованого планування, модель Constraint Satisfaction Problem, алгоритм Min-Conflicts, документація React та TypeScript, документація мови програмування Python, посібники для роботи з PostgreSQL, матеріали з OpenAI API та NLP. _____

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі _____

2) Проєктування системи _____

3) Вибір програмних засобів та технологій _____

4) Опис реалізованої програми _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.05.2026	виконано
2	Аналіз предметної галузі	19.05.2026	виконано
3	Огляд існуючих систем та застосунків	21.05.2025	виконано
4	Концептуальна розробка системи	22.05.2026	виконано
5	Визначення вимог до розроблюваної системи	22.05.2026	виконано
6	Дослідження Constrain Satisfaction Problem	23.05.2026	виконано
7	Аналіз алгоритму Min-Conflict	23.05.2026	виконано
8	Проектування архітектури системи	24.05.2026	виконано
9	Проектування структури бази даних	24.05.2026	виконано
10	Розробка backend-частини системи	25.05.2025	виконано
11	Розробка frontend-інтерфейсу	26.05.2026	виконано
12	Реалізація алгоритму автоматичного формування робочих графіків	27.05.2026	виконано
13	Інтеграція OpenAI API та NLP-модуля	28.05.2026	виконано
14	Тестування програмної системи	29.05.2026	виконано
15	Аналіз результатів роботи	29.05.2026	виконано
16	Написання пояснювальної записки	30.05.2026	виконано
17	Підготовка презентації та доповіді	30.05.2026	виконано
18	Попередній захист	12.06.2026	виконано
19	Захист перед ДЕК	17.06.2026	

Дата видачі завдання 18 травня 20 26 р.

Здобувач 
(підпис)

Керівник роботи _____
(підпис)

ст. викл. Дмитро ВОДЯНИЦЬКИЙ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 104 с., 40 рис., 5 табл., 3 дод., 20 джерел.

АВТОМАТИЗОВАНЕ ПЛАНУВАННЯ ГРАФІКІВ, АЛГОРИТМИ ПЛАНУВАННЯ, ВЕБ-СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ, CONSTRAINT SATISFACTION PROBLEM, FASTAPI, MIN-CONFLICTS, NLP, OPENAI API, POSTGRESQL, REACT, TYPESCRIPT.

Об'єктом дослідження є процес автоматизованого формування графіків робочих змін із використанням сучасних веб-технологій та алгоритмів штучного інтелекту.

Предметом дослідження є архітектура, функціональність та програмна реалізація інтелектуальної веб-системи автоматизованого планування графіків робочих змін із використанням алгоритму Min-Conflicts.

Метою роботи є розробка системи автоматизованого планування графіків робочих змін, здатної враховувати складні обмеження, побажання працівників та особливості організації робочого процесу.

У роботі проведено аналіз предметної галузі та проблем ручного планування графіків. Для реалізації системи запропоновано багатoshарову архітектуру із використанням React і TypeScript для frontend, FastAPI для backend та PostgreSQL для збереження даних. Основою алгоритмічної частини є алгоритм Min-Conflicts для розв'язання задачі Constraint Satisfaction Problem, а також інтеграція NLP-технологій та OpenAI API для обробки побажань користувачів природною мовою.

Результати роботи можуть бути використані під час розробки корпоративних систем управління персоналом, платформ автоматизованого планування чергувань, медичних систем формування змін та інших веб-застосунків, що потребують автоматизованого розподілу ресурсів.

ABSTRACT

Bachelor's thesis contains: 104 pp., 40 fig., 5 tabl., 3 ann., 20 references.

ARTIFICIAL INTELLIGENCE, AUTOMATED SCHEDULING, CONSTRAINT SATISFACTION PROBLEM, FASTAPI, MIN-CONFLICTS, NLP, OPENAI API, POSTGRESQL, REACT, SCHEDULING ALGORITHMS, TYPESCRIPT, WEB SYSTEM.

The object of the research is the process of automated work shift scheduling using modern web technologies and artificial intelligence algorithms.

The subject of the research is the architecture, functionality, and software implementation of an intelligent web-based shift scheduling system using the Min-Conflicts algorithm.

The purpose of the work is to develop an automated scheduling system capable of considering complex constraints, employee preferences, and specific characteristics of the working process.

The paper analyzes the subject area and the problems of manual schedule planning. A multilayer architecture based on React and TypeScript for the frontend, FastAPI for the backend, and PostgreSQL for data storage is proposed for the implementation of the system. The core of the algorithmic component is the Min-Conflicts algorithm for solving the Constraint Satisfaction Problem, as well as the integration of NLP technologies and OpenAI API for processing user requests expressed in natural language.

The results of the work can be used in the development of corporate workforce management systems, automated shift scheduling platforms, medical scheduling systems, and other web applications that require automated resource allocation.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі	12
1.1 Опис предметної галузі	12
1.2 Аналіз існуючих рішень	13
1.2.1 Ручні та напівавтоматизовані методи планування	13
1.2.2 Програмні системи планування змін	14
1.2.3 Опис існуючих рішень	15
1.3 Порівняльний аналіз	23
1.4 Постановка задачі.....	24
2 Проєктування системи.....	26
2.1 Функціональні вимоги до системи.....	26
2.2 Вибір архітектури системи.....	28
2.3 Проєктування архітектури системи	31
2.3.1 Діаграма варіантів використання (Use Case Diagram)	31
2.3.2 Схема користувацького потоку (User Flow)	34
2.3.3 Activity Diagram – BPwin-діаграма бізнес-процесу.....	37
2.4 Проєктування бази даних	38
2.4.1 Таблиця users	40
2.4.2 Таблиця availability	40
2.4.3 Таблиця schedule_assignments.....	41
2.4.4 Нормалізація бази даних	41
3 Вибір програмних засобів та технологій	44
3.1 Python.....	44
3.2 FastAPI.....	47
3.3 React.....	49
3.4 TypeScript	52
3.5 PostgreSQL	54

3.6 OpenAI API.....	56
4 Опис реалізованої програми	59
4.1 Constraint Satisfaction Problem.....	59
4.2 Алгоритм Min-Conflicts	61
4.3 Формалізація обмежень.....	63
4.4 Реалізація backend	69
4.5 Реалізація NLP-модуля	73
4.6 Реалізація frontend та користувацького інтерфейсу	76
4.6.1 Архітектура frontend-системи.....	76
4.6.2 Реалізація інтерфейсу введення даних	83
4.6.3 Реалізація інтерфейсу перегляду графіка.....	87
4.6.4 Реалізація інтерфейсу введення даних	91
Висновки	94
Перелік джерел посилання	97
Додаток А Схеми та інтерфейс системи.....	99
Додаток Б Реалізація алгоритму Min-Conflicts	102
Додаток В Відомість кваліфікаційної роботи	104

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- 1НФ – перша нормальна форма;
- 2НФ – друга нормальна форма;
- 3НФ – третя нормальна форма;
- БД – база даних;
- ІТ – інформаційні технології;
- ІІІ – штучний інтелект;
- CSP – Constraint Satisfaction Problem – задача задоволення обмежень;
- FastAPI – вебфреймворк для розробки API на Python;
- Min-Conflicts – алгоритм локального пошуку для розв’язання задач задоволення обмежень;
- NLP – Natural Language Processing – обробка природної мови;
- OpenAI API – Application Programming Interface – програмний інтерфейс для доступу до сервісів OpenAI;
- REST API – Representational State Transfer Application Programming Interface – програмний інтерфейс взаємодії на основі архітектурного стилю REST;
- Use Case Diagram – діаграма варіантів використання;
- User Flow – схема користувацького потоку.

ВСТУП

Організація змінної роботи є однією з найбільш складних задач управління персоналом у сучасних компаніях, що працюють у режимі безперервного функціонування. Особливо актуальною дана проблема є для команд технічної підтримки, центрів моніторингу інфраструктури, мережових операційних центрів (NOC) та інших підрозділів, діяльність яких передбачає цілодобове покриття змін, чергування у нічний час та підтримку standby-режиму. У подібних системах планування графіків повинно враховувати значну кількість взаємопов'язаних обмежень, серед яких доступність працівників, вимоги до відпочинку між змінами, рівномірність навантаження, специфіка нічних чергувань та індивідуальні побажання персоналу.

Традиційні підходи до планування змін здебільшого базуються на ручному формуванні графіків або використанні напів автоматизованих інструментів. Подібні методи є малоефективними при роботі з великими командами та складними міжперіодними залежностями. Значна частина існуючих програмних рішень орієнтована на типові сценарії змінної роботи та використовує жорстко зафіксовані моделі планування, що обмежує можливість адаптації системи до специфічних вимог конкретної організації. Крім цього, сучасні системи часто не забезпечують достатнього рівня прозорості алгоритмічних рішень та не підтримують гнучке врахування індивідуальних побажань працівників.

У зв'язку з цим актуальною є задача розробки системи автоматизованого планування графіків змін, яка поєднує алгоритмічні методи constraint-based planning із сучасними веб-технологіями та засобами взаємодії природною мовою. Використання алгоритму Min-Conflicts дозволяє розглядати процес формування графіка як задачу задоволення обмежень (Constraint Satisfaction Problem) та виконувати пошук

допустимого розкладу шляхом мінімізації конфліктів між жорсткими та м'якими обмеженнями.

Особливістю даної роботи є інтеграція NLP-модуля на основі OpenAI API для підтримки взаємодії користувача із системою природною мовою. Це дозволяє спростити процес введення даних про доступність та забезпечити більш гнучку формалізацію індивідуальних побажань працівників.

Об'єктом дослідження є процес автоматизованого планування графіків робочих змін у командах із безперервним режимом роботи.

Предметом дослідження є методи формалізації обмежень змінної роботи, алгоритми constraint-based planning та засоби побудови інтелектуальних веб-систем підтримки планування змін.

Метою кваліфікаційної роботи є розробка системи автоматизованого планування графіків робочих змін із застосуванням алгоритму Min-Conflicts та NLP-компонентів для підтримки взаємодії природною мовою. Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

- проаналізувати предметну галузь та існуючі системи планування змін;
- дослідити особливості constraint-based planning та алгоритму Min-Conflicts;
- спроектувати архітектуру системи автоматизованого планування;
- реалізувати backend та frontend компоненти системи;
- реалізувати механізм формалізації обмежень;
- інтегрувати NLP-модуль на основі OpenAI API;
- реалізувати інтерфейси введення доступності та перегляду графіка;
- забезпечити механізми аналізу навантаження та візуалізації статистичних даних.

Практичне значення роботи полягає у створенні програмної системи, яка дозволяє автоматизувати процес формування графіків змін, зменшити вплив людського фактора, підвищити справедливість розподілу навантаження та спростити взаємодію працівників із системою планування.

У роботі використано методи *constraint satisfaction*, алгоритм локального пошуку *Min-Conflicts*, підходи багатошарової веб-архітектури, *REST API*, технології *frontend-розробки React і TypeScript*, а також *NLP-технології* для обробки текстових повідомлень користувачів.

Структурно кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. У першому розділі виконано аналіз предметної галузі та існуючих рішень. У другому розділі описано проєктування архітектури системи та структури БД. Третій розділ присвячений вибору програмних засобів і технологій. У четвертому розділі розглянуто реалізацію алгоритмічної частини системи, *backend-* та *frontend-*компонентів, а також *NLP-модуля* та користувацького інтерфейсу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

Планування робочих змін є прикладною задачею управління колективом, яка сформувалася разом із розвитком сфер трудової діяльності, що потребують безперервного функціонування. Перші підходи до організації змінної роботи з'явилися у промисловості, де зупинка виробництва була неможливою або економічно недоцільною. Із розвитком інформаційних технологій та сервісів цілодобової підтримки, сфера застосування цих принципів охопила ІТ-інфраструктуру, центри обробки даних, а також служби моніторингу та оперативного реагування. В ІТ-компаніях запровадили посади інженерів NOC (Network Operations Center) та SOC (Security Operations Center), які часто обіймають студенти. Необхідність поєднання роботи з навчанням створює складну систему формальних та неформальних обмежень, що суттєво ускладнює процес формування робочих графіків.

На початкових етапах планування змін здійснювалося вручну з використанням таблиць, що дозволяло оперативно реагувати на зміни, але не забезпечувало масштабованості, системності та автоматизації. Зі збільшенням кількості працівників, ускладненням структури змін та появою додаткових вимог, таких як чергування у нічний час, резервні (standby) зміни та регламентований відпочинок між змінами, існуючі методи стали малоефективними. Це призвело до активного використання автоматизованих скриптів і спеціалізованого програмного забезпечення, однак такі рішення часто мають жорстко зафіксовану логіку та не адаптовані до специфічних умов конкретних команд.

Сучасні команди, що працюють цілодобово, зокрема у сфері мережевих операцій та технічного моніторингу, функціонують в умовах підвищеної складності організації робочого часу. Особливості змінної

роботи, зокрема нічні та резервні (standby) чергування, формують додаткові обмеження щодо доступності персоналу та ускладнюють процес формування робочих графіків. Значна кількість таких обмежень має міжперіодний характер і не завжди може бути формалізована за допомогою простих засобів планування. Крім цього, збір інформації про доступність працівників часто здійснюється у неструктурованому вигляді, що ускладнює її подальшу автоматизовану обробку.

Відсутність автоматизованого зворотного зв'язку та алгоритмів, що могли би пояснити користувачеві певні обмеження, призводить до необхідності додаткової комунікації та збільшує навантаження на осіб, відповідальних за формування графіків. У результаті процес планування перетворюється на трудомістку процедуру, що залежить від людського фактора та не виключає виникнення помилок.

Актуальність теми зумовлена потребою в автоматизації процесу планування змін за наявності складних обмежень, що залежать від режиму роботи та попередніх чергувань. Ця система автоматизує рутинні процеси, мінімізуючи людський чинник, та забезпечує узгодженість графіків з організаційними та регламентними вимогами, що, в свою чергу, дозволяє вивести управління змінною роботою на новий рівень ефективності.

1.2 Аналіз існуючих рішень

1.2.1 Ручні та напівавтоматизовані методи планування

Найпростішим підходом до планування змін є ручне формування графіків із використанням табличних інструментів або текстових документів. Такий підхід ґрунтується на безпосередньому врахуванні доступності працівників та організаційних вимог відповідальною особою.

У невеликих командах за відсутності складних обмежень ручні методи можуть бути достатніми, однак зі зростанням кількості працівників і типів змін їх ефективність суттєво знижується.

Для часткової автоматизації планування змін використовуються скрипти або прості програмні засоби, що працюють із даними, що працівники вносять до таблиці. Відсутність механізмів пояснення обмежень і необхідність ручного втручання обмежують їх ефективність у складних сценаріях змінної роботи.

1.2.2 Програмні системи планування змін

На ринку представлені програмні системи для планування змін, орієнтовані на різні типи організацій і сценарії змінної роботи. Актуальність задачі планування змін зумовила появу як готових прикладних рішень, так і програмних фреймворків, призначених для побудови власних систем планування.

Готові програмні системи планування змін надають завершений робочий процес для кінцевих користувачів і, як правило, включають створення та публікацію графіків, облік доступності працівників, механізми керування змінами, а також засоби комунікації з персоналом. До таких рішень належать «When I Work», «Deputy», «Planday» та подібні платформи. Ці системи орієнтовані на типові сценарії змінної роботи та забезпечують зручність використання за рахунок наявності користувацького інтерфейсу і фіксованої моделі планування.

Поряд із готовими системами існують програмні фреймворки та бібліотеки, призначені для алгоритмічного розв'язання задач планування змін і оптимізації розкладів. До цієї групи належать, зокрема, «Timefold Solver», «OptaPlanner», «Google OR-Tools» та інші інструменти constraint-based planning.

Такі рішення орієнтовані на формалізацію задачі планування і не передбачають прямої взаємодії з кінцевими користувачами. Для їх практичного застосування необхідна розробка прикладного шару, що забезпечує введення обмежень, перегляд результатів і пояснення прийнятих рішень.

1.2.3 Опис існуючих рішень

«When I Work» – це спеціалізована програмна платформа для управління змінними графіками, орієнтована на сегмент малого та середнього бізнесу. Система забезпечує базовий цикл адміністрування персоналу: від формування та публікації розкладів (рисунок 1.1) до обліку доступності працівників і внутрішньої комунікації.

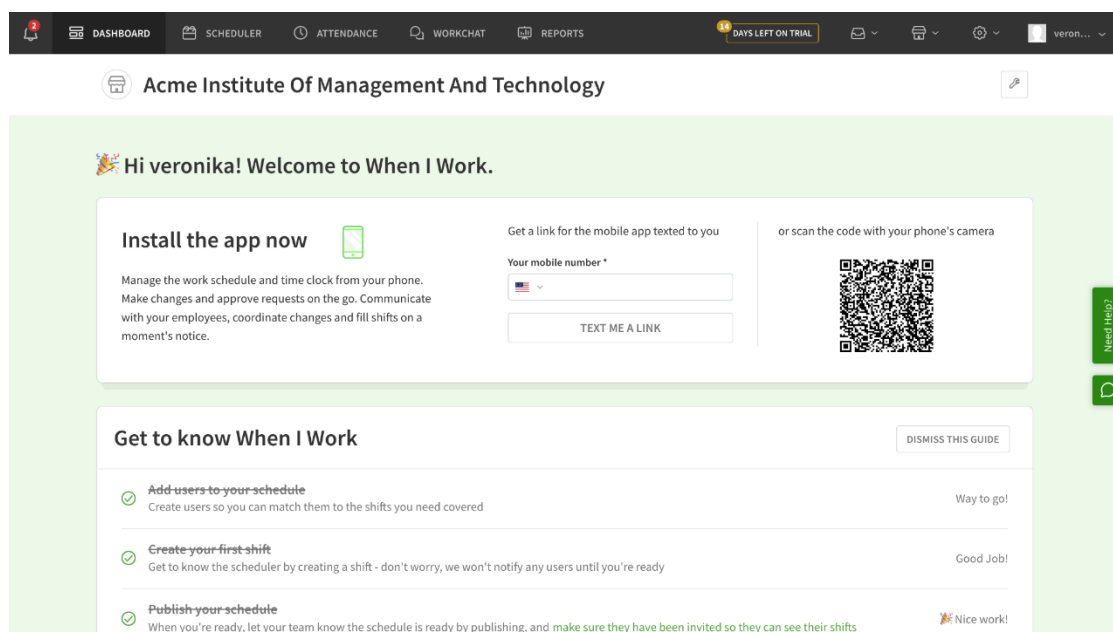


Рисунок 1.1 – Інтерфейс «When I Work»: початок роботи в системі

Проте аналіз функціональних можливостей системи виявив низку критичних обмежень. Алгоритми базуються на фіксованому наборі правил, що не дозволяє враховувати складні міжперіодні залежності. Зокрема,

система не аналізує вплив навантаження попереднього періоду на допустимість призначень у наступному, що робить її малоефективною для організацій із нестандартними регламентами.

Працівник обмежений лише можливістю позначення бажаних змін, тоді як логіка розподілу залишається закритою та незмінною. Процес часто передбачає директивне призначення змін адміністратором (рисунок 1.2), змушуючи персонал адаптуватися до заздалегідь визначеної моделі замість участі у формалізації правил.

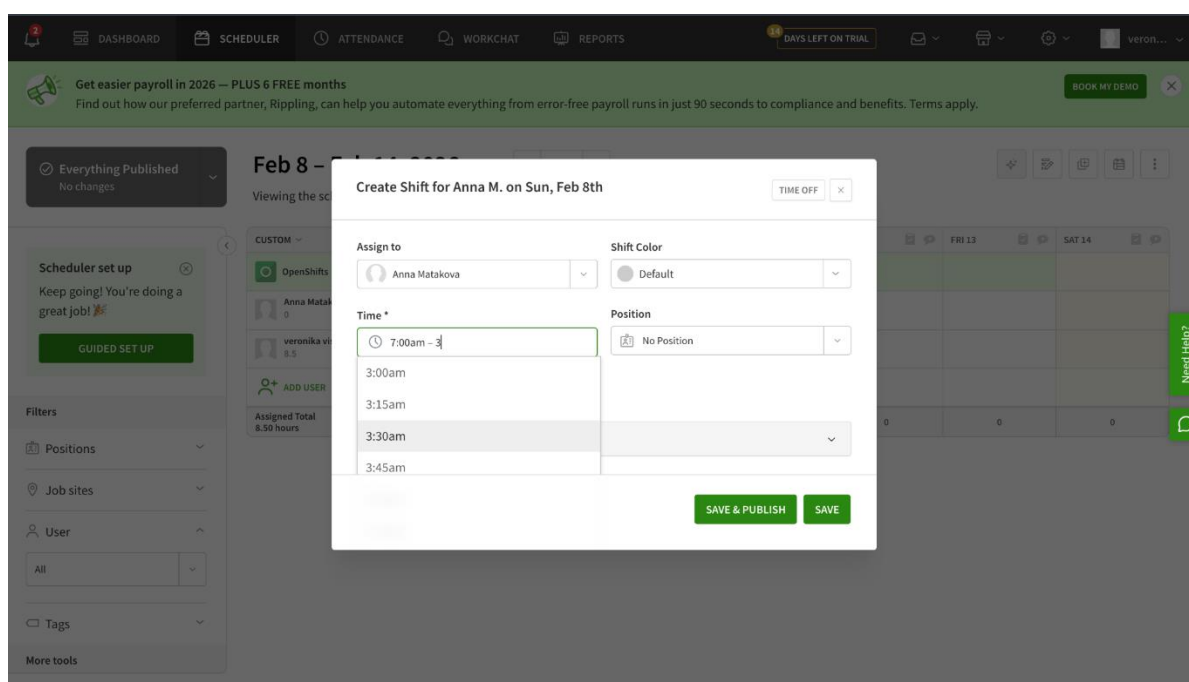


Рисунок 1.2 – Інтерфейс «When I Work»: створення зміни для працівника

Інтегрований у систему чат-бот (рисунок 1.3) виконує виключно інформаційно-навігаційну функцію. Як демонструє аналіз взаємодії, він не має доступу до функціонального ядра та не здатний виконувати операційні завдання, наприклад, автоматизоване коригування графіка за запитом.

