ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
(повне найменування вищого навчального закладу)
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування інституту, назва факультету (відділення))
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи
магістра
(освітній рівень)

на тему:

«Дослідження використання мови Python для розробки інформаційно-довідкових систем»

Виконав: студент групи 6ПР2
спеціальності
121 – «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Рибас Віталій Володимирович
(прізвище та ініціали)
Керівник к.т.н., доцент Хохлов В.А.
(прізвище та ініціали)
Рецензент к.т.н., доцент Веселовська Г.В.

(прізвище та ініціали)

Хмельницький – 2025

Херсонський національний технічний університет
(повне найменування вищого навчального закладу)

Факультет, відділення **Інформаційних технологій та дизайну**
Кафедра **Програмних засобів і технологій**
Освітній рівень **магістр**
Галузі знань **12 «Інформаційні технології»**
Спеціальність **121 – Інженерія програмного забезпечення**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗіТ
Програмних засобів і технологій
к.т.н. доц. О.Є. Огнєва
"___" _____ 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рибасу Віталію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження використання мови Python для розробки інформаційно-довідкових систем»
керівник роботи к.т.н., доцент Хохлов В.А.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від 15 .09 .2025 р. № 417–с
2. Строк подання студентом роботи 15.12.2025
3. Вихідні дані до роботи літературні та періодичні джерела з тематики інформаційно-довідкових систем і розробки ПЗ; офіційна документація та специфікації Python, Django, Django REST Framework, JWT, PyQt6, SQLite; матеріали переддипломної практики та технічні вимоги до системи; вихідні коди та структура проєкту Optima (серверна і клієнтська частини), тестові дані для перевірки роботи системи
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити):
1. Аналіз предметної області та теоретичних основ інформаційно-довідкових систем; визначення вимог до системи Optima та аналіз аналогів
2. Дослідження мови Python і порівняння з іншими мовами програмування; вибір інструментів та фреймворків для реалізації (Django/DRF, JWT, PyQt6, SQLite)
3. Проєктування інформаційно-довідкової системи Optima: архітектура клієнт-сервер, модель даних, структура REST API, проєктування інтерфейсу користувача
4. Реалізація функціоналу системи засобами Python, інтеграція клієнтської та серверної частин, тестування та аналіз ефективності роботи системи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 29.09.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1	Отримання завдання	29.09.2025	Виконано
2	Підбір і аналіз літературних джерел з	01.10.2025	Виконано
3	Аналіз предметної області та існуючих систем	04.10.2025	Виконано
4	Формування функціональних і нефункціональних вимог до системи	07.10.2025	Виконано
5	Проєктування архітектури клієнт-серверної системи	16.10.2025	Виконано
6	Реалізація серверної частини системи засобами Python	31.10.2025	Виконано
7	Реалізація клієнтської частини системи з графічним інтерфейсом користувача	10.11.2025	Виконано
8	Інтеграція клієнтської та серверної частин, реалізація обміну даними через REST API	20.11.2025	Виконано
9	Тестування системи, оптимізація роботи, виправлення виявлених помилок	30.11.2025	Виконано
10	Оформлення пояснювальної записки	08.12.2025	Виконано
11	Підготовка до захисту та захист кваліфікаційної роботи	11.12.2025	Виконано
12	Захист кваліфікаційної роботи	15.12.2025	Виконано

Студент _____ В.В.Рибас
(підпис) (прізвище та ініціали)

Керівник роботи _____ В.А. Хохлов
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота присвячена дослідженню та розробленню інформаційно-довідкової системи Optima з використанням мови програмування Python. У роботі розглянуто процес створення клієнт-серверної системи, призначеної для зберігання, обробки та наочного відображення довідкових даних про події у вигляді календаря. Основну увагу приділено реалізації серверної частини на базі фреймворків Django та Django REST Framework, а також клієнтської частини у вигляді настільного застосунку з графічним інтерфейсом користувача.

У роботі висвітлено архітектуру системи Optima, принципи організації REST API, механізми автентифікації та авторизації користувачів із використанням JWT, проєктування моделі даних та інтеграцію клієнтської і серверної частин.

У практичній частині виконано проєктування структури системи, реалізацію серверного та клієнтського функціоналу, інтеграцію компонентів через HTTP-запити та тестування основних сценаріїв роботи системи.

Результатом дипломної роботи є повноцінна інформаційно-довідкова система Optima, яка забезпечує централізоване збереження даних, зручний календарний інтерфейс, контроль доступу до інформації та можливість подальшого розширення і вдосконалення функціоналу.

Звіт містить 85 сторінок, 11 рисунків, 8 таблиць, 30 використаних джерел.

Ключові слова: PYTHON, DJANGO, DJANGO REST FRAMEWORK, PYQT6, INFORMATION SYSTEM, INFORMATION-REFERENCE SYSTEM, CLIENT-SERVER ARCHITECTURE, REST API, JWT, CALENDAR, DATABASE, USER INTERFACE.

АНОТАЦІЯ

Дипломна робота магістра присвячена дослідженню та розробленню інформаційно-довідкової системи Optima з використанням мови програмування Python. Система призначена для зберігання, обробки та наочного відображення довідкових даних про події у вигляді календаря з підтримкою клієнт-серверної взаємодії. Структура роботи складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. У вступі обґрунтовано актуальність теми, визначено мету дослідження, основні завдання та методологічні засади.

У першому розділі наведено теоретичні основи інформаційно-довідкових систем, розглянуто їх призначення, функції та вимоги до сучасних прикладних систем такого типу. Проаналізовано особливості організації довідкових даних та принципи їх подання користувачеві. У другому розділі здійснено аналіз мови програмування Python і порівняння її з іншими мовами програмування, досліджено та обґрунтовано вибір фреймворків і інструментів для реалізації системи, зокрема Django, Django REST Framework, JWT та PyQt6.

Третій розділ присвячено проєктуванню інформаційно-довідкової системи Optima. У ньому розглянуто аналіз