На відміну від розглянутого рішення, запропонований у цій роботі підхід передбачає інтеграцію індивідуальних обмежень та побажань користувача безпосередньо у модель планування. Це дозволяє алгоритму

динамічно адаптувати розподіл змін, забезпечуючи гнучкість, прозорість та високу відповідність результату реальним потребам персоналу.

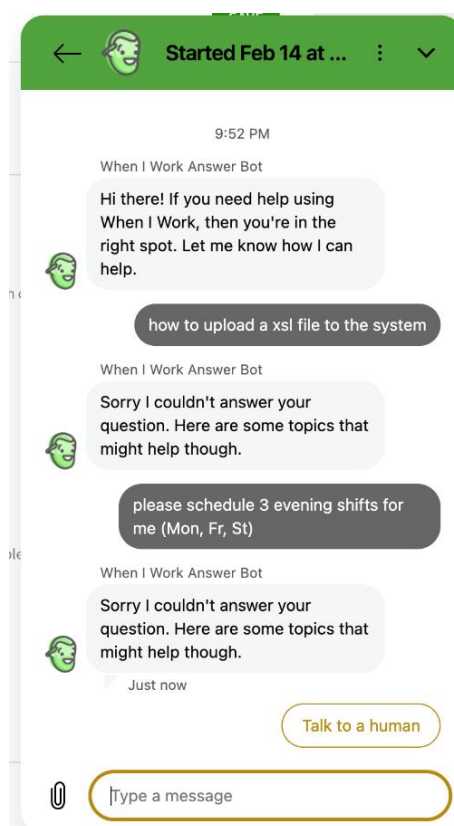


Рисунок 1.3 – Інтелектуальний помічник «When I Work»

«Deputy» – це комплексна платформа, що поєднує інструменти планування графіків із функціями обліку робочого часу (рисунок 1.4 та рисунок 1.5). Система підтримує базову автоматизацію створення розкладів на основі доступності персоналу та операційного навантаження. Однак використання даної платформи для специфічних задач, таких як підтримка безперервної IT-інфраструктури, виявило недоліки. Зокрема, система має обмежені можливості врахування складних наборів обмежень, пов'язаних із чергуванням змін, рівномірним розподілом навантаження між працівниками та дотриманням індивідуальних побажань персоналу. Крім того, платформа не забезпечує достатньої гнучкості при адаптації алгоритмів планування до специфічних вимог організації, що ускладнює її

використання для задач із високим рівнем залежностей та динамічними умовами планування.

Архітектура «Deputy» оптимізована під сценарії ритейлу, складської логістики та сфери послуг. Це робить систему менш адаптивною до складних режимів чергування, характерних для технічних служб підтримки.

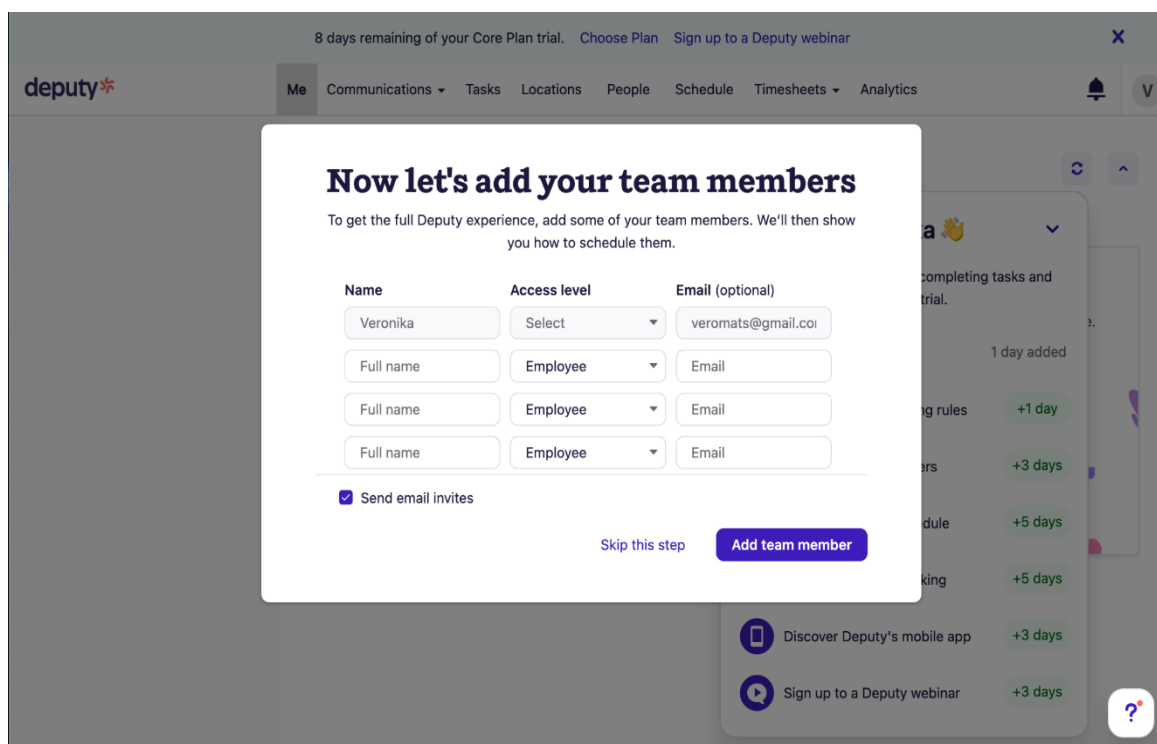


Рисунок 1.4 – Інтерфейс «Deputy»

Це ускладнює інтеграцію специфічних правил формування графіків та адаптацію системи до внутрішніх процесів організації. Крім того, використання хмарної інфраструктури платформи може створювати додаткові вимоги щодо захисту службових даних та залежність від стабільності зовнішніх сервісів.

Функція автоматичного генерування графіків працює за жорсткими алгоритмами, які не передбачають глибокого налаштування (рисунок 1.6). Користувач не має можливості вносити зміни до структури обмежень або розширювати модель планування власними правилами.

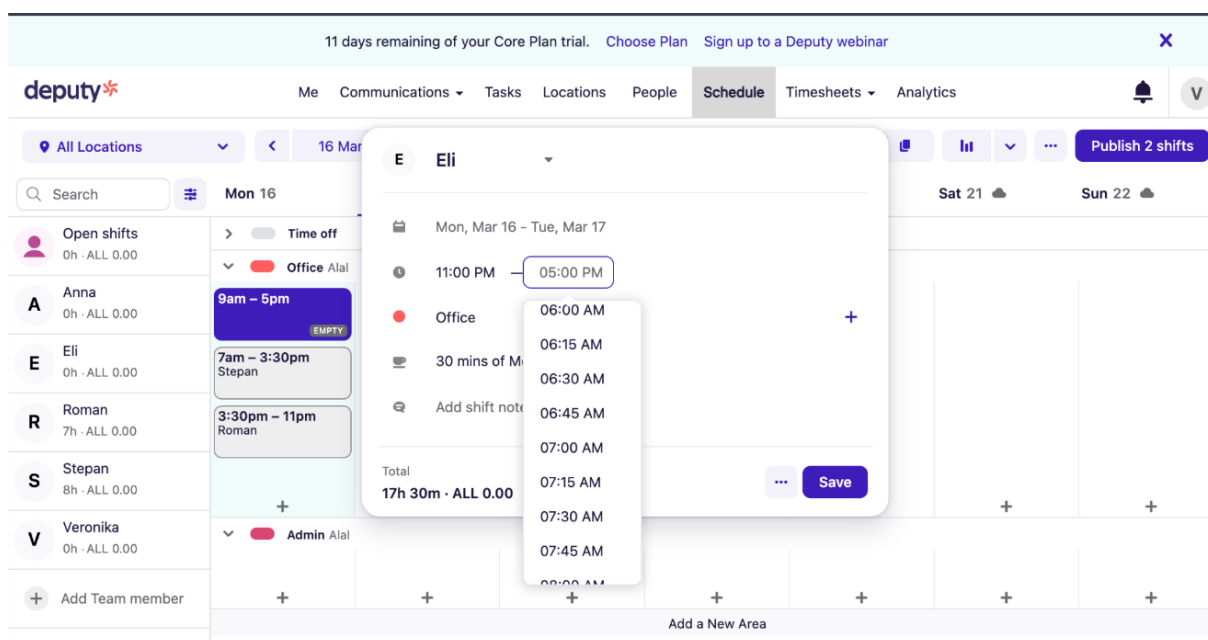


Рисунок 1.5 – Інтерфейс «Deputy»: створення графіку вручну

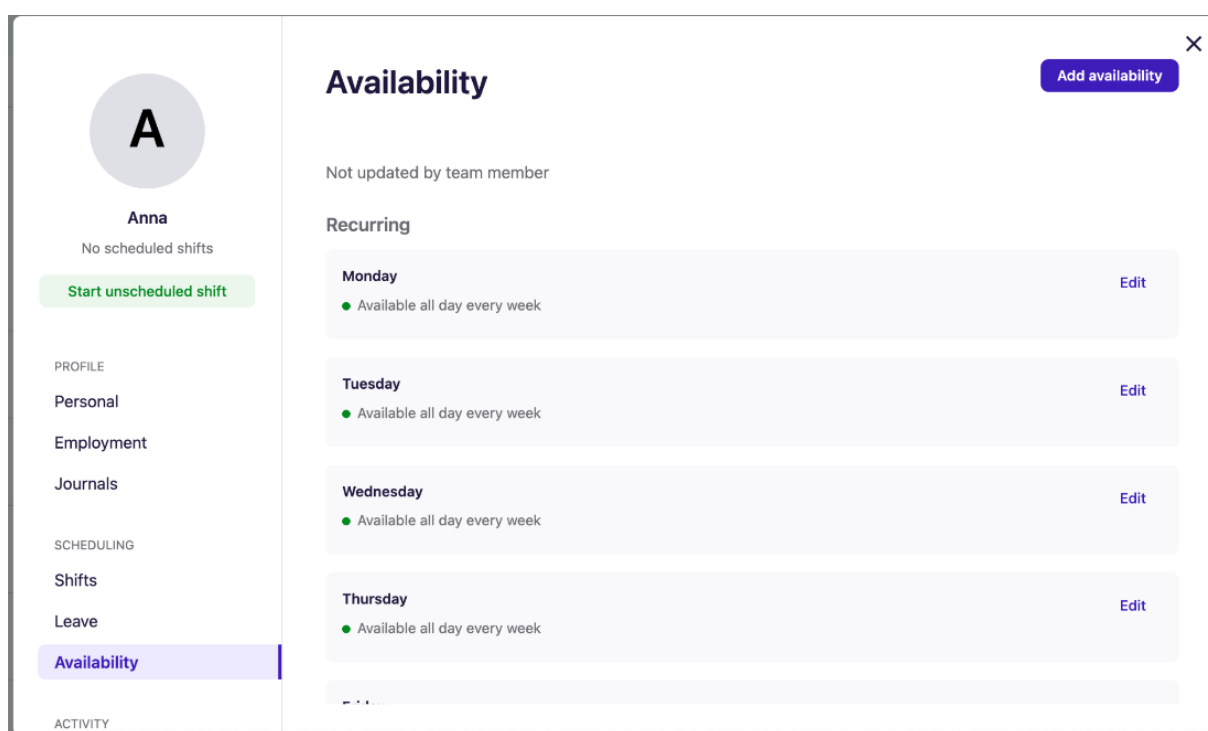


Рисунок 1.6 – Особистий кабінет у «Deputy»

Платформа не дозволяє ефективно моделювати міжперіодні залежності або специфічні умови для нічних і резервних змін. Зокрема,

у системі відсутні механізми врахування індивідуальних уподобань як частини математичної моделі розподілу.

Таким чином, незважаючи на розвинені інтерфейсні рішення, «Deputy» залишається інструментом для типових задач управління персоналом, що не забезпечує необхідної гнучкості для реалізації складних інтелектуальних моделей планування.

«Planday» – це хмарна платформа для управління персоналом, розроблена для організацій із базовими потребами у змінних розкладах. Система забезпечує зручне середовище для візуального редагування графіків, обміну повідомленнями та синхронізації робочих календарів (рисунк 1.7). Основна увага в платформі приділяється спрощенню процесу координації персоналу та зменшенню адміністративного навантаження під час формування змін.

	Mon, Mar 16	Today	Wed, Mar 18	Thu, Mar 19	Fri, Mar 20	Sat, Mar 21	Sun, Mar 22
0 Shifts	0 Shifts	0 Shifts	1 Shift	0 Shifts	0 Shifts	0 Shifts	0 Shifts
ver vie			monitoring team 09:00 - 17:00				
Anna Siu							
Gary Shevchenko							
Roman Korz							
Anna Popova							
Eli Copter							
HOURS	8h 0m	0h 0m	0h 0m	8h 0m	0h 0m	0h 0m	0h 0m
PAYROLL	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00
REVENUE (FORECAST)	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00
REVENUE (ACTUAL)	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00	ALL 0.00
PAYROLL PERCENTAGE	Actual: 0%	0% (0%)	0% (0%)	0% (0%)	0% (0%)	0% (0%)	0% (0%)
Target: 0%	Forecast: 0%						

Рисунок 1.7 – Інтерфейс «Planday»

Платформа надає користувачеві можливість самостійно формувати шаблони графіків та визначати структуру змін (рисунк 1.8).

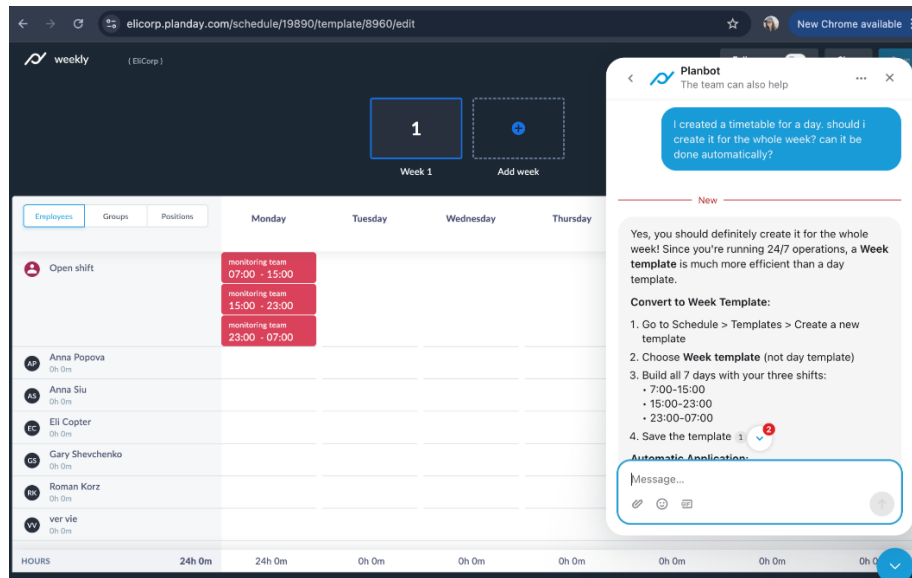


Рисунок 1.8 – Створення шаблону графіку в «Planday»

Попри розвинені комунікаційні можливості, система має суттєві обмеження в контексті інтелектуального планування (рисунок 1.9).

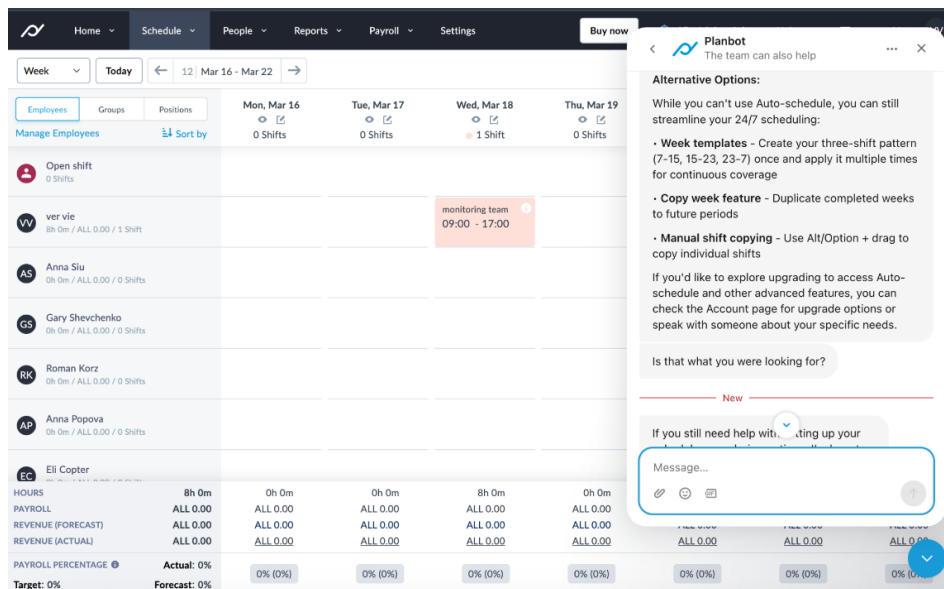


Рисунок 1.9 – Створення графіку в «Planday»

У «Planday» не реалізовано механізми автоматичного генерування графіків. Усі дані, призначення та перевірки на відповідність правилам

здійснюються користувачем вручну, що значно підвищує трудовитрати та ризик виникнення помилок при масштабуванні команди.

Функціонал системи орієнтований на стандартні, циклічні сценарії роботи. Вона не підтримує глибокої формалізації складних обмежень, як-от динамічний аналіз історії змін або облік специфічних міжперіодних взаємозв'язків. Хоча система дозволяє персоналу вказувати свої вподобання, вони мають виключно рекомендаційний характер для менеджера. Відсутня можливість інтеграції цих побажань безпосередньо в логіку розподілу ресурсів через математичні ваги чи пріоритети.

Таким чином, «Planday» позиціонується як інструмент цифрової фіксації графіків, а не як система підтримки прийняття рішень. Це робить її малоефективною для складних інфраструктурних об'єктів, де критично важливою є автоматична оптимізація розкладу з урахуванням безлічі змінних.

«Timefold Solver» – це фреймворк, що надає можливість формалізувати та розв'язувати задачі планування змін як задачі розподілу з обмеженнями. Він дозволяє моделювати велику кількість правил і здійснювати пошук допустимих або оптимальних рішень відповідно до заданих критеріїв. Водночас «Timefold» не є готовою прикладною системою. Його використання передбачає розробку візуального інтерфейсу шару, що забезпечує взаємодію з користувачами та інтерпретацію результатів планування..

«OptaPlanner» – це Java/Kotlin фреймворк для розв'язання задач оптимізації з обмеженнями. «OptaPlanner» дозволяє формалізувати задачу планування змін у вигляді набору обмежень (constraints) і цільових функцій, після чого здійснює пошук допустимих або оптимальних розкладів із урахуванням заданих правил. Фреймворк широко використовується у задачах розкладання, балансування навантаження, призначення ресурсів та інших оптимізаційних сценаріях, де існує велика кількість взаємопов'язаних обмежень. Використання даного фреймворку вимагає

розроблення користувацького інтерфейсу, що забезпечить ввід даних, взаємодію з користувачем та інтерпретацію результатів.

«Google OR-Tools» – це бібліотека відкритого коду, розроблена компанією «Google» для розв’язання широкого класу задач оптимізації, включаючи задачі маршрутизації, розподілу ресурсів і розкладів. Вона надає набір алгоритмів для роботи з лінійним і цілочисельним програмуванням, задачами з обмеженнями та іншими підходами оптимізації, що дозволяє формалізувати та ефективно розв’язувати складні комбінаторні задачі. У контексті планування робочих змін «Google OR-Tools» може бути використана для побудови моделей, які враховують різноманітні обмеження і здійснюють пошук допустимих або оптимальних розкладів відповідно до цільових критеріїв. «Google OR-Tools» представляє собою інструмент для побудови оптимізаційних компонентів у системах планування змін, але, як і інші оптимізаційні бібліотеки, потребує створення візуального інтерфейсу.

1.3 Порівняльний аналіз

Проаналізовані програмні рішення забезпечують автоматизацію планування змін у межах заздалегідь визначених моделей і є ефективними для типових сценаріїв змінної роботи. Водночас їх спільною рисою є використання фіксованої структури правил, у межах якої працівник може лише звітувати про доступність, але має доступу до індивідуальних налаштувань, що враховують його уподобання (наприклад, допустимість кількох нічних змін підряд). Це обмежує можливість гнучкого врахування нестандартних вимог у процесі формування графіка роботи.

Таким чином, існуючі програмні рішення не забезпечують достатньої гнучкості у підтримці безперервної роботи, що обґрунтовує доцільність розробки системи підтримки планування змін, орієнтованої на поєднання

алгоритмічних методів із можливістю налаштування правил на рівні окремого працівника.

Результати аналізу продемонстровані у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз систем

	Гнучкість правил	Участь працівника	Міжперіодна пам'ять	Готовність до роботи
«When I Work»	—	—	+/-	+
«Deputy»	—	—	+/-	+
«Planday»	—	—	—	+
«OptaPlanner»	+/-	+/-	+/-	—
«Google OR-Tools»	+/-	+/-	+/-	—
«Timefold Solver»	+/-	+/-	+/-	—
Кваліфікаційний проєкт	+	+	+	+

1.4 Постановка задачі

Метою даного кваліфікаційного проєкту є розробка системи підтримки планування змін для команд, що працюють цілодобово, з урахуванням складних обмежень змінної роботи та необхідності прозорого пояснення рішень кінцевим користувачам.

Особливістю цієї задачі є етап формалізації індивідуальних уподобань працівників, де суб'єктивні запити перетворюються на математичні ваги для подальшої обробки. Використання алгоритму Min-Conflicts дозволяє реалізувати систему інтелектуальної підтримки прийняття рішень, яка одночасно оперує двома типами вхідних даних: жорсткими обмеженнями (нормативні типи змін, обов'язкові резерви, законодавчі

ліміти робочого часу) та м'якими чинниками (персональні побажання, історія навантаження у попередніх періодах). Такий підхід забезпечує високу точність розрахунків та дозволяє системі стабільно функціонувати в умовах великої кількості змінних, зберігаючи високу швидкість обробки даних незалежно від масштабу організаційної структури.

Успішне втілення поставленої задачі передбачає виконання наступних завдань:

- збір, моделювання та формалізація побажань працівників щодо робочого часу;
- автоматизована перевірка коректності заданих обмежень з урахуванням історії чергувань;
- алгоритмічний розподіл змін відповідно до заданих правил;
- інформування користувачів про неможливість призначення окремих змін із поясненням причин таких обмежень.

Результатом виконання роботи є програмна система підтримки планування змін, яка реалізує формалізовану модель обмежень змінної роботи, забезпечує автоматизовану перевірку конфліктів і дозволяє отримувати коректні графіки з урахуванням міжперіодних залежностей. Розроблена система має слугувати прикладною інтеграцією алгоритмічних методів планування з механізмами взаємодії з користувачем та пояснення прийнятих рішень.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Функціональні вимоги до системи

Система автоматизованого планування графіків змін надає функціональні можливості, спрямовані на підтримку процесу розподілу робочого навантаження між працівниками в умовах складних обмежень та змінного режиму роботи. Архітектура системи передбачає розподіл функцій відповідно до ролей користувачів – працівника та адміністратора, що забезпечує розмежування доступу до операцій планування та контролю.

Базовою функціональною можливістю системи є управління обліковими записами, що включає реєстрацію користувачів, автентифікацію та подальший доступ до персоналізованого інтерфейсу. Після входу в систему користувач отримує можливість взаємодіяти з функціоналом відповідно до своєї ролі.

Для працівника ключовою функцією є внесення даних для планування. Це включає введення інформації про доступність у вигляді часових інтервалів, задання періодів відсутності (відпустка, армійська служба), а також формування індивідуальних побажань щодо типів і режиму змін. Особливістю системи є підтримка двох способів введення даних: структурованого через календарні форми та неструктурованого через ШІ чат-бот природною мовою. У другому випадку система виконує інтерпретацію текстових запитів користувача з подальшим перетворенням їх у формалізовані параметри моделі.

У системі реалізовано механізм автоматичної інтерпретації введених даних, у межах якого побажання трансформуються у м'які обмеження з відповідними пріоритетами. Це дозволяє уникнути перевантаження користувача складними налаштуваннями та водночас забезпечує достатню точність моделювання його переваг.

Адміністратор керує процесом планування робочих змін. До його можливостей належить перевірка повноти введених даних, налаштування обмежень (наприклад, мінімальний час відпочинку, кількість змін певного типу), а також ініціація процесу генерації графіка. Після виконання алгоритму адміністратор аналізує отриманий результат та приймає рішення щодо його затвердження або повторної генерації.

Центральним функціональним компонентом системи є модуль автоматичного планування, реалізований із використанням алгоритму Min-Conflicts. Планування формалізується як задача задоволення обмежень, у якій кожна зміна розглядається як змінна, а можливі призначення працівників – як її значення. Обмеження поділяються на жорсткі (регламентовані правила, часові інтервали, нормативи) та м'які (індивідуальні побажання, історія навантаження).

Алгоритм Min-Conflicts здійснює ітеративний пошук рішення, мінімізуючи кількість порушених обмежень шляхом локального перепризначення змін. На кожному кроці обирається змінна, що перебуває у конфлікті, після чого визначається значення, яке мінімізує кількість конфліктів у системі. Такий підхід забезпечує високу обчислювальну ефективність і дозволяє працювати з великими обсягами даних у реальному часі.

Результатом роботи алгоритму є сформований графік змін, який відображається у вигляді структурованої таблиці та доступний для перегляду як у загальному вигляді, так і у персоналізованому форматі для кожного працівника. Додатково система виконує аналіз результату з метою виявлення незаповнених змін та ініціює взаємодію з користувачами шляхом надсилання відповідних повідомлень.

Окремою функціональною можливістю є забезпечення прозорості роботи алгоритму. Система надає пояснення прийнятих рішень, зокрема у випадках неврахування побажань користувача або виникнення конфліктів

обмежень. Це реалізується шляхом відображення причинно-наслідкових зв'язків між заданими умовами та отриманим результатом.

Таким чином, функціональні вимоги системи охоплюють повний цикл планування змін: від збору та формалізації даних до їх алгоритмічної обробки із застосуванням методу Min-Conflicts і подальшого представлення результатів із можливістю аналізу та пояснення.

2.2 Вибір архітектури системи

Під час проєктування системи автоматизованого планування графіків змін необхідно обрати архітектурний підхід, який забезпечить коректну реалізацію основного функціоналу, зручність супроводу та можливість подальшого розвитку програмного продукту. Для цього доцільно розглянути три поширені архітектурні підходи: монолітну, мікросервісну та багатошарову архітектуру.

Монолітна архітектура передбачає реалізацію всіх основних компонентів системи в межах єдиного застосунку. Її перевагами є відносна простота розробки, єдине середовище виконання, зручність налагодження та спрощене розгортання. Для систем, що мають обмежену предметну галузь і не потребують незалежного масштабування великої кількості компонентів, монолітний підхід є практичним рішенням. Водночас у класичному моноліті з часом ускладнюється підтримка коду, оскільки логіка інтерфейсу, бізнес-правила, обробка даних і допоміжні модулі можуть втрачати чітке розмежування.

Мікросервісна архітектура ґрунтується на поділі системи на набір незалежних сервісів, кожен із яких реалізує окрему функцію. Її перевагою є можливість незалежного масштабування компонентів, ізольованого розгортання та технологічної гнучкості. Такий підхід доцільний для великих розподілених систем із високим навантаженням, складною інтеграцією або потребою в автономному розвитку окремих підсистем.

Однак для системи планування змін мікросервісна архітектура на початковому етапі є надмірно складною. Її використання потребує додаткової інфраструктури для взаємодії сервісів, керування мережевими викликами, синхронізації даних, моніторингу та забезпечення відмовостійкості. Це підвищує складність реалізації без пропорційного виграшу для задачі, що має відносно цілісну предметну логіку.

Багатошарова архітектура базується на поділі системи на логічні рівні, кожен із яких відповідає за окремий аспект обробки даних. Зазвичай виділяють рівень представлення, рівень прикладної логіки, рівень доменної або алгоритмічної обробки та рівень доступу до даних. Основною перевагою такого підходу є чітке розмежування відповідальності між компонентами, що підвищує зрозумілість системи, полегшує тестування та спрощує модифікацію окремих частин без суттєвого впливу на інші. Для інтелектуальної системи планування змін це особливо важливо, оскільки алгоритм Min-Conflicts, механізми формалізації побажань, модуль пояснення рішень і засоби відображення результатів мають різну функціональну природу та повинні бути відокремлені один від одного.

З урахуванням особливостей предметної галузі для даної системи найбільш доцільною є багатошарова монолітна архітектура. Такий вибір поєднує переваги монолітного підходу на рівні розгортання і супроводу з перевагами багатошарового підходу на рівні внутрішньої організації системи. Для задачі автоматизованого планування змін це є виправданим, оскільки система працює в межах єдиної предметної галузі, а її ключові компоненти тісно пов'язані між собою: введення доступності, обробка побажань, генерація графіка, аналітика та пояснення результатів формують єдиний логічний цикл.

У межах обраної архітектури доцільно виділити такі основні шари. Рівень представлення забезпечує взаємодію з працівником і адміністратором через веб-інтерфейс або ШІ-чат. На цьому рівні реалізуються форми введення доступності, перегляд графіка, аналітичні

панелі та повідомлення системи. Прикладний рівень координує виконання сценаріїв роботи, обробляє запити користувачів, ініціює запуск алгоритму та формує відповіді для інтерфейсу.

Доменний рівень містить основну логіку системи: моделі працівників, змін, обмежень, правила перевірки коректності даних, модуль формалізації побажань і алгоритмічний компонент Min-Conflicts. Рівень доступу до даних відповідає за збереження облікових записів, доступності, історії змін, налаштувань та результатів планування у БД.

Актуальність саме такої архітектури зумовлена тим, що система повинна не лише зберігати та відображати дані, а й виконувати інтелектуальну обробку обмежень.

Алгоритм Min-Conflicts доцільно розмістити в окремому доменному модулі, ізольованому від інтерфейсної частини. Це дозволяє змінювати або вдосконалювати алгоритм планування без перебудови всієї системи. Аналогічно, механізм пояснення рішень має бути пов'язаний із доменною логікою, а не з UI, оскільки пояснення повинні формуватися на основі фактичних причин конфліктів і обмежень.

Отже, у результаті порівняльного аналізу можна зробити висновок, що для системи автоматизованого планування графіків змін із застосуванням алгоритму Min-Conflicts найбільш придатною є багат шарова монолітна архітектура.

Монолітний формат забезпечує простоту реалізації та розгортання, тоді як багат шаровий поділ дозволяє структуровано організувати інтерфейсну, прикладну, алгоритмічну та дану логіку.

Саме тому для системи такого типу доцільно зупинитися на багат шаровій архітектурі з монолітним способом реалізації як на рішенні, що найкраще відповідає поточним функціональним і технологічним вимогам.

2.3 Проєктування архітектури системи

Проєктування архітектури системи автоматизованого планування графіків змін доцільно виконувати не лише на рівні структурних компонентів, а й на рівні сценаріїв взаємодії користувачів із системою. Для цього використовуються діаграми, що дозволяють відобразити систему з різних точок зору: на рівні зовнішніх функцій, послідовності дій користувача та внутрішнього бізнес-процесу обробки даних. У межах даної роботи для опису архітектури застосунку доцільно використати діаграму варіантів використання (Use Case Diagram), схему користувацького потоку (User Flow) та activity diagram процесу формування графіка.

Таке поєднання є виправданим, оскільки кожна зі схем виконує окрему функцію. Діаграма Use Case показує, які саме функціональні можливості надаються різним типам користувачів. User Flow відображає реальну послідовність взаємодії користувача з інтерфейсом. Activity Diagram дозволяє формалізувати внутрішній процес роботи системи, включаючи етапи збору даних, алгоритмічної обробки та прийняття рішення. Разом ці схеми формують цілісне уявлення про логіку функціонування застосунку.

2.3.1 Діаграма варіантів використання (Use Case Diagram)

Діаграма варіантів використання призначена для відображення основних функціональних можливостей системи та ролей користувачів, які взаємодіють із нею.

Для системи автоматизованого планування графіків змін доцільно виділити двох основних акторів: працівника та адміністратора. Саме така рольова модель реалізована у проєкті: працівник відповідає за введення персональних даних, пов'язаних із доступністю та побажаннями щодо змін,

тоді як адміністратор здійснює контроль вхідної інформації, запускає процес генерації графіка та виконує його затвердження.

У межах діаграми працівник взаємодіє з такими сценаріями: реєстрація та авторизація в системі, введення доступності, зазначення періодів відсутності, формування побажань щодо змін, перегляд персонального та загального графіка, отримання повідомлень про незакриті зміни, вибір доступної зміни та перегляд пояснень щодо прийнятих системою рішень.

Перелічені сценарії відповідають функціонально-компонентній структурі користувацької частини системи, яка охоплює введення вхідних даних, взаємодію з результатами планування та механізми оперативного реагування на незаповнені зміни.

Адміністратор взаємодіє з окремою групою функціональних сценаріїв, що пов'язані з управлінням процесом формування графіка. До них належать перевірка повноти та коректності вхідних даних, налаштування обмежень планування, запуск алгоритму генерації графіка, перегляд побажань працівників, аналіз сформованого розкладу, контроль незакритих змін, а також затвердження графіка або ініціація його повторного формування. Таким чином, адміністратор виконує роль координатора процесу планування та контролює відповідність результатів заданим вимогам і обмеженням.

Окремим ключовим прецедентом системи є сценарій «Сформувати графік змін», який об'єднує процеси перевірки обмежень, формалізації побажань працівників та запуску алгоритму Min-Conflicts. Виділення цього прецеденту є важливим, оскільки алгоритмічна складова виступає центральним елементом системи автоматизованого планування.

У межах цього процесу введені користувачами дані перетворюються на набір формалізованих параметрів, після чого система здійснює побудову графіка з одночасним урахуванням жорстких і м'яких обмежень (рисунк 2.1). Це дозволяє забезпечити баланс між

організаційними вимогами, доступністю персоналу та індивідуальними побажаннями працівників.

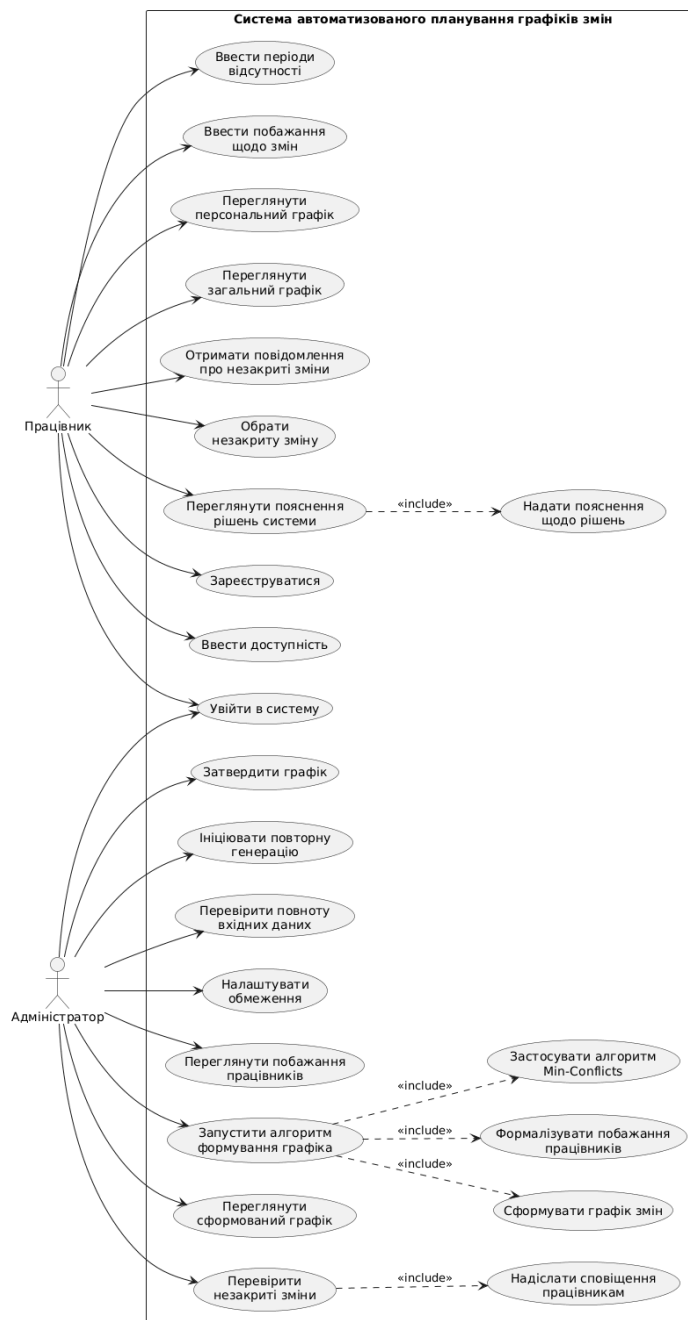


Рисунок 2.1 – Діаграма Use Case користування системою

На діаграмі відображено двох основних акторів системи: працівника та адміністратора. Працівник вводить доступність, періоди відсутності та побажання щодо змін, переглядає сформований графік, отримує

повідомлення про незакриті зміни та може переглядати пояснення рішень системи. Адміністратор контролює повноту введених даних, налаштовує обмеження, запускає алгоритм формування графіка, аналізує результат і приймає рішення щодо його затвердження або повторної генерації. Окремо показано включені підпроцеси: формалізацію побажань, застосування алгоритму Min-Conflicts, формування графіка та надсилання сповіщень.

Таким чином, Use Case Diagram дає узагальнене уявлення про зовнішню функціональність застосунку та показує, як саме ролі користувачів співвідносяться з функціями системи.

2.3.2 Схема користувацького потоку (User Flow)

Схема User Flow використовується для опису послідовності дій користувача під час взаємодії з інтерфейсом системи. На відміну від Use Case Diagram, вона відображає не лише перелік доступних функцій, а й логіку переходів між екранами, точки прийняття рішень та можливі розгалуження сценаріїв взаємодії. Такий підхід дозволяє детальніше описати поведінку користувача в межах системи та структурувати основні сценарії роботи з інтерфейсом.

Для працівника користувацький потік починається з авторизації в системі, після чого відкривається сторінка введення доступності. На цьому етапі працівник може надати інформацію двома способами: через структурований інтерфейс у вигляді календаря або за допомогою чат-бота з використанням природної мови. Така організація взаємодії забезпечує більш гнучкий спосіб введення даних та спрощує процес заповнення інформації про доступність.

Після збереження введених даних система виконує їх обробку та використовує під час генерації графіка змін. Коли процес формування розкладу завершено, працівник отримує відповідне повідомлення та переходить до сторінки перегляду графіка, де може ознайомитися

з призначеними змінами. У разі виявлення незакритих змін система додатково надсилає сповіщення із пропозицією обрати доступну зміну для покриття. Після підтвердження вибору графік автоматично оновлюється, а зміни синхронізуються між усіма користувачами системи (рисунок 2.2).

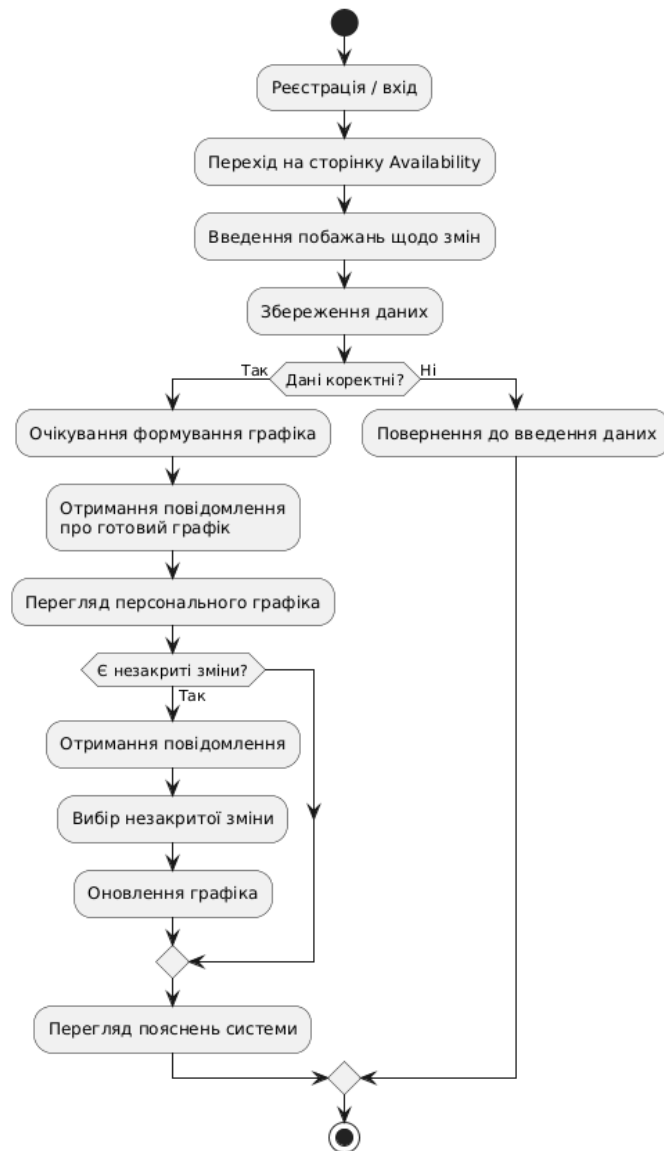


Рисунок 2.2 – Діаграма User Flow для працівника

Для адміністратора користувацький потік має іншу логіку (рисунок 2.3). Після входу в систему адміністратор переходить до панелі керування, де перевіряє, чи всі працівники ввели необхідні дані. Далі

він налаштовує або уточнює обмеження, наприклад, вводить обмеження, після чого запускає алгоритм формування графіка.

Після завершення генерації система відображає результати у вигляді таблиці побажань та попереднього розкладу. Наступною точкою потоку є прийняття рішення: якщо результат задовольняє вимоги, адміністратор затверджує графік; якщо ні, ініціює повторну генерацію. У випадку наявності незакритих змін система запускає додатковий цикл інформування працівників про потребу закрити окремі зміни.

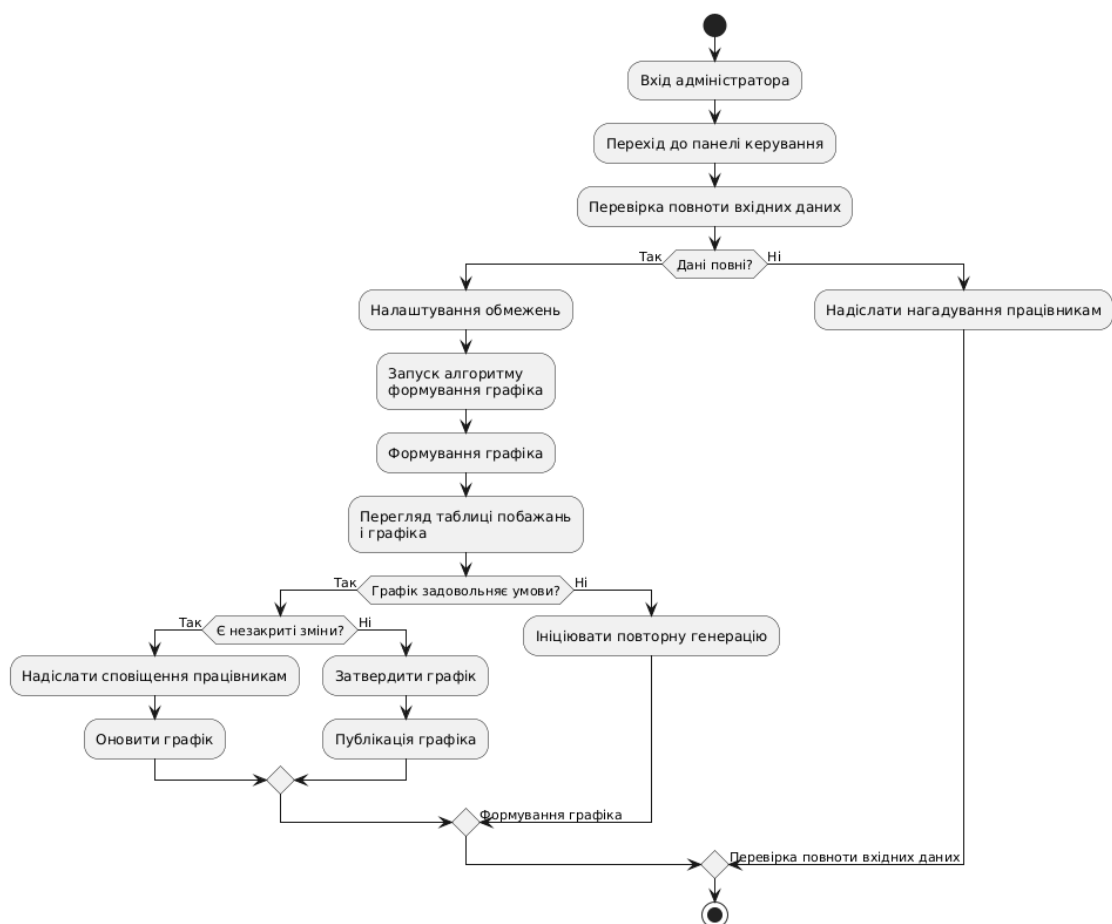


Рисунок 2.3 – Діаграма User Flow для адміністратора

Описана схема User Flow є важливою з погляду архітектури застосунку, оскільки дозволяє пов'язати окремі інтерфейсні сторінки з бізнес-логікою системи. Вона також демонструє, що взаємодія не

є лінійною: система підтримує циклічний сценарій, прив'язаний до двотижневого періоду планування, з можливістю повторної генерації графіка та подальшого доукомплектування незакритих змін.

2.3.3 Activity Diagram – BPwin-діаграма бізнес-процесу

Activity Diagram або BPwin-діаграма доцільна для опису внутрішнього процесу функціонування системи. Якщо Use Case і User Flow концентруються переважно на зовнішній взаємодії, то activity diagram відображає логіку проходження даних через функціональні етапи системи (додаток А – рисунок А.1).

Початковою подією процесу є відкриття нового циклу планування. Після цього система переходить до етапу збору вхідних даних: працівники вносять доступність, періоди відсутності та індивідуальні побажання. Далі виконується перевірка повноти введених даних. Якщо виявлено, що частина користувачів не надала необхідну інформацію, система надсилає нагадування та повертає процес до етапу очікування введення. Якщо дані є достатніми, адміністратор ініціює запуск алгоритму.

На наступному етапі відбувається формалізація даних: текстові або структуровані побажання інтерпретуються та перетворюються на набір обмежень і параметрів моделі. Далі система виконує алгоритм Min-Conflicts, який намагається побудувати графік із мінімальною кількістю конфліктів між жорсткими та м'якими обмеженнями. Після завершення алгоритмічної обробки формується попередній варіант графіка, який передається адміністратору для перевірки.

У межах діаграми передбачена точка прийняття рішення: якщо графік задовольняє вимоги, адміністратор його затверджує, після чого система публікує результат для всіх користувачів. Якщо графік не відповідає очікуванням або містить критичні недоліки, процес повертається до етапу налаштування параметрів і повторної генерації. Після затвердження графіка

система перевіряє наявність незакритих змін. Якщо такі зміни існують, запускається додатковий підпроцес сповіщення доступних працівників, отримання їхніх відповідей та оновлення розкладу. Завершальним етапом є надання користувачам доступу до остаточного графіка та пояснень щодо прийнятих рішень.

З точки зору VRwin-моделі цей процес можна подати як послідовність функціональних блоків: збір даних, перевірка повноти, формалізація обмежень, генерація графіка, аналіз результату, затвердження або повторна генерація, обробка незакритих змін, публікація підсумкового розкладу. Така декомпозиція є корисною, оскільки дозволяє відокремити інтерфейсні операції від алгоритмічного ядра та допоміжних сервісів сповіщення.

Діаграма варіантів використання фіксує перелік функцій, доступних працівникові та адміністратору. User Flow відображає маршрути руху користувача між екранами та точками прийняття рішень. Activity Diagram або VRwin-діаграма описує внутрішній бізнес-процес формування графіка змін, включаючи збір даних, формалізацію обмежень, застосування алгоритму Min-Conflicts та публікацію результату.

У сукупності ці схеми дозволяють повноцінно описати архітектуру застосунку на концептуальному рівні та показати зв'язок між користувацькою взаємодією, функціональними можливостями системи й алгоритмічним ядром планування. Це сприяє кращому розумінню структури програмного забезпечення, спрощує подальше проектування компонентів системи та забезпечує узгодженість між функціональними вимогами й реалізованими механізмами планування графіків змін.

2.4 Проектування бази даних

База даних є впорядкованою сукупністю взаємопов'язаних даних, організованих таким чином, щоб забезпечити їх ефективне зберігання, обробку та доступ у межах інформаційної системи. У розробленій системі

автоматизованого планування змін БД виконує роль централізованого сховища вхідних даних, введених користувачами, а також результатів формування графіка.

Проектування БД виконано з урахуванням принципу мінімальної достатності, відповідно до якого до моделі включено лише ті сутності, що безпосередньо використовуються в процесі збору даних і формування графіка змін. Такий підхід дозволяє уникнути надлишковості структури та забезпечує чітке розмежування відповідальності між елементами системи.

Для забезпечення коректності організації даних у кожній таблиці використовується первинний ключ (Primary Key), атрибут або набір атрибутів, що однозначно ідентифікує кожен запис. Первинний ключ не допускає повторень і не може містити невизначених значень, що гарантує унікальність збереженої інформації.

Оскільки дані розподілені між кількома взаємопов'язаними таблицями, для встановлення зв'язків між ними застосовуються зовнішні ключі (Foreign Key). Зовнішній ключ – це атрибут, який посиляється на первинний ключ іншої таблиці та забезпечує узгодженість зв'язків між сутностями. Використання зовнішніх ключів дозволяє підтримувати цілісність даних і коректну взаємодію між різними частинами БД.

У результаті аналізу функціональних вимог системи визначено три основні сутності, які формують структуру БД:

- користувачі системи;
- дані про доступність працівників;
- результати призначення змін.

Відповідно до цього БД містить такі таблиці:

- users,
- availability,
- schedule_assignments.

2.4.1 Таблиця users

Таблиця users призначена для зберігання інформації про користувачів системи. Ця таблиця містить наступні поля:

- user_id – первинний ключ;
- full_name – ім'я користувача;
- email – адреса електронної пошти;
- password_hash – хеш пароля;
- role – роль користувача (працівник або адміністратор).

Таблиця використовується для реалізації механізмів автентифікації та розмежування доступу до функціональних можливостей системи. Роль користувача визначає доступ до операцій введення даних або управління процесом планування.

2.4.2 Таблиця availability

Таблиця availability зберігає вхідні дані, які вводяться працівниками перед формуванням графіка. Ця таблиця містить такі поля:

- availability_id – первинний ключ;
- user_id – зовнішній ключ на таблицю users;
- work_date – дата;
- shift_type – тип зміни;
- availability_status – статус доступності;
- comment – додатковий коментар.

Ця таблиця є ключовим джерелом вхідних даних для алгоритму планування. У ній уніфіковано зберігаються різні аспекти взаємодії працівника з графіком, зокрема індивідуальні побажання щодо змін. Об'єднання цих аспектів в одній таблиці дозволяє уникнути надлишкового поділу близьких за змістом сутностей та спрощує обробку даних на рівні алгоритму.

2.4.3 Таблиця schedule_assignments

Таблиця schedule_assignments містить результати автоматизованого формування графіка змін. Вона містить такі поля:

- assignment_id – первинний ключ;
- user_id – зовнішній ключ на таблицю users (може бути відсутнім для незакритих змін);
- work_date – дата;
- shift_type – тип зміни;
- assignment_status – статус призначення.

Таблиця відображає кінцевий результат роботи алгоритму планування. Кожен запис відповідає призначенню працівника на конкретну зміну або фіксує незаповнену зміну. Таким чином, сукупність записів цієї таблиці формує повний графік змін для заданого періоду.

2.4.4 Нормалізація бази даних

Спроектowana БД приведена до 3НФ, що забезпечує логічну узгодженість структури та відсутність надлишковості даних.

На першому етапі нормалізації всі таблиці приведено до 1НФ, що передбачає використання лише атомарних значень та відсутність повторюваних груп атрибутів. Кожне поле таблиці містить одне значення, що унеможливорює неоднозначність інтерпретації даних.

2НФ досягається за рахунок повної функціональної залежності неключових атрибутів від первинного ключа. Усі атрибути таблиць залежать від усього ключа, а не від його частини, що виключає часткові залежності.

На завершальному етапі структура БД приведена до 3НФ, відповідно до якої відсутні транзитивні залежності між неключовими атрибутами.

Кожен неключовий атрибут визначається виключно первинним ключем своєї таблиці та не залежить від інших неключових атрибутів.

У межах розробленої моделі це реалізовано таким чином: дані користувачів винесені в окрему таблицю `users`, що усуває їх дублювання в інших таблицях; інформація про доступність зберігається окремо від результатів призначень, що дозволяє чітко розмежувати вхідні та вихідні дані системи; результати роботи алгоритму представлені в таблиці `schedule_assignments` і не змішуються з даними, введеними користувачами.

Така організація БД забезпечує цілісність інформації, зменшує дублювання даних і спрощує їх оновлення та супровід (рисунок 2.4).

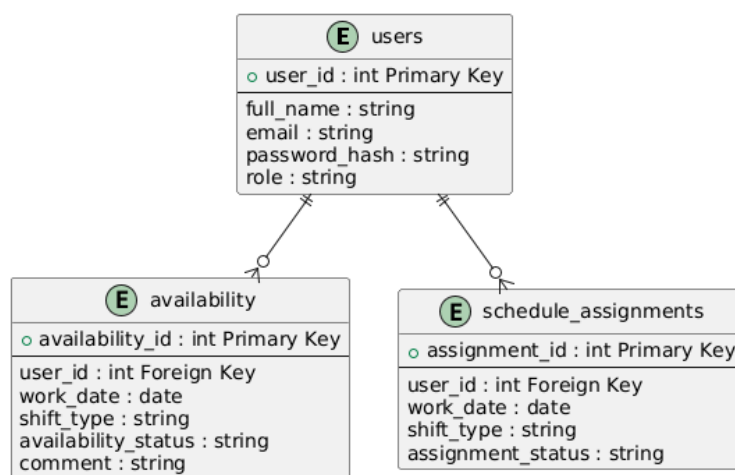


Рисунок 2.4 – ER-Діаграма сутностей і зв'язків бази даних

У результаті проєктування було сформовано компактну та узгоджену структуру БД, що включає лише необхідні сутності предметної галузі. Обрана модель дозволяє ефективно зберігати вхідні дані та результати планування, забезпечує цілісність інформації та відповідає вимогам нормалізації. Водночас відмова від надлишкових таблиць і винесення алгоритмічної логіки за межі БД дозволяє спростити структуру системи без втрати її функціональності.

Алгоритм Min-Conflicts належить до прикладної логіки системи. Його призначення полягає в обробці даних, що зберігаються в БД, та побудові графіка змін. Алгоритм виконує такі функції:

- аналізує доступність працівників;
- враховує жорсткі та м'які обмеження;
- здійснює пошук призначень із мінімальною кількістю конфліктів;
- визначає незакриті зміни;
- формує результат, що заноситься до таблиці `schedule_assignments`.

Таким чином, БД і алгоритмічний компонент виконують різні, але взаємопов'язані функції: БД забезпечує зберігання інформації, тоді як алгоритм забезпечує її інтелектуальну обробку та генерацію графіка.

3 ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ

Для реалізації системи автоматизованого планування графіків змін необхідно обрати програмні засоби та технології, які забезпечать підтримку багатошарової архітектури, реалізацію алгоритмічної логіки, обробку користувацьких запитів та інтеграцію NLP-компонентів. Оскільки система поєднує задачі веб-розробки, роботи з БД та використання алгоритмів штучного інтелекту, обраний стек технологій повинен забезпечувати продуктивність, масштабованість та зручність подальшого супроводу програмного забезпечення.

Під час вибору технологій враховувалися такі критерії:

- підтримка сучасної веб-архітектури;
- зручність реалізації REST API;
- підтримка asynchronous programming;
- можливість інтеграції ШІ та NLP-модулів;
- підтримка роботи з реляційними БД;
- наявність документації та активної спільноти розробників;
- можливість масштабування системи у майбутньому.

Для реалізації системи було обрано такі технології: Python, FastAPI, React, TypeScript, PostgreSQL, OpenAI API. Основним середовищем розробки було обрано Visual Studio Code, що забезпечує підтримку Python, TypeScript, React та сучасних веб-технологій у межах єдиного інтегрованого середовища розробки.

3.1 Python

Для реалізації backend-частини системи було обрано мову програмування Python. Python є високорівневою мовою програмування загального призначення, яка широко використовується у веб-розробці, автоматизації, аналізі даних, машинному навчанні та системах штучного

інтелекту [1, с. 28]. Завдяки поєднанню простоти синтаксису, гнучкості та великої кількості бібліотек Python став однією з найбільш популярних мов програмування для розробки сучасних інформаційних систем.

Однією з ключових особливостей Python є його орієнтованість на читабельність програмного коду [1, с. 27]. Синтаксис мови побудований таким чином, щоб програмний код був максимально зрозумілим для розробника та легко підтримувався у процесі подальшого розвитку програмного забезпечення [2, с. 16]. Це є особливо важливим для великих проєктів, у яких система поступово ускладнюється та включає значну кількість взаємопов'язаних компонентів.

Python підтримує декілька парадигм програмування, серед яких процедурне, об'єктно-орієнтоване та функціональне програмування [3, с. 54]. Подібна універсальність дозволяє використовувати мову для реалізації як окремих алгоритмічних модулів, так і складних багаторівневих веб-систем. У межах даного проєкту це є важливим, оскільки система автоматизованого планування графіків поєднує алгоритмічну логіку, механізми роботи з БД, REST API та NLP-компоненти.

Ще однією важливою перевагою Python є велика екосистема бібліотек та framework-рішень. Мова активно використовується у сфері штучного інтелекту та machine learning, що забезпечує наявність великої кількості інструментів для роботи з алгоритмами оптимізації, NLP-моделями та ШІ-сервісами [4, с. 29]. Для системи автоматизованого планування графіків це має особливе значення, оскільки backend повинен забезпечувати реалізацію constraint-based алгоритмів та інтеграцію з OpenAI API.

У межах розроблюваної системи Python використовується як основна мова реалізації backend-логіки. За допомогою Python реалізується алгоритм Min-Conflicts, який використовується для автоматизованого формування графіків змін із урахуванням жорстких та м'яких обмежень. Крім цього,

Python використовується для реалізації REST API, взаємодії з PostgreSQL та інтеграції NLP-модулів.

Особливу роль Python відіграє у реалізації алгоритмічної частини системи. Задача автоматизованого планування графіків належить до класу Constraint Satisfaction Problem, для якого необхідно ефективно працювати з великою кількістю взаємопов'язаних обмежень. Python дозволяє реалізувати подібні алгоритми у відносно компактному та зрозумілому вигляді, що суттєво спрощує подальшу модифікацію системи.

Крім цього, Python підтримує asynchronous programming, що є важливим для сучасних веб-застосунків [3, с. 47]. Асинхронна модель виконання дозволяє ефективно обробляти велику кількість одночасних HTTP-запитів та забезпечує стабільну роботу backend-системи навіть при високому навантаженні. Для систем автоматизованого планування це має особливе значення, оскільки backend одночасно взаємодіє з frontend, БД, NLP-модулем та зовнішніми API-сервісами.

Важливою перевагою Python є також його хороша інтеграція з сучасними веб-фреймворками. У межах даного проєкту використовується framework FastAPI, який забезпечує реалізацію REST API та asynchronous веб-архітектури [5, с. 32]. Python дозволяє ефективно поєднувати веб-компоненти, алгоритмічну логіку та ШІ-модулі у межах єдиної backend-системи.

Велика кількість документації, наукових публікацій та готових програмних рішень дозволяє значно спростити процес розробки та забезпечити більш ефективний супровід програмного забезпечення у майбутньому.

Таким чином, використання Python у межах даного проєкту дозволяє забезпечити гнучкість backend-архітектури, спростити реалізацію алгоритмів автоматизованого планування та забезпечити інтеграцію сучасних ШІ та NLP-технологій у систему автоматизованого формування графіків змін.

3.2 FastAPI

Для реалізації серверної частини системи було обрано фреймворк FastAPI. FastAPI є сучасним веб-фреймворком для мови програмування Python, який використовується для створення REST API та asynchronous веб-застосунків [5, с. 34]. Framework був розроблений з орієнтацією на високу продуктивність, підтримку сучасних стандартів веб-розробки та спрощення процесу створення backend-систем.

Однією з ключових особливостей FastAPI є використання стандарту ASGI (Asynchronous Server Gateway Interface), який забезпечує підтримку асинхронної обробки HTTP-запитів [5, с. 67]. На відміну від традиційних синхронних веб-фреймворків, FastAPI дозволяє ефективно працювати з великою кількістю одночасних підключень та значно покращує продуктивність системи при високому навантаженні.

Для системи автоматизованого планування графіків це має особливе значення, оскільки backend одночасно виконує декілька типів задач: обробку HTTP-запитів від frontend, взаємодію з PostgreSQL, роботу з алгоритмічними модулями планування та інтеграцію з OpenAI API. Асинхронна модель виконання дозволяє більш ефективно використовувати серверні ресурси та забезпечує стабільну роботу системи у режимі реального часу.

Ще однією важливою перевагою FastAPI є висока швидкість роботи. Він вважається одним із найбільш продуктивних Python фреймворків для створення REST API [6, с. 32]. Це досягається завдяки використанню asynchronous programming та оптимізованих механізмів обробки запитів. Висока продуктивність backend-системи є важливою для задач автоматизованого планування, оскільки система повинна швидко обробляти значну кількість користувацьких запитів, оновлень графіків та операцій аналізу обмежень.

FastAPI також підтримує автоматичну генерацію документації REST API відповідно до стандартів OpenAPI та Swagger [5, с. 78]. Подібний механізм дозволяє автоматично формувати інтерактивну документацію для всіх endpoint системи. Це значно спрощує тестування backend-компонентів та забезпечує більш зручну взаємодію між frontend і backend-частинами застосунку.

У межах даного проєкту FastAPI використовується як основний framework для реалізації серверної логіки системи. За допомогою FastAPI реалізуються REST API endpoint для роботи з графіками змін, обробки доступності працівників, взаємодії з NLP-модулем та передачі результатів роботи алгоритму планування.

Важливу роль FastAPI відіграє у реалізації механізмів авторизації та захисту API-запитів. У системі використовується JWT-автентифікація (JSON Web Token), яка дозволяє реалізувати stateless authentication та забезпечити контроль доступу до backend-функціональності. Після успішної авторизації користувач отримує access token, який надалі використовується для підтвердження прав доступу під час виконання HTTP-запитів. Подібний підхід дозволяє забезпечити безпечну взаємодію між frontend і backend та захистити персональні дані працівників і службову інформацію про графіки змін.

FastAPI забезпечує зручну підтримку middleware та dependency injection, що значно спрощує реалізацію JWT-автентифікації та механізмів перевірки доступу [5, с. 84]. Це дозволяє централізовано реалізувати логіку авторизації та забезпечити більш структуровану організацію backend-коду.

Крім цього, FastAPI добре інтегрується з ORM-бібліотеками та PostgreSQL. Це дозволяє спростити реалізацію механізмів взаємодії з БД та забезпечити більш структуровану організацію backend-системи. Використання ORM також дозволяє підвищити рівень абстракції під час роботи з SQL-запитами та спростити підтримку програмного забезпечення у майбутньому.

Важливою перевагою FastAPI є підтримка сучасної багатoshарової архітектури. Framework дозволяє розділити backend-систему на окремі модулі, серед яких routers, services, repositories та schemas. Подібний підхід забезпечує модульність програмного забезпечення та спрощує масштабування системи.

Ще однією важливою особливістю FastAPI є підтримка WebSocket-з'єднань. Це дозволяє реалізувати оновлення графіків та систему повідомлень у режимі реального часу. Для систем автоматизованого планування це є важливим, оскільки працівники повинні оперативно отримувати інформацію про зміни графіка та нові призначення.

FastAPI також забезпечує хорошу інтеграцію з Python-бібліотеками для роботи зі штучним інтелектом та NLP-технологіями. Завдяки цьому backend може ефективно взаємодіяти з OpenAI API та реалізовувати механізми обробки текстових повідомлень користувачів.

З урахуванням особливостей предметної галузі для даної системи найбільш доцільною є багатoshарова монолітна архітектура відповідно до патерна Layers [7, с. 20–31], у межах якої серверну частину реалізовано засобами FastAPI, що забезпечує високу продуктивність, підтримку асинхронного програмування, інтеграцію компонентів штучного інтелекту та побудову сучасного REST API для системи автоматизованого планування графіків змін.

3.3 React

Для реалізації frontend-частини системи було обрано бібліотеку React. React є JavaScript-бібліотекою для створення користувацьких інтерфейсів, яка була розроблена компанією Meta та сьогодні широко використовується для побудови сучасних веб-застосунків [8, с. 31].

Основною сферою застосування React є створення динамічних Single Page Application, у яких значна частина взаємодії між користувачем і системою виконується без повного перезавантаження сторінки.

Однією з ключових особливостей React є компонентний підхід до побудови інтерфейсу [9, с. 54]. Відповідно до даної концепції, користувацький інтерфейс формується з окремих незалежних компонентів, кожен із яких відповідає за реалізацію певної частини функціональності системи. Подібний підхід дозволяє значно спростити підтримку frontend-коду, забезпечити модульність системи та підвищити рівень повторного використання програмних компонентів.

Для системи автоматизованого планування графіків компонентна архітектура має особливе значення, оскільки frontend містить велику кількість інтерактивних елементів, серед яких календар змін, dashboard адміністратора, система повідомлень, форми введення доступності та NLP-чат. Реалізація кожного з цих модулів у вигляді окремого React-компонента дозволяє ізолювати функціональність та спростити подальший розвиток системи.

Ще однією важливою особливістю React є підтримка Virtual DOM [8, с. 57]. На відміну від традиційного підходу до оновлення HTML-сторінок, React використовує віртуальне представлення DOM-структури та виконує оновлення лише тих елементів інтерфейсу, які були змінені. Подібний механізм дозволяє суттєво покращити продуктивність веб-застосунку та забезпечити більш плавну взаємодію користувача із системою.

Для даного проєкту це є важливим, оскільки система автоматизованого планування графіків повинна працювати з великою кількістю динамічних даних. Під час оновлення графіків змін, отримання повідомлень або взаємодії з NLP-модулем frontend повинен оперативно оновлювати окремі елементи інтерфейсу без затримок та повного перезавантаження сторінки.

React також підтримує побудову Single Page Application, що є одним із найбільш поширених підходів у сучасній веб-розробці [8, с. 56]. SPA-архітектура дозволяє завантажувати основну частину застосунку один раз, після чого оновлюються лише окремі компоненти інтерфейсу. Це дозволяє забезпечити більш високу швидкість роботи застосунку та покращити зручність користування системою.

У межах розроблюваної системи React використовується для реалізації dashboard адміністратора, календаря змін, системи повідомлень, сторінок авторизації та NLP-чату. Крім цього, React забезпечує інтеграцію frontend-компонентів із REST API backend-системи та підтримує динамічне оновлення даних у режимі реального часу.

Ще однією перевагою React є велика екосистема додаткових бібліотек та UI-компонентів. Це дозволяє значно пришвидшити процес розробки frontend-системи та використовувати готові рішення для реалізації таблиць, календарів, форм, повідомлень та інших елементів інтерфейсу.

Важливою особливістю React є також хороша інтеграція з REST API та asynchronous веб-архітектурою. Frontend може ефективно взаємодіяти з backend через HTTP-запити та отримувати дані у форматі JSON. Це дозволяє забезпечити стабільний обмін інформацією між компонентами системи. Крім цього, React добре підходить для реалізації адаптивних інтерфейсів. Для системи автоматизованого планування це має важливе значення, оскільки користувачі можуть працювати із застосунком як на ПК, так і на мобільних телефонах.

Таким чином, використання React у межах даного проєкту дозволяє реалізувати сучасний інтерактивний frontend-інтерфейс, забезпечити підтримку SPA-архітектури, покращити продуктивність веб-застосунку та спростити реалізацію складних користувацьких компонентів системи автоматизованого планування графіків змін.

3.4 TypeScript

Для підвищення надійності frontend-частини системи використовується TypeScript. TypeScript є мовою програмування, розробленою компанією Microsoft, яка являє собою надбудову над JavaScript та додає механізми статичної типізації [10, с. 23]. Основною метою TypeScript є спрощення розробки великих веб-застосунків та підвищення стабільності програмного забезпечення.

Однією з головних проблем JavaScript є відсутність статичної типізації. У традиційному JavaScript типи змінних визначаються лише під час виконання програми, що може призводити до runtime помилок та ускладнювати підтримку великих frontend-систем. TypeScript вирішує дану проблему шляхом введення механізмів типізації, які дозволяють виявляти значну частину помилок ще на етапі компіляції [11, с. 258].

Для системи автоматизованого планування графіків це має особливе значення, оскільки frontend працює з великою кількістю структурованих даних. Зокрема, система взаємодіє з графіками змін, обмеженнями, інформацією про доступність працівників, повідомленнями та результатами роботи REST API. Використання TypeScript дозволяє забезпечити більш надійну роботу frontend-компонентів та зменшити ймовірність помилок під час передачі й обробки даних.

Ще однією важливою перевагою TypeScript є покращення структури програмного коду. Завдяки використанню інтерфейсів, узагальнених типів (generics) та механізмів типізації розробник може більш чітко описувати структуру даних та взаємодію між компонентами системи [10, с. 21]. Це є важливим для великих frontend-проектів із багаторівневою архітектурою та значною кількістю взаємопов'язаних модулів.

У межах даного проєкту TypeScript використовується для типізації React-компонентів, роботи з API-моделями, опису структури даних та валідації інформації, що передається між frontend і backend. Завдяки цьому

забезпечується більш стабільна взаємодія між компонентами системи та зменшується кількість runtime-помилки.

Важливу роль TypeScript відіграє під час роботи з REST API. Frontend постійно взаємодіє з backend-системою та отримує дані у форматі JSON. Використання типізованих моделей дозволяє чітко визначити структуру API-відповідей та забезпечити коректну обробку інформації у frontend-компонентах.

Крім цього, TypeScript забезпечує покращену інтеграцію із сучасними IDE та редакторами програмного коду. Завдяки підтримці autocomplete, type checking та навігації між компонентами значно спрощується процес розробки frontend-системи [11, с. 258]. Це дозволяє підвищити продуктивність розробки та покращити підтримку програмного забезпечення у майбутньому.

Ще однією важливою перевагою TypeScript є його хороша інтеграція з React. Використання TypeScript разом із React дозволяє створювати більш структуровані та масштабовані frontend-застосунки. Це є особливо важливим для систем автоматизованого планування графіків, у яких frontend поступово ускладнюється та включає значну кількість інтерактивних компонентів.

TypeScript також сприяє покращенню командної розробки. Чіткий опис типів даних та структур компонентів дозволяє різним розробникам швидше розуміти архітектуру frontend-системи та зменшує ризик виникнення помилок під час модифікації програмного коду.

Важливою особливістю TypeScript є те, що він компілюється у звичайний JavaScript, що забезпечує повну сумісність із сучасними браузерами та веб-технологіями [10, с. 21]. Завдяки цьому TypeScript можна використовувати у будь-яких сучасних frontend-проектах без необхідності зміни базової архітектури веб-застосунку.

Таким чином, використання TypeScript у межах даного проєкту дозволяє підвищити надійність frontend-системи, забезпечити більш чітку

структуру програмного коду, зменшити кількість runtime-помилки та спростити підтримку й масштабування веб-застосунку автоматизованого планування графіків змін.

3.5 PostgreSQL

Для збереження та обробки даних у системі використовується PostgreSQL. PostgreSQL є об'єктно-реляційною системою управління БД із відкритим кодом, яка широко застосовується у сучасних інформаційних системах та веб-застосунках [12, с. 16]. Система підтримує стандарти SQL та забезпечує високий рівень стабільності, надійності й масштабованості, що робить її придатною для реалізації backend-архітектури систем автоматизованого планування.

Однією з головних причин вибору PostgreSQL є підтримка механізмів забезпечення цілісності даних та транзакційності [13, с. 23]. Для системи автоматизованого планування графіків це має особливе значення, оскільки backend одночасно працює з великою кількістю взаємопов'язаних даних, серед яких інформація про працівників, графіки змін, доступність користувачів, обмеження та результати роботи алгоритму планування.

У межах розроблюваної системи PostgreSQL використовується для централізованого збереження інформації про користувачів, графіки змін, доступність працівників, повідомлення системи, історію змін та результати роботи алгоритму Min-Conflicts. Використання реляційної моделі даних дозволяє забезпечити логічну структуру інформації та підтримувати зв'язки між окремими сутностями системи.

Особливо важливим для даного проєкту є використання механізмів foreign keys та constraints [14, с. 13]. Система автоматизованого планування працює з великою кількістю взаємозалежних сутностей, тому підтримка цілісності даних є критично важливою. Наприклад, кожен запис про зміну повинен бути пов'язаний із конкретним працівником, а всі обмеження

мають відповідати існуючим користувачам та типам змін. PostgreSQL дозволяє забезпечити автоматичний контроль подібних зв'язків на рівні БД.

Ще однією важливою перевагою PostgreSQL є підтримка транзакцій та принципів ACID (Atomicity, Consistency, Isolation, Durability) [15, с. 23]. Це дозволяє гарантувати коректність обробки даних навіть у випадку помилок або одночасного виконання великої кількості операцій. Для систем автоматизованого планування це має особливе значення, оскільки backend може одночасно виконувати генерацію графіків, оновлення доступності працівників та обробку користувацьких запитів.

Важливу роль PostgreSQL відіграє також у реалізації механізмів масштабування системи. СУБД підтримує індексацію, оптимізацію SQL-запитів та ефективну роботу з великими обсягами даних [14, с. 44]. Це дозволяє забезпечити стабільну роботу системи навіть при збільшенні кількості працівників, графіків та користувацьких операцій.

Для даного проєкту PostgreSQL є також зручним завдяки хорошій інтеграції з Python та сучасними ORM-бібліотеками. Backend-система реалізована з використанням FastAPI та ORM-механізмів, що дозволяє працювати з моделями даних на більш високому рівні абстракції та спрощує реалізацію SQL-операцій.

Крім цього, PostgreSQL підтримує механізми резервного копіювання, журналювання змін та відновлення даних [12, с. 18]. Для корпоративних систем автоматизованого планування це є важливим фактором, оскільки система працює з критично важливою інформацією про графіки та персонал організації.

Ще однією перевагою PostgreSQL є його відкритий код та активна спільнота розробників. Завдяки цьому система постійно оновлюється, підтримує сучасні стандарти SQL та має велику кількість документації й готових рішень для інтеграції із веб-застосунками [16, с. 32].

Використання PostgreSQL також дозволяє реалізувати більш складні механізми аналітики та роботи з даними. У майбутньому система може бути

розширена засобами аналізу навантаження, прогнозування змін та формування статистичних звітів без необхідності переходу на іншу систему управління БД.

Отже, використання PostgreSQL у межах даного проєкту дозволяє забезпечити надійне збереження даних, підтримку транзакційності, цілісність інформації та стабільну роботу backend-системи автоматизованого планування графіків змін.

3.6 OpenAI API

Для реалізації NLP-функціональності у системі використовується OpenAI API. Інтеграція великих мовних моделей дозволяє забезпечити підтримку взаємодії природною мовою та автоматизувати процес інтерпретації текстових повідомлень користувачів [17, с. 4]. Використання NLP-технологій є одним із ключових елементів розроблюваної системи, оскільки дозволяє спростити взаємодію працівників із системою автоматизованого планування графіків змін.

Natural Language Processing є напрямом штучного інтелекту, який досліджує методи автоматичної обробки та аналізу природної мови [18, с. 113]. Сучасні NLP-системи використовують великі мовні моделі (Large Language Models), здатні аналізувати текст, визначати наміри користувачів та генерувати структуровані відповіді. У межах даного проєкту подібний підхід використовується для перетворення текстових побажань працівників у обмеження, які надалі застосовуються алгоритмом планування.

Однією з головних проблем традиційних систем автоматизованого планування є складність введення обмежень та побажань користувачів. У багатьох системах працівники змушені використовувати спеціальні форми або вручну налаштовувати параметри доступності.

Використання OpenAI API дозволяє реалізувати більш природний спосіб взаємодії, у межах якого користувач може формулювати побажання звичайною мовою.

Наприклад, працівник може вказати повідомлення типу: «не можу працювати після нічної зміни» або «бажано не ставити standby у вихідні». NLP-модуль аналізує текст повідомлення, визначає його зміст та автоматично формує структуроване обмеження для алгоритму Min-Conflicts.

У межах розроблюваної системи OpenAI API використовується для аналізу текстових повідомлень, визначення намірів користувача, інтерпретації часових параметрів та формування constraint-моделей. Крім цього, ШІ-модулі застосовуються для генерації пояснень щодо рішень алгоритму планування. Це дозволяє підвищити прозорість роботи системи та покращити взаємодію між користувачем і алгоритмом.

Особливу роль OpenAI API відіграє у реалізації персоналізації графіків змін. Система може враховувати індивідуальні побажання працівників та адаптувати графік відповідно до особливостей конкретного користувача. Подібний підхід дозволяє підвищити рівень задоволеності працівників та зменшити кількість конфліктних ситуацій.

Ще однією важливою перевагою використання великих мовних моделей є можливість реалізації ШІ-асистента для взаємодії з користувачами. Подібний модуль може допомагати працівникам у процесі роботи із системою, пояснювати причини певних рішень та відповідати на запитання щодо графіка змін.

Інтеграція OpenAI API також дозволяє спростити подальший розвиток NLP-функціональності системи. Завдяки використанню готових ШІ-моделей немає необхідності самостійно реалізовувати складні алгоритми обробки природної мови або створювати власні мовні моделі.

Крім цього, використання OpenAI API добре поєднується із backend-архітектурою на основі Python та FastAPI. Backend-система може

асинхронно надсилати запити до ШІ-моделі та отримувати результати аналізу тексту у режимі реального часу. Це дозволяє забезпечити швидку обробку повідомлень користувачів та підтримувати інтерактивну взаємодію із системою.

Використання NLP-компонентів також створює передумови для подальшого розвитку системи. У майбутньому ШІ-модулі можуть використовуватися для прогнозування конфліктів у графіках, аналізу навантаження працівників та автоматичної генерації рекомендацій щодо оптимізації розкладу.

Інтеграція OpenAI API та NLP-технологій дозволяє забезпечити більш природну взаємодію користувачів із системою автоматизованого планування графіків змін, автоматизувати процес обробки побажань працівників та підвищити рівень інтелектуалізації програмного забезпечення.

4 ОПИС РЕАЛІЗОВАНОЇ ПРОГРАМИ

4.1 Constraint Satisfaction Problem

Задача автоматизованого планування графіків робочих змін належить до класу задач Constraint Satisfaction Problem (CSP). Constraint Satisfaction Problem є однією з фундаментальних моделей штучного інтелекту та комбінаторної оптимізації, яка використовується для розв'язання задач, пов'язаних із пошуком допустимого набору значень за наявності певних обмежень [19, с. 164].

У загальному вигляді CSP-задача складається з трьох основних компонентів: множини змінних, доменів можливих значень та системи обмежень [19, с. 164]. Кожна змінна може набувати значення лише з визначеного домену, а всі призначення повинні задовольняти встановлені обмеження.

У контексті системи автоматизованого планування графіків змін змінними є окремі зміни або часові слоти, доменами – можливі працівники, які можуть бути призначені на відповідні зміни, а constraints визначають правила та обмеження формування графіка.

Основною особливістю CSP-задач є наявність великої кількості взаємозалежних обмежень [20, с. 46]. У задачах планування графіків обмеження можуть стосуватися доступності працівників, максимальної кількості змін, часових інтервалів між чергуваннями, нічних змін, standby-чергувань та індивідуальних побажань користувачів.

Для систем автоматизованого планування графіків особливе значення має поділ обмежень на жорсткі та м'які. Жорсткі обмеження (hard constraints) є обов'язковими для виконання та не можуть бути порушені за жодних умов. Наприклад, працівник не може бути призначений на зміну у період власної недоступності або працювати одночасно у двох змінах.

М'які обмеження (soft constraints) є бажаними, однак у деяких випадках можуть частково порушуватися. До подібних обмежень можуть належати побажання працівників щодо певних типів змін, уникнення нічних чергувань або рівномірний розподіл навантаження між персоналом.

Задачі Constraint Satisfaction Problem є обчислювально складними та часто належать до NP-hard задач. При збільшенні кількості працівників, змін та обмежень кількість можливих комбінацій зростає експоненціально. Через це використання brute force-підходів або повного перебору варіантів є практично неможливим для реальних систем планування.

У сучасних ІІІ-системах для розв'язання CSP-задач використовуються різні підходи, серед яких backtracking, локальний пошук, constraint propagation та евристичні алгоритми [19, с. 166]. Для задач автоматизованого планування графіків особливо ефективними є алгоритми локального пошуку, які дозволяють поступово покращувати поточний розв'язок та зменшувати кількість конфліктів.

Однією з головних переваг CSP-моделі є її універсальність. Constraint-based підхід дозволяє формалізувати складні правила планування та адаптувати систему до різних предметних галузей. У межах даного проєкту CSP-модель використовується для опису взаємозв'язків між працівниками, змінами та обмеженнями системи.

Формалізація задачі у вигляді Constraint Satisfaction Problem також дозволяє інтегрувати алгоритми штучного інтелекту та NLP-компоненти у процес автоматизованого планування. Побажання користувачів, сформульовані природною мовою, можуть автоматично перетворюватися у обмеження та враховуватися алгоритмом генерації графіка.

Ще однією важливою перевагою CSP-підходу є можливість динамічної перебудови графіка. У випадку зміни доступності працівників або виникнення нових обмежень система може повторно виконати пошук допустимого розв'язку без необхідності повного ручного перепланування графіка.

Для розв'язання задачі Constraint Satisfaction Problem у межах даної роботи використовується алгоритм Min-Conflicts, який належить до методів локального пошуку та забезпечує ефективне формування графіків змін при значній кількості обмежень.

Отже, використання моделі Constraint Satisfaction Problem дозволяє формалізувати задачу автоматизованого планування графіків змін, забезпечити підтримку великої кількості обмежень та створити основу для використання алгоритмів штучного інтелекту у процесі генерації графіка.

4.2 Алгоритм Min-Conflicts

Центральним компонентом системи автоматизованого планування графіків змін є алгоритм Min-Conflicts, який використовується для побудови розкладу з урахуванням великої кількості обмежень. У межах розробленої системи алгоритм реалізується як окремий backend-модуль, інтегрований із серверною частиною застосунку та БД.

Основною задачею алгоритму є автоматизоване призначення працівників на зміни таким чином, щоб мінімізувати кількість конфліктів між заданими обмеженнями. У системі використовується модель Constraint Satisfaction Problem, у якій кожна зміна розглядається як окрема змінна, а множина доступних працівників виступає доменом можливих значень.

На відміну від класичних алгоритмів повного перебору, Min-Conflicts реалізує локальний пошук рішення. Це дозволяє уникнути експоненційного зростання складності при збільшенні кількості працівників та змін. Алгоритм не намагається одразу знайти ідеальний графік, а поступово зменшує кількість конфліктів шляхом локального перепризначення змін.

У межах реалізації системи алгоритм працює з двома типами обмежень: hard constraints та soft constraints.

Hard constraints є обов'язковими для виконання та не можуть бути порушені. До них належать зайнятість працівника, перетин змін,

мінімальний інтервал між змінами, максимальна кількість змін, а також обмеження нічних чергувань.

Soft constraints використовуються для врахування індивідуальних побажань працівників та оптимізації навантаження. До них належать бажані типи змін, небажані дні, побажання щодо standby-змін, рівномірність навантаження і уникнення декількох нічних змін поспіль.

На початковому етапі система формує стартовий графік змін. Для цього використовується початкове призначення працівників на основі їх доступності. Отриманий графік може містити конфлікти, однак він виступає базою для подальшої оптимізації.

Після формування початкового розкладу алгоритм виконує аналіз конфліктних змін. Для кожного призначення система визначає кількість порушених обмежень та формує оцінку конфліктності. Далі обирається одна із конфліктних змін, після чого виконується пошук нового призначення, яке мінімізує кількість порушень.

У межах реалізації використовується функція scoring, яка оцінює допустимість кожного можливого призначення. Приклад реалізації функції оцінки конфліктів наведено у додатку (додаток Б – рисунок Б.1).

Даний фрагмент коду демонструє механізм оцінки кількості конфліктів для конкретного призначення працівника на зміну. Порушення жорстких обмежень отримують значно більшу вагу порівняно з soft constraints. Це дозволяє системі в першу чергу забезпечувати коректність графіка, а вже потім оптимізувати його відповідно до індивідуальних побажань користувачів.

Після оцінки всіх можливих кандидатів алгоритм обирає працівника з мінімальною кількістю конфліктів та оновлює графік (додаток Б – рисунок Б.2). Подібний процес повторюється ітеративно до моменту, поки кількість конфліктів не буде мінімізована або не буде досягнуто встановленого ліміту ітерацій.

У межах backend-архітектури алгоритм реалізується як окремий service-модуль FastAPI. Frontend надсилає запит на генерацію графіка через REST API, після чого backend отримує дані з PostgreSQL, формує CSP-модель та запускає алгоритм Min-Conflicts.

Після завершення роботи алгоритму сформований графік зберігається у таблиці `schedule_assignments` та передається frontend-частині системи у форматі JSON. Це дозволяє динамічно відображати результати планування у календарі змін та dashboard-компонентах.

Важливою особливістю реалізації є інтеграція NLP-модуля з алгоритмом планування. Побажання користувачів, введені природною мовою через чат-інтерфейс, автоматично інтерпретуються за допомогою OpenAI API та перетворюються у обмеження. Наприклад, повідомлення користувача «не можу працювати після нічної зміни» автоматично формалізується у `soft constraint` та враховується під час генерації графіка.

Ще однією важливою особливістю системи є підтримка повторної генерації графіка. У випадку зміни доступності працівників або виникнення нових обмежень адміністратор може повторно запустити алгоритм Min-Conflicts, після чого система автоматично перебудує розклад із урахуванням нових даних.

Таким чином, реалізація алгоритму Min-Conflicts дозволяє автоматизувати процес формування графіків змін, забезпечити врахування великої кількості обмежень та реалізувати інтелектуальну систему підтримки планування із використанням сучасних ШІ та NLP-технологій.

4.3 Формалізація обмежень

Одним із ключових етапів реалізації системи автоматизованого планування графіків змін є формалізація обмежень, які використовуються алгоритмом Min-Conflicts під час генерації розкладу.

Оскільки система базується на підході Constraint Satisfaction Problem, усі організаційні правила, обмеження та побажання працівників повинні бути представлені у формальному вигляді, придатному для автоматичної обробки алгоритмом. Крім того, формалізація обмежень забезпечує можливість автоматизованого виявлення конфліктів і пошуку рішень, що найбільшою мірою відповідають встановленим вимогам.

У межах даної системи constraints визначають допустимі умови призначення працівників на зміни та використовуються під час оцінки конфліктності графіка. Кожне призначення аналізується системою на відповідність набору hard constraints та soft constraints.

Hard constraints є критичними обмеженнями, які не можуть бути порушені за жодних умов. Якщо хоча б одне hard constraint порушується, сформований графік вважається неприпустимим. Подібні обмеження використовуються для забезпечення безпечного та коректного формування розкладу.

Soft constraints не є обов'язковими до виконання, однак система намагається максимально враховувати їх під час генерації графіка. Подібні обмеження використовуються для забезпечення справедливості, рівномірного навантаження та персоналізації розкладу.

Одним із найважливіших hard constraints у системі є обмеження на призначення змін після нічного чергування. Працівник, який працював у нічну зміну, не може бути призначений на ранкову, вечірню або повторну нічну зміну наступного дня, а також на standby.

Дане правило використовується для забезпечення достатнього часу відпочинку та запобігання перевтомі працівників. Крім цього, після нічної зміни працівник не може бути призначений на standby-чергування. Подібне правило пов'язане з максимально допустимим навантаженням працівника та необхідністю забезпечення періоду відпочинку після нічної роботи (рисунок 4.1).

```
def violates_post_night_rules(employee, shift):
    for s in employee.assignments:
        if s.shift_type == "night":
            next_day = s.day + 1
            if shift.day == next_day:
                restricted = {
                    "morning",
                    "evening",
                    "night",
                    "standby"
                }
                if shift.shift_type in restricted:
                    return True
    return False
```

Рисунок 4.1 – Приклад hard constraints у системі (обмеження на призначення нічних змін після нічного чергування)

У системі також використовується обмеження максимальної кількості робочих годин. Загальна тривалість зміни разом із standby не повинна перевищувати 11 годин (рисунок 4.2). У виняткових ситуаціях допускається навантаження до 12 годин, однак система намагається мінімізувати подібні випадки.

```
def exceeds_daily_workload(employee, shift):
    total_hours = shift.duration
    for s in employee.assignments:
        if s.day == shift.day:
            total_hours += s.duration
    return total_hours > 11
```

Рисунок 4.2 – Перевірка максимального добового навантаження

Ще одним важливим hard constraint є заборона standby-призначення у день основної зміни працівника (рисунок 4.3).

```
def violates_standby_rule(employee, shift):
    if shift.shift_type != "standby":
        return False
    for s in employee.assignments:
        if s.day == shift.day:
            if s.shift_type != "standby":
                return True
    return False
```

Рисунок 4.3 – Реалізація hard constraint для заборони standby-чергування у день основної зміни

У межах системи також використовується обмеження на послідовні нічні зміни. Працівник не може працювати дві нічні зміни поспіль, оскільки подібний графік створює надмірне фізичне та психологічне навантаження (рисунок 4.4).

```
def violates_consecutive_nights(employee, shift):
    if shift.shift_type != "night":
        return False
    for s in employee.assignments:
        if s.shift_type == "night":
            if abs(s.day - shift.day) == 1:
                return True
    return False
```

Рисунок 4.4 – Hard constraint для заборони послідовних нічних змін

Крім hard constraints, система використовує набір soft constraints, спрямованих на забезпечення справедливого розподілу навантаження між працівниками.

Однією з проблем ручного планування графіків є ситуація, коли один працівник отримує декілька однотипних змін поспіль, тоді як інші члени команди працюють у більш комфортному режимі.

Подібні ситуації можуть створювати враження нерівного ставлення та несправедливого розподілу навантаження. Для мінімізації подібних випадків система аналізує послідовність однотипних змін та намагається уникати призначення однакових shift-type декілька днів поспіль (рисунок 4.5).

```
def repeated_shift_pattern(employee, shift):
    consecutive = 0
    for s in employee.assignments:
        if s.shift_type == shift.shift_type:
            consecutive += 1
    return consecutive >= 3
```

Рисунок 4.5 – Фрагмент коду перевірки повторюваних shift-pattern

Особливу увагу система приділяє розподілу нічних змін. Відповідно до внутрішніх правил організації, працівники повинні працювати у нічні зміни приблизно 1–2 рази на тиждень.

На практиці частина працівників уникає нічних змін, через що інші члени команди змушені працювати у нічний час значно частіше. Для забезпечення fairness система аналізує кількість нічних змін кожного працівника та використовує механізм контролю рівномірності нічного навантаження (рисунок 4.6).

```
def generate_night_shift_recommendations(employees):
    recommendations = []
    for employee in employees:
        if insufficient_night_shifts(employee):
            recommendations.append(
                f"{employee.name} should consider "
                f"adding more night shifts."
            )
    return recommendations
```

Рисунок 4.6 – Soft constraint для контролю рівномірності нічних змін

Якщо працівник протягом тривалого періоду не обирає нічні зміни, система може автоматично сформувати рекомендацію щодо необхідності додавання нічних чергувань у власний графік доступності.

У межах реалізації системи підтримується параметр звільнення працівника від нічних змін – `exempt_from_nights` (рисунок 4.7). Подібне обмеження використовується під час генерації графіка та враховується алгоритмом Min-Conflicts. У подальшому розвитку системи передбачається реалізація окремого механізму адміністративного контролю для керування подібними обмеженнями.

```
def violates_night_exemption(employee, shift):
    if shift.shift_type == "night":
        return employee.exempt_from_nights
    return False
```

Рисунок 4.7 – Функція перевірки обмеження на нічні зміни

У межах реалізації алгоритму кожне обмеження має власну вагу у `scoring`-системі. Порушення `hard constraints` отримують критичний рівень штрафу, тоді як `soft constraints` використовуються для оцінки якості та `fairness` сформованого графіка (рисунок 4.8).

```
conflict_score = (
    hard_conflicts * 100 +
    soft_conflicts * 10
)

return conflict_score
```

Рисунок 4.8 – Реалізація `scoring`-функції оцінки конфліктності графіка

Подібний підхід дозволяє алгоритму Min-Conflicts у першу чергу усувати критичні конфлікти, а вже після цього оптимізувати графік відповідно до принципів справедливого розподілу навантаження.

Особливу роль у системі відіграє NLP-модуль, який дозволяє користувачам вводити побажання природною мовою. Після аналізу повідомлення за допомогою OpenAI API система автоматично перетворює текст у обмеження.

Усі обмеження зберігаються у структурованому вигляді у PostgreSQL та передаються backend-модулю планування під час запуску алгоритму Min-Conflicts. У випадку зміни доступності працівника або появи нових обмежень система може автоматично оновити constraint-модель та повторно перебудувати графік.

Таким чином, формалізація обмежень дозволяє перетворити організаційні правила, вимоги безпеки та побажання працівників у формальну модель, придатну для автоматизованої обробки алгоритмом Min-Conflicts, що забезпечує коректне, збалансоване та справедливе формування графіків змін.

4.4 Реалізація backend

Backend-частина системи автоматизованого планування графіків змін реалізована з використанням framework FastAPI та мови програмування Python. Backend відповідає за обробку HTTP-запитів, взаємодію з БД, реалізацію алгоритму Min-Conflicts, інтеграцію NLP-компонентів та забезпечення авторизації користувачів.

Під час проєктування backend-системи використовувався принцип багатошарової архітектури. Подібний підхід дозволяє розділити програмну логіку на окремі незалежні компоненти та забезпечити кращу масштабованість і підтримку програмного забезпечення.

У межах реалізації backend-система складається з декількох основних рівнів: routers; services; repositories; models; schemas; authentication modules; AI modules.

Router-рівень відповідає за обробку HTTP-запитів та взаємодію frontend із backend-системою. Кожен router реалізує окрему групу endpoint, пов'язаних із певним функціональним модулем системи.

У межах системи реалізовано такі основні routers:

- authentication router;
- schedule router;
- availability router;
- NLP router;
- notification router.

Service-рівень містить основну бізнес-логіку системи. Саме на цьому рівні реалізуються алгоритм Min-Conflicts, перевірка обмежень, scoring-механізми та NLP-обробка повідомлень користувачів.

Repository-рівень використовується для взаємодії з PostgreSQL та ізолює SQL-операції від бізнес-логіки системи. Подібний підхід дозволяє спростити підтримку backend коду та забезпечити більш гнучку структуру програмного забезпечення.

Model-рівень використовується для опису структури таблиць БД. Для цього застосовуються ORM-механізми, які дозволяють працювати з PostgreSQL через моделі Python без необхідності написання великої кількості SQL-запитів вручну.

Schema-рівень використовується для валідації HTTP-запитів та API-відповідей. FastAPI активно використовує Pydantic-схеми, що дозволяє автоматично перевіряти коректність переданих даних.

Основною задачею backend-системи є реалізація REST API для взаємодії frontend і алгоритмічного модуля. Frontend надсилає HTTP-запити до backend, після чого система виконує необхідну обробку даних та повертає результат у форматі JSON.

У межах системи реалізовано такі основні endpoint: POST /login; POST /generate_schedule; GET /schedule; POST /availability; POST /chat; GET /notifications.

Endpoint /login використовується для авторизації користувачів та генерації JWT-token. Endpoint /generate_schedule відповідає за запуск алгоритму Min-Conflicts та автоматичне формування графіка змін. Endpoint /availability використовується для збереження доступності працівників, тоді як /chat забезпечує взаємодію з NLP модулем системи.

Для забезпечення безпеки backend-системи використовується JWT-автентифікація. Після успішного входу користувач отримує access token, який використовується для підтвердження прав доступу під час виконання API-запитів. Використання JWT дозволяє реалізувати stateless authentication та забезпечити захист backend-функціональності без необхідності зберігання session-даних на сервері.

Особливу роль у системі відіграє role-based access control. Частина функціональності доступна лише адміністраторам системи. Наприклад, лише адміністратор може дозволити exemption від нічних змін, примусово перебудовувати графік, змінити параметри справедливості та переглядати статистику навантаження.

Подібний підхід дозволяє уникнути зловживань з боку користувачів та забезпечити більш коректну роботу системи.

Backend-система також реалізує окремий scheduler-service, який відповідає за запуск алгоритму Min-Conflicts. Після отримання запиту на генерацію графіка backend:

- отримує дані працівників із PostgreSQL;
- формує модель обмежень;
- запускає алгоритм Min-Conflicts;
- оцінює conflict score;
- зберігає результат у БД.

Загальна схема роботи backend під час генерації графіка виглядає таким чином: frontend надсилає запит на генерацію графіка, backend отримує дані. Після цього формуються обмеження, запускається

Min-Conflicts, генерується розклад, а результат зберігається у PostgreSQL. Далі frontend отримує оновлений графік.

Для реалізації NLP-функціональності backend інтегрується з OpenAI API. Користувацькі повідомлення передаються ШІ-модулю, після чого система інтерпретує текст та формує відповідні обмеження. Наприклад, повідомлення: «Не можу працювати у п'ятницю» автоматично перетворюється у: `unavailable(day=«Friday»)`. Подібний підхід дозволяє забезпечити більш природну взаємодію користувачів із системою планування.

Для роботи з PostgreSQL backend використовує ORM-моделі. У межах системи реалізовано таблиці `users`, `availability` та `schedules`, які використовуються для збереження інформації про користувачів, доступність працівників та сформовані графіки змін. Між таблицями використовуються зв'язки, що дозволяє підтримувати цілісність даних та забезпечити коректну роботу алгоритму планування. Між таблицями використовуються зв'язки, що дозволяє підтримувати цілісність даних та забезпечити коректну роботу алгоритму планування.

Ще однією важливою особливістю backend-системи є підтримка WebSocket-з'єднань. Це дозволяє реалізувати оновлення графіка у режимі реального часу та оперативне надсилання повідомлень користувачам.

Backend-система також підтримує механізми логування та обробки помилок. Усі критичні операції, пов'язані з генерацією графіка, авторизацією та зміною обмежень, фіксуються у системі логування. Це дозволяє спростити діагностику помилок та забезпечити аудит змін графіка.

Таким чином, реалізація backend-системи забезпечує підтримку REST API, інтеграцію алгоритму Min-Conflicts, роботу з PostgreSQL, NLP-обробку повідомлень користувачів та механізми авторизації, що дозволяє створити масштабовану та функціональну систему автоматизованого планування графіків змін.

4.5 Реалізація NLP-модуля

Одним із ключових компонентів системи автоматизованого планування графіків змін є NLP-модуль, який забезпечує взаємодію користувачів із системою за допомогою природної мови. Основною задачею модуля є автоматична інтерпретація текстових повідомлень користувачів та перетворення їх у структуру обмежень, придатну для подальшої обробки алгоритмом Min-Conflicts.

Реалізація NLP-модуля базується на використанні OpenAI API та великих мовних моделей, здатних аналізувати контекст повідомлень, визначати наміри користувача та формувати структуровані відповіді. Архітектурно NLP-модуль інтегрований у backend-систему та взаємодіє з frontend через WebSocket-з'єднання та REST API. Подібний підхід дозволяє забезпечити потокову передачу відповідей та реалізувати взаємодію із системою у режимі реального часу.

Загальна схема роботи NLP-модуля виглядає таким чином:

- користувач вводить повідомлення;
- frontend надсилає повідомлення через WebSocket;
- backend формує промпт;
- OpenAI API аналізує повідомлення;
- NLP-модуль визначає user intent;
- формуються обмеження;
- результат передається алгоритму планування.

Frontend надсилає повідомлення користувача через WebSocket. Backend отримує повідомлення та формує контекст, який містить:

- системний промпт та роль ШІ-асистента;
- попередні повідомлення;
- інформацію про тип користувача.

У системному prompt визначається роль ШІ-модуля як scheduling assistant, який допомагає формувати графіки змін та аналізувати обмеження.

Після формування контексту backend надсилає запит до OpenAI API. Для цього використовується asynchronous HTTP-запит, що дозволяє уникнути блокування backend-потoku під час генерації відповіді. Приклад виклику OpenAI API наведено нижче (рисунок 4.9).

```
if msg_type == 'chat':
    chat_log.append({'role': 'user', 'content': user_input})

    try:
        response = openai.chat.completions.create(
            model='gpt-4o',
            messages=chat_log,
            temperature=0.7,
            stream=True
        )

        ai_response = ''
        for chunk in response:
            if chunk.choices[0].delta.content is not None:
                content = chunk.choices[0].delta.content
                ai_response += content
                await websocket.send_text(content)

    chat_log.append({'role': 'assistant', 'content': ai_response})
```

Рисунок 4.9 – Виклик OpenAI API

Після отримання відповіді користувача система виконує обробку даних про доступність. Backend аналізує визначені NLP-модулем параметри, зокрема тип зміни, день та статус доступності працівника. Після цього отримані дані автоматично зберігаються у PostgreSQL для подальшого використання алгоритмом планування графіків (рисунок 4.10).

Однією з ключових особливостей реалізації NLP-модуля є підтримка streaming відповідей. Мовна модель повертає відповідь не повністю, а у вигляді окремих текстових чанків. Backend отримує кожен фрагмент відповіді та негайно передає його frontend через WebSocket.

Подібний механізм створює ефект «живої» відповіді та значно покращує користувацький досвід взаємодії із системою.

На backend-стороні кожен chunk перевіряється на наявність тексту та додається до змінної ai_response, яка зберігає повну відповідь моделі.

```

if user_id:
    extracted_data = extract_availability_data(user_input)
    store_chat_interaction(user_id, user_input, ai_response, extracted_data)

    save_keywords = ['save', 'ok', 'yes', 'confirm', 'go ahead']
    user_input_lower = user_input.lower()
    should_save = any(keyword in user_input_lower for keyword in save_keywords)

    if extracted_data.get('availability'):
        work_date = extracted_data.get('work_date')
        shift_type = extracted_data.get('shift_type')
        availability_status = extracted_data.get('availability_status', 'available')

        if work_date and shift_type:
            import re
            if re.match(r'^\d{4}-\d{2}-\d{2}$', work_date) or should_save:
                store_availability(user_id, shift_type, work_date, availability_status, user_input)

```

Рисунок 4.10 – Реалізація механізму обробки NLP-відповіді та збереження даних у PostgreSQL

Frontend отримує чанки у режимі реального часу та динамічно оновлює DOM-структуру сторінки без необхідності перезавантаження інтерфейсу.

Однією з найбільш складних задач під час реалізації NLP-модуля була коректна обробка streaming відповідей. Через особливості chunk-based передачі тексту виникали проблеми зі зникненням пробілів або «склеюванням» слів. Для вирішення даної проблеми було реалізовано механізм коректного об'єднання текстових чанків перед їх відображенням у frontend-інтерфейсі.

Крім генерації текстових відповідей NLP-модуль використовується для аналізу повідомлень користувачів та визначення їх намірів. Після отримання повідомлення система аналізує текст та намагається визначити небажані типи змін, побажання щодо нічних чергувань чи standby.

Для підвищення стабільності роботи системи реалізовано fallback-механізми. Якщо OpenAI API недоступний або повідомлення не вдалося інтерпретувати, система може використовувати predefined-responses або базові keyword-механізми.

У межах реалізації використовується словник keywords, який містить набір ключових слів та відповідних реакцій системи. Подібний підхід

дозволяє забезпечити базову функціональність навіть у випадку проблем із зовнішнім ШІ-сервісом.

Ще однією важливою особливістю NLP-модуля є підтримка context-aware interaction. Система враховує попередні повідомлення користувача та підтримує історію діалогу, що дозволяє формувати більш релевантні відповіді.

Backend також реалізує механізми обробки помилок OpenAI API та логування NLP-запитів. У випадку помилки система не завершує роботу аварійно, а повертає користувачу повідомлення про тимчасову недоступність ШІ-модуля.

Таким чином, реалізація NLP-модуля дозволяє забезпечити інтелектуальну взаємодію користувачів із системою автоматизованого планування графіків, автоматичну інтерпретацію побажань природною мовою та інтеграцію ШІ-компонентів у процес генерації розкладу.

4.6 Реалізація frontend та користувацького інтерфейсу

Розробка системи передбачає не лише алгоритмічне вирішення задачі планування, але й повний цикл проєктування користувацького інтерфейсу. У процесі роботи виконуються етапи визначення цілей і сценаріїв взаємодії, розробки структури інтерфейсу, створення форм введення та відображення даних, а також побудови макетів екранів із урахуванням принципів візуальної ієрархії, візуального потоку, групування та використання шаблонів компоновки.

4.6.1 Архітектура frontend-системи

Система автоматизованого планування графіків змін призначена для формування робочого розкладу з урахуванням доступності працівників,

обмежень та побажань користувачів. Основними учасниками системи є працівник та адміністратор.

Працівник взаємодіє із системою через веб-інтерфейс або ШІ-чат, вводячи інформацію про власну доступність, побажання щодо змін, відпустки або інші обмеження. На основі введених даних система формує набір обмежень, які враховуються під час генерації графіка.

Адміністратор відповідає за контроль процесу планування, налаштування обмежень та запуск алгоритму формування графіка. Крім цього, адміністратор може переглядати результати роботи системи, аналізувати незакриті зміни та ініціювати повторну генерацію розкладу.

Формування графіка здійснюється автоматично із використанням алгоритму Min-Conflicts, який враховує:

- доступність працівників;
- типи змін;
- обмеження справедливості;
- історію попередніх чергувань.

У системі підтримуються денні, нічні, standby-зміни та робота в офісні години. Під час генерації графіка враховуються обмеження на мінімальний час відпочинку між змінами, кількість нічних чергувань та рівномірність навантаження між працівниками.

Після генерації графіка користувачі отримують доступ до загального та персоналізованого розкладу. У випадку наявності незакритих змін система може повідомляти доступних працівників про можливість добровільного вибору додаткової зміни. Для підвищення прозорості процесу планування система також надає інформацію про причини прийняття окремих рішень під час формування розкладу.

Компонентно-функціональна структура взаємодії працівника із системою наведена у таблиці 4.1, а структура взаємодії адміністратора – у таблиці 4.2.

Таблиця 4.1 – Компонентно-функціональна структура для працівника

Мета	Задачі	Процедури	Дії користувача і системи
1			Отримання готового графіку роботи
	1.1		Введення даних
		1.1.1	Ввести доступність через інтерфейс або чат-бот
		1.1.2	Ввести дані про відпустку чи військову службу
		1.1.3	Зберегти введені дані
	1.2		Отримання графіка
		1.2.1	Отримати повідомлення про сформований графік
		1.2.2	Переглянути графік (загальний та персональний)
	1.3		Робота з незакритими змінами
		1.3.1	Отримати повідомлення про незакриті зміни
		1.3.2	Обрати незакриту зміну
		1.3.3	Підтвердити вибір зміни

Таблиця 4.2 – Компонентно-функціональна структура для адміністратора

Мета	Задачі	Процедури	Дії адміністратора і системи
1			Отримання графіку для колективу

Продовження таблиці 4.2

Мета	Задачі	Процедури	Дії адміністратора і системи
	1.1		Контроль вхідної інформації
		1.1.1	Перевірити наявність введених даних від працівників
		1.1.2	Встановити або змінити обмеження
	1.2		Формування розкладу
		1.2.1	Запустити алгоритм формування графіка
		1.2.2	Отримати результат роботи алгоритму
	1.3		Аналіз результату
		1.3.1	Переглянути таблицю побажань працівників
		1.3.2	Переглянути сформований графік
		1.3.3	Перевірити наявність незакритих змін
	1.4		Прийняття рішення
		1.4.1	Затвердити графік
		1.4.2	Ініціювати повторну генерацію

Функціонально-об'єктна структура для працівника та адміністратора представлена у таблицях 4.3 та 4.4 відповідно. Дані моделі описують взаємозв'язок між користувачами, інтерфейсом системи, засобами взаємодії та результатами виконання операцій. Вони відображають набір дій, доступних кожній ролі, а також об'єкти системи, з якими відбувається взаємодія під час виконання цих дій, що дозволяє деталізувати функціональні можливості системи.

Таблиця 4.3 – Функціонально-об’єктна структура для працівника

№	Ергатичний елемент	Засоби інтерфейсу	Знаряддя праці	Предмети праці	Продукти праці
1.1.1	Працівник	Чат-бот / веб-форма	ПК / смартфон	Дані доступності	Збережена доступність
1.2.1	Працівник	Веб-інтерфейс	ПК / смартфон	Графік	Переглянути графік
1.3.1	Працівник	Веб-інтерфейс	ПК / смартфон	Незакриті зміни	Закрита зміна

Таблиця 4.4 – Функціонально-об’єктна структура для адміністратора

№	Ергатичний елемент	Засоби інтерфейсу	Знаряддя	Предмет	Продукт
1.1.1	Адміністратор	Панель	ПК / смартфон	Обмеження	Налаштовані правила
1.1.2	Адміністратор	Панель керування	ПК / смартфон	Дані	Перевірені дані
1.2.1	Адміністратор	Інтерфейс	ПК / смартфон	Дані	Запуск алгоритму
1.2.2	Система	Алгоритм	Сервер	Дані	Графік
1.3.1	Адміністратор	Інтерфейс	ПК / смартфон	Графік	Аналіз

Система орієнтована на використання у командах із безперервним режимом роботи, зокрема в ІТ-компаніях, центрах моніторингу та службах технічної підтримки. Взаємодія із системою здійснюється через

веб-інтерфейс з підтримкою англійської та іврити. Основна обробка даних виконується на серверній стороні, тому клієнтська частина потребує лише сучасного веб-браузера та доступу до мережі Інтернет.

Сценарій роботи системи має циклічний характер та прив'язаний до двотижневого періоду планування. Після введення даних про доступність адміністратор запускає алгоритм формування графіка, система виконує генерацію розкладу, після чого користувачі отримують доступ до результатів планування. Функціонально-часові структури для працівника наведено на рисунку 4.11.

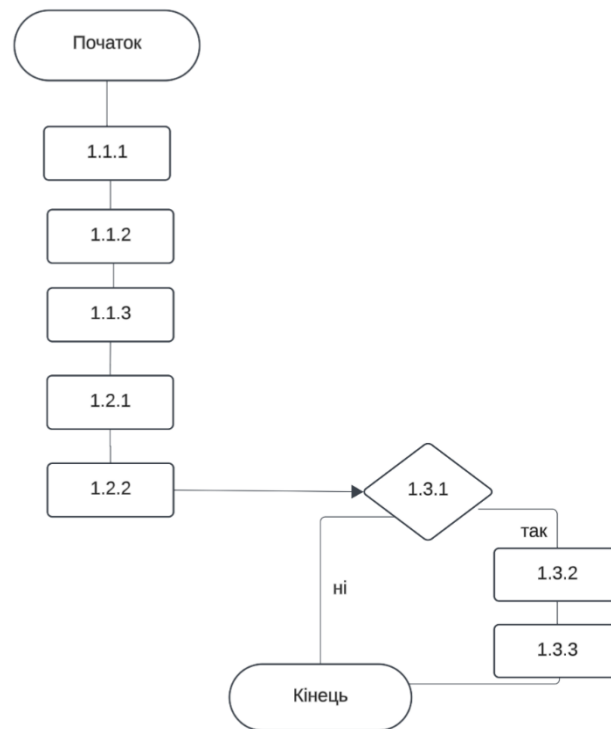


Рисунок 4.11 – Функціонально-часова структура для працівників

У межах кожного циклу планування система забезпечує централізований збір вхідних даних, автоматизовану перевірку обмежень та формування розкладу з урахуванням доступності працівників і заданих правил чергування. Після генерації графіка користувачі можуть переглядати призначені зміни, отримувати сповіщення про оновлення розкладу та

взаємодіяти із системою у випадку виникнення незакритих змін. Така організація процесу дозволяє зменшити кількість ручних операцій під час планування та підвищити оперативність оновлення графіків у разі зміни вхідних умов.

Функціонально-часова структура для адміністратора наведена на рисунку 4.12. На відміну від сценарію роботи працівника, дії адміністратора охоплюють етапи контролю та керування процесом формування графіка. Адміністратор здійснює перевірку повноти введених даних, налаштовує параметри планування та ініціює запуск алгоритму генерації розкладу. Після завершення обчислень виконується аналіз сформованого графіка, перевірка наявності конфліктів або незакритих змін, а також затвердження результатів планування або повторний запуск генерації у разі необхідності коригування параметрів.

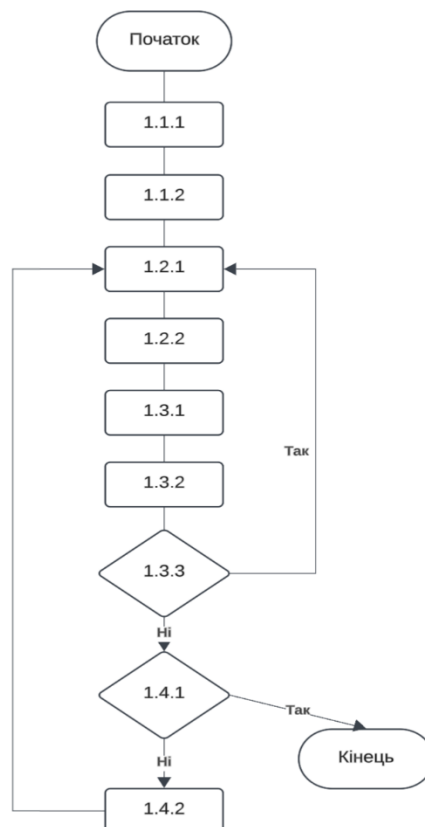


Рисунок 4.12 – Функціонально-часова структура для адміністратора

4.6.2 Реалізація інтерфейсу введення даних

Сторінка Availability є основною формою введення даних у системі автоматизованого планування графіків змін. Саме через даний інтерфейс користувач взаємодіє із системою, задає власну доступність, формує обмеження та передає інформацію алгоритму генерації графіка.

Інтерфейс сторінки реалізовано у вигляді інтерактивного календаря, що дозволяє користувачу швидко позначати доступні та недоступні часові проміжки, а також вказувати додаткові побажання щодо роботи у конкретні дні або зміни (рисунок 4.13).

The screenshot displays the 'Availability' page in the ShiftFlow application. At the top, there is a navigation bar with links: ShiftFlow, Availability (active), Schedule, and Analytics. The main content area is divided into three sections:

- Calendar:** A calendar for April 2026. The days of the week are listed at the top. The dates 1 through 9 are visible. The date 1 (Wednesday) is highlighted with a blue border. Below the calendar, there are four blue buttons labeled 5, 6, 7, and 8, which likely represent different shift types or time slots.
- Selected dates:** A section showing the selected dates: 'Apr 5, Apr 6, Apr 7, Apr 8'. Below this, there is a 'Shift type' section with three radio buttons: 'Morning' (unselected), 'Day' (selected), and 'Night' (selected). A 'Save' button is located below the shift type selection.
- Vacation / Military service:** A section for marking vacation or military service. It includes 'Start date' (09/04/2026) and 'End date' (12/04/2026) fields, each with a calendar icon. A 'Save' button is located below these fields.

On the right side of the page, there is an 'AI Assistant' chat window. It contains a blue message bubble that says 'I can work evenings next week except Friday'. Below this, there is a text input field with the text 'I've noted that you're available for evening shifts next week. Monday through Thursday. Would you like me to mark these dates on your calendar?' and a 'Send' button.

Рисунок 4.13 – Сторінка Availability із календарем, формою введення та ІІІ-помічником

Календар використовується для вибору дат та відображення доступності користувача. Використання календарного елемента дозволяє обмежити введення лише допустимими значеннями та зменшує ймовірність помилок під час роботи з датами. Користувач може швидко перемикатися між місяцями та переглядати загальну структуру власної доступності.

Після вибору конкретної дати користувач може задати тип зміни за допомогою прапорців – checkbox (рисунок 4.14). У системі підтримуються три основні типи змін: Morning; Day; Night.

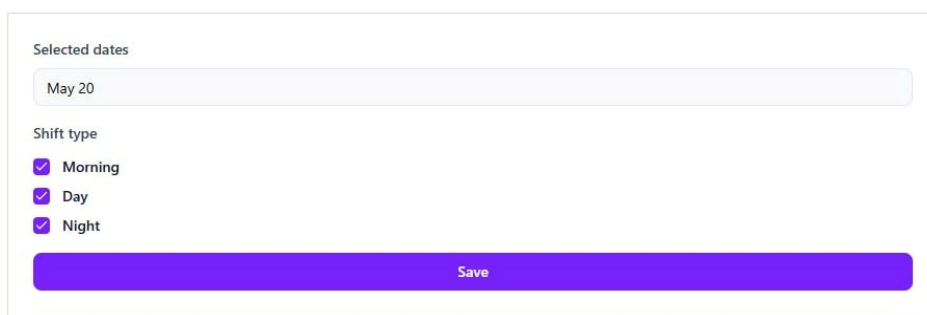


Рисунок 4.14 – Checkbox Morning; Day; Night

Для вибору shift-type використовується група checkbox-елементів, оскільки користувач може обрати декілька варіантів одночасно. Подібний підхід відповідає сценарію множинного вибору та забезпечує швидкий доступ до всіх можливих варіантів зміни.

Під календарем розташована форма редагування вибраної дати. Вона містить:

- поле відображення обраної дати;
- елементи вибору типу зміни;
- кнопку збереження даних про доступність.

Після натискання кнопки «Save» frontend виконує збереження інформації та передає відповідні дані backend-системі.

Для задання довготривалих обмежень у системі реалізовано окремий блок Vacation / Military service (рисунок 4.15). Даний компонент дозволяє

користувачеві задавати періоди відсутності за допомогою двох полів введення дати: Start date; End date.

Подібний підхід дозволяє задавати діапазони дат без необхідності редагування кожного дня окремо. Це особливо важливо для випадків тривалих відпусток, військової служби або інших довгострокових обмежень доступності.

The screenshot displays a user interface for selecting dates and shift types. At the top, there is a 'Selected dates' input field. Below it, the 'Shift type' section includes three radio button options: 'Morning', 'Day', and 'Night'. The 'Vacation / Military service' section contains 'Start date' and 'End date' input fields, both pre-filled with '09/04/2026'. A calendar pop-up is open, showing the month of April 2026. The calendar grid highlights the 9th and 16th. At the bottom of the interface is a prominent blue 'Save' button.

Рисунок 4.15 – Введення діапазону дат (початкова та кінцева дата)

Окрім структурованого введення даних, система підтримує альтернативний спосіб взаємодії через NLP-інтерфейс. Праворуч від календаря розташована панель ШІ-помічника, реалізована у вигляді чат-інтерфейсу.

Даний модуль дозволяє користувачеві вводити інформацію природною мовою без необхідності ручного налаштування constraint-параметрів (додаток А – рисунок А.2). Наприклад, користувач може вводити повідомлення: «I can do only morning and night shifts from Monday till Friday» (рисунок 4.16).

Після введення повідомлення frontend передає текст backend-системі, де NLP-модуль виконує інтерпретацію запиту та перетворення його

у структуровані обмеження. Отримані результати автоматично відображаються у календарі, при цьому відповідні дні підсвічуються, а користувач може переглянути інформацію про згенеровані зміни.

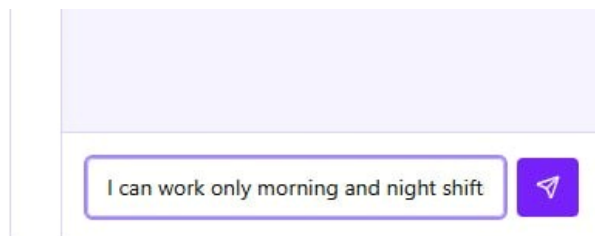


Рисунок 4.16 – Інтерфейс ІІІ-чату

Подібний механізм забезпечує взаємозв'язок між текстовим введенням та візуальним представленням даних. Користувач отримує можливість контролювати результати роботи NLP-модуля та перевіряти коректність інтерпретації введених побажань. У разі виявлення помилок або неточностей користувач може внести коригування безпосередньо через календарний інтерфейс. Такий підхід підвищує зручність введення даних, зменшує кількість ручних операцій та спрощує процес формування обмежень для подальшої генерації графіка.

Таким чином, система підтримує два основні способи введення даних:

- структуроване введення через календарний інтерфейс;
- введення природною мовою через ІІІ-чат.

Під час проєктування інтерфейсу особлива увага приділялася поєднанню традиційних елементів введення даних із можливостями NLP-взаємодії. Це дозволяє користувачеві самостійно обирати найбільш зручний спосіб роботи із системою.

Інтерфейс побудований за принципом поділу на функціональні зони:

- центральна область для роботи з календарем;
- нижня частина для редагування доступності та періодів відсутності;

- права панель для NLP-взаємодії через ШІ-помічника.

Подібна структура забезпечує зрозумілу організацію робочого простору та дозволяє користувачеві швидко виконувати основні дії без необхідності переходу між великою кількістю окремих сторінок.

Для реалізації frontend-логіки використовуються React компоненти та TypeScript. Календар, форми введення та NLP-панель реалізовані як окремі компоненти, що спрощує підтримку frontend-коду та забезпечує модульність системи.

Таким чином, сторінка Availability забезпечує гнучкий та інтуїтивно зрозумілий механізм введення даних, поєднуючи структуровану роботу з календарем та можливості інтелектуальної обробки природної мови у межах єдиного користувацького інтерфейсу.

4.6.3 Реалізація інтерфейсу перегляду графіка

Сторінка Schedule призначена для відображення сформованого графіка змін та є основним інструментом аналізу розкладу як для працівників, так і для адміністратора. Основна задача даного інтерфейсу полягає у наданні структурованого представлення графіка, що дозволяє швидко оцінити розподіл змін, навантаження працівників та наявність незайнятих змін.

Інтерфейс сторінки реалізовано у вигляді таблиці або календарного представлення, де користувач може переглядати зміни за днями, працівниками або часовими інтервалами.

Для покращення сприйняття інформації використовуються візуальні позначення, які дозволяють швидко ідентифікувати тип зміни, статус її заповнення та можливі конфлікти у розкладі.

Основним елементом сторінки є таблиця графіка змін, яка займає більшу частину інтерфейсу (додаток А – рисунок А.3). Таблична модель

дозволяє відобразити великий обсяг структурованої інформації та забезпечує зручне порівняння змін між працівниками й днями.

У межах таблиці поєднуються календарні дні, типи змін, призначення працівників, standby-зміни та інформація про незакриті зміни.

Така структура забезпечує високу щільність подання інформації та дозволяє користувачеві швидко аналізувати сформований графік.

У верхній частині таблиці розташована часова шкала із зазначенням дат та днів тижня. Це забезпечує контекстуальне сприйняття графіка та дозволяє аналізувати розподіл змін у межах конкретного періоду.

У комірках таблиці відображаються працівники, призначені на відповідну зміну у конкретний день. Подібний підхід дозволяє швидко визначати навантаження працівників та виявляти можливі проблеми у сформованому графіку.

Рядки таблиці відповідають окремим типам змін. У системі підтримуються Morning Shift, Evening Shift, Night Shift, Peak Hours Shift, Morning Standby, Evening Standby та Night Standby.

Для покращення візуального сприйняття інформації використовується кольорове кодування окремих станів. Наприклад, незаповнені або проблемні зміни можуть виділятися контрастним кольором, що дозволяє швидко ідентифікувати незакриті позиції без необхідності детального аналізу всієї таблиці.

Під час реалізації таблиці використовувалися принципи візуального розділення рядків і стовпців. Для підвищення читабельності застосовується *alternating row styling*, при якому сусідні рядки мають різне фонове оформлення.

Додатково реалізовано *hover* ефект, який дозволяє візуально підсвічувати активний рядок або комірку таблиці під час наведення курсора.

Подібний підхід полегшує навігацію у межах великого обсягу даних та дозволяє користувачеві швидше співвідносити інформацію по горизонталі та вертикалі таблиці.

Для звичайного користувача система додатково підтримує персоналізований перегляд графіка. Користувач може відфільтрувати таблицю та переглядати лише власні зміни. Це дозволяє зосередитися на персональному розкладі без необхідності аналізувати весь графік команди.

Для реалізації фільтрації використовується механізм вибору працівника зі списку. Після вибору відповідного користувача система автоматично відображає лише релевантні записи графіка.

Подібний підхід реалізує принцип selective data visualization, при якому користувач працює лише з необхідною підмножиною інформації.

Окрім перегляду графіка, інтерфейс підтримує елементи керування процесом генерації розкладу. Для адміністратора доступні кнопки «Generate New» та «Approve Schedule» (рисунок 4.17).



Рисунок 4.17 – Кнопки для керування процесом формування графіку

Кнопка «Generate New» використовується для повторного запуску алгоритму Min-Conflicts та перебудови графіка, тоді як «Approve Schedule» дозволяє підтвердити фінальну версію розкладу.

Розташування елементів керування під таблицею забезпечує логічну послідовність взаємодії: спочатку користувач аналізує сформований графік, після чого приймає рішення щодо його підтвердження або повторної генерації. У межах сторінки також можуть відображатися текстові повідомлення про стан системи, наприклад інформація про наявність незакритих змін або перевищення навантаження окремих працівників.

Під час проєктування сторінки особлива увага приділялася організації великого обсягу багатовимірних даних. Таблична модель дозволяє поєднати часову структуру, типи змін, працівників, а також результати роботи

алгоритму планування. Завдяки цьому користувач отримує цілісне уявлення про сформований графік та може ефективно аналізувати результати роботи системи автоматизованого планування.

У таблиці можливо вивести лише частину стовпців, залишивши інші прихованими залежно від вибору користувача (рисунок 4.18 та рисунок 4.19).



Рисунок 4.18 – Використання фільтрації за тижнем

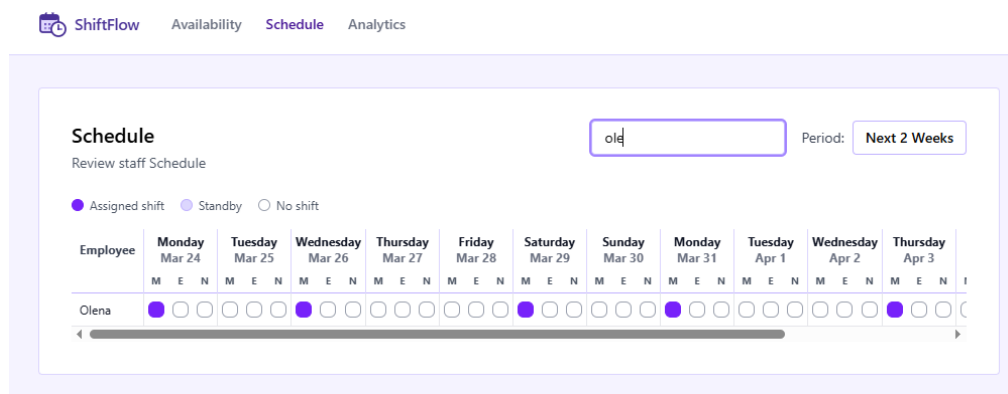


Рисунок 4.19 – Пошук і фільтрація в таблиці Schedule

Користувач самостійно визначає, які аспекти даних необхідно переглянути, а система відображає лише релевантні фрагменти. Для покращення сприйняття таблиці використано принципи візуального розділення рядків і стовпців. Дані організовані таким чином, щоб користувач міг легко орієнтуватися навіть у великому обсязі інформації. Також використано виділення окремих клітинок, що дозволяє побачити, які зміни бажані або призначені, що відноситься до техніки data brushing, коли вибірка даних виділяється та відображається в іншому контексті.

4.6.4 Реалізація інтерфейсу введення даних

Сторінка Analytics призначена для аналізу розподілу змін та оцінки навантаження працівників. Даний інтерфейс доповнює функціональність системи автоматизованого планування та забезпечує користувачів інструментами для перегляду статистики, аналізу fairness розподілу змін і контролю робочого навантаження.

У верхній частині сторінки відображаються основні статистичні показники системи, зокрема загальна кількість змін за визначений період, середня кількість змін на одного працівника і coverage rate графіка.

Для структурування інформації використовується система вкладок: Overview, Shifts та Performance. Подібний підхід дозволяє розділити різні типи аналітичної інформації та забезпечує більш зручну навігацію між статистичними даними.

Для покращення візуального сприйняття інформації у графічному інтерфейсі використовуються кольорове кодування типів змін, легенди, осі координат та tooltip-підказки.

У вкладці Overview реалізовано графічне представлення популярності різних типів змін протягом тижня (рисунок 4.20). Стовпчикові діаграми дозволяють аналізувати співвідношення morning, evening та night shifts для окремих днів і виявляти найбільш або найменш завантажені періоди.

Вкладка Shifts використовується для аналізу розподілу змін між працівниками (рисунок 4.21). Для цього застосовується stacked bar chart, що дозволяє одночасно оцінювати загальне навантаження працівника та співвідношення різних типів змін у його графіку.

При наведенні курсора на окремий елемент графіка система відображає детальну інформацію щодо кількості morning, evening та night shifts конкретного працівника. Це дозволяє швидко виявляти дисбаланс у розподілі навантаження та оцінювати рівномірність призначення різних типів змін між працівниками.

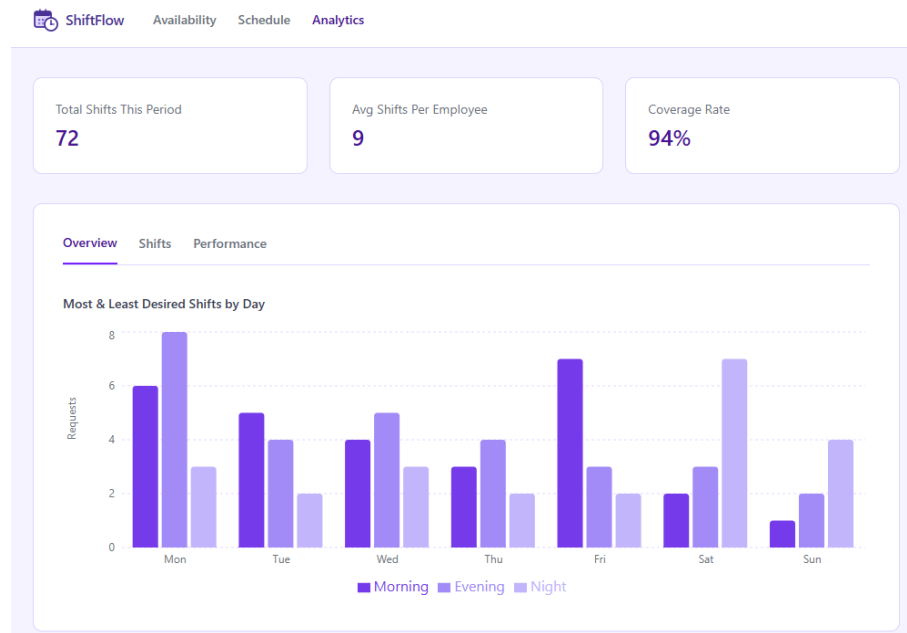


Рисунок 4.20 – Analytics з графічним представленням статистики (Overview)

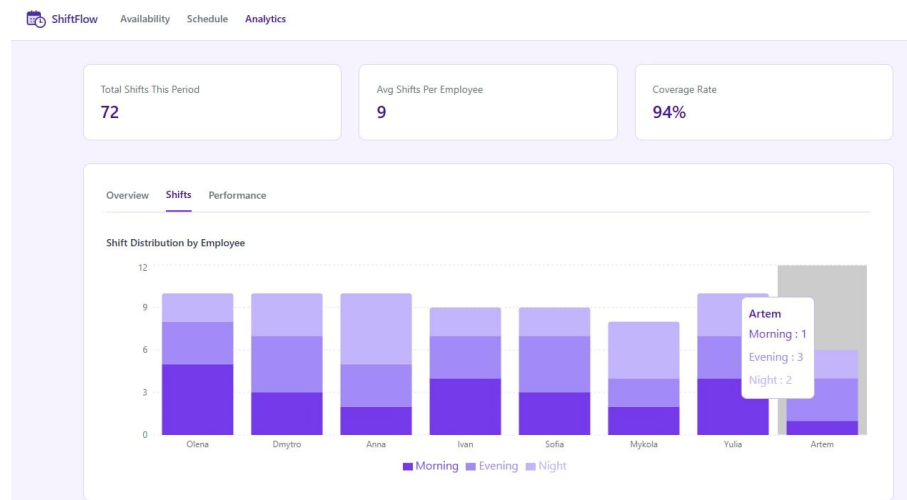


Рисунок 4.21 – Analytics з графічним представленням статистики (Shifts)

У вкладці Performance відображається аналітика ефективності сформованого графіка та оцінка якості розподілу навантаження (рисунок 4.22). Даний розділ дозволяє аналізувати рівномірність призначення змін, кількість нічних чергувань, баланс між працівниками та загальний рівень fairness графіка.

У межах вкладки використовуються:

- показники середнього навантаження працівників;
- статистика нічних змін;
- кількість обмежень;
- оцінка рівномірності розподілу змін;
- аналіз перевантаження окремих працівників.

Подібна аналітика дозволяє швидко виявляти потенційні проблеми графіка та оцінювати ефективність роботи алгоритму Min-Conflicts під час автоматичного формування розкладу.

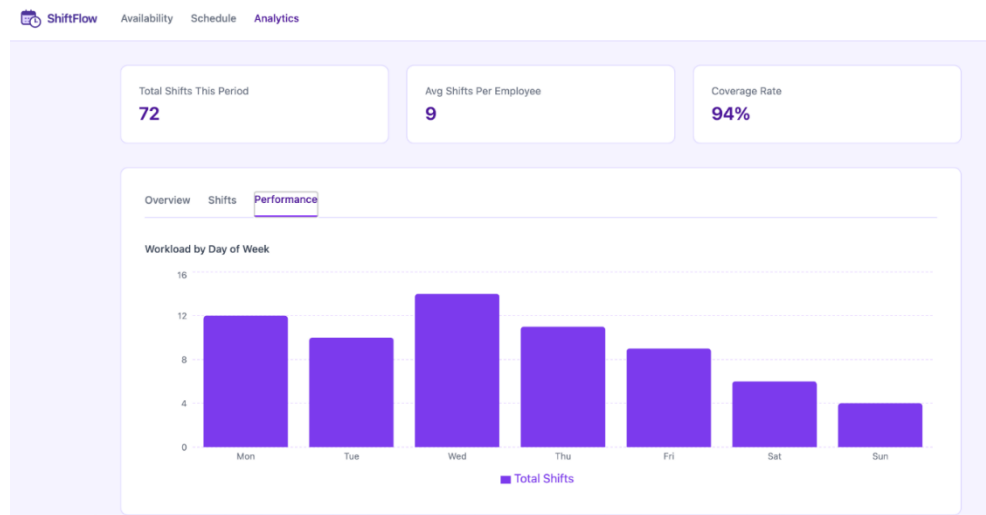


Рисунок 4.22 – Analytics з графічним представленням статистики (Performance)

Таким чином, поєднання календарної, табличної та графічної моделей дозволяє забезпечити ефективне представлення інформації та підтримує різні сценарії аналізу даних, що відповідає вимогам заключного етапу проєктування інтерфейсу. Хороша інтерактивна інформаційна графіка повинна допомагати організовувати дані, виявляти зв'язки між ними, вивчати їх, змінювати порядок перегляду, бачити конкретні значення та показувати лише необхідні фрагменти.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено концепцію та архітектуру системи автоматизованого планування графіків робочих змін із застосуванням алгоритму Min-Conflicts та NLP-компонентів для підтримки взаємодії природною мовою.

Під час виконання роботи було проведено аналіз предметної галузі та досліджено особливості організації змінної роботи у командах із безперервним режимом функціонування. Встановлено, що сучасні системи планування графіків мають низку обмежень, пов'язаних із недостатньою гнучкістю правил, складністю врахування міжперіодних залежностей та обмеженими можливостями персоналізації процесу планування. Аналіз існуючих програмних рішень показав, що більшість із них орієнтована на типові сценарії змінної роботи та не забезпечує повноцінної підтримки складних обмежень і механізмів пояснення рішень.

У межах роботи було спроектовано багатоваріантну архітектуру системи, яка забезпечує розділення frontend, backend, алгоритмічної логіки та логіки роботи з даними. Обраний архітектурний підхід дозволяє забезпечити модульність програмного забезпечення, спростити супровід системи та створити основу для подальшого масштабування функціональності. Для реалізації backend-частини було обрано мову програмування Python та фреймворк FastAPI, що забезпечують підтримку асинхронного програмування, REST API та інтеграцію з ШІ-компонентами. frontend реалізована із використанням React та TypeScript, що дозволило створити інтерактивний користувацький інтерфейс та забезпечити підтримку SPA-архітектури. Для збереження даних використовується PostgreSQL.

Одним із ключових результатів роботи є формалізація задачі планування графіків змін як задачі задоволення обмежень (Constraint

Satisfaction Problem). У роботі реалізовано модель жорстких та м'яких обмежень, яка враховує:

- доступність працівників;
- обмеження між змінами;
- правила роботи з нічними змінами;
- standby-чергування;
- обмеження навантаження;
- вимоги справедливого та рівномірного розподілу змін;
- індивідуальні побажання користувачів.

Для автоматизованого формування графіка використовується алгоритм Min-Conflicts, який реалізує локальний пошук рішення шляхом мінімізації кількості конфліктів між обмеженнями. Використання даного алгоритму дозволяє ефективно працювати з великою кількістю взаємопов'язаних constraint-параметрів та забезпечує достатню гнучкість процесу планування.

У межах роботи також реалізовано NLP-модуль на основі OpenAI API, який дозволяє користувачам взаємодіяти із системою природною мовою. NLP-компонент інтегрований безпосередньо у процес введення даних про доступність та використовується для автоматичної інтерпретації побажань працівників і формування обмежень. Подібний підхід дозволяє спростити процес введення даних та підвищити зручність використання системи.

Було реалізовано користувацькі інтерфейси для:

- введення доступності;
- перегляду графіка змін;
- аналізу навантаження;
- перегляду статистичних даних та метрик справедливості.

Особлива увага приділялася об'єднанню структурованого введення даних через календарний інтерфейс із можливостями NLP-взаємодії. Це дозволяє користувачам обирати найбільш зручний спосіб роботи із системою.

Практичне значення роботи полягає у створенні програмної системи, яка може бути використана для підтримки процесу планування змін у командах технічної підтримки, центрах моніторингу інфраструктури та інших організаціях із безперервним режимом роботи. Запропонований підхід дозволяє зменшити вплив людського фактора, автоматизувати процес обробки обмежень та забезпечити більш справедливий розподіл навантаження між працівниками.

Перспективами подальшого розвитку системи є:

- удосконалення алгоритмів оптимізації графіка;
- реалізація механізмів self-learning constraint analysis;
- розширення NLP-функціональності;
- інтеграція prediction-моделей для аналізу навантаження;
- підтримка мобільних клієнтів;
- реалізація більш складних механізмів аналізу метрик справедливості.

Таким чином, поставлена мета кваліфікаційної роботи була досягнута, а результати роботи підтверджують доцільність використання алгоритму Min-Conflicts та NLP-технологій для побудови систем автоматизованого планування графіків робочих змін.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Matthes E. Python crash course, 3rd edition. No Starch Press, Incorporated, 2022. 552 p.
2. Ramalho L. Fluent Python. O'Reilly Media, 2015. 792 p.
3. Downey A. Think Python: an introduction to software design. CreateSpace Independent Publishing Platform, 2011. 300 p.
4. Scikit-learn / Varoquaux G. et al. *GetMobile: mobile computing and communications*. 2015. Vol. 19, no. 1. P. 29–33. DOI: <https://doi.org/10.1145/2786984.2786995> (date of access: 07.05.2026).
5. Lubanovic B. FastAPI. O'Reilly Media, 2023. 264 p.
6. Lanaro G. Python high performance programming. Packt Publishing, Limited, 2013. 324 p.
7. Fowler M. Patterns of enterprise application architecture. Addison-Wesley Professional, 2002. 560 p.
8. Banks A., Porcello E. Learning React: functional web development with React and Redux. O'Reilly Media, 2017. 350 p.
9. Osmani A. Learning JavaScript design patterns. O'Reilly Media, 2012. 254 p.
10. Pardi P. TypeScript programming language. Independently Published, 2019. 248 p.
11. Understanding TypeScript / Bierman G., Abadi M., Torgersen M. *ECOOP 2014 – Object Oriented Programming*. Berlin, Heidelberg, 2014. P. 257-281. DOI: https://doi.org/10.1007/978-3-662-44202-9_11 (date of access: 07.05.2026).
12. The PostgreSQL Global Development Group. The PostgreSQL reference manual. SQL language reference. Network Theory Ltd., 2007. Vol. 1. 716 p.
13. The PostgreSQL Global Development Group. The PostgreSQL reference manual. Programming guide. Network Theory Ltd., 2007. Vol. 2. 408 p.

14. The PostgreSQL Global Development Group. The PostgreSQL reference manual. Server administration guide. Network Theory Ltd., 2007. Vol. 3. 204 p.
15. Momjian B. PostgreSQL: introduction and concepts. Pearson Education, 2000. 462 p.
16. Douglas K., Douglas S. PostgreSQL. Sams, 2003. 816 p.
17. Yanev M. Building AI applications with OpenAI APIs: leverage ChatGPT, Whisper, and DALL-E APIs to build 10 innovative AI projects, 2nd edition. Packt Publishing, 2024. 252 p.
18. Martin J. H., Jurafsky D. Speech and language processing. Prentice Hall, 2006. 944 p.
19. Russell S. J., Norvig P. Artificial Intelligence. Pearson Education, Limited, 2019. 1152 p.
20. Dechter R. Constraint processing. Morgan Kaufmann, 2003. 481 p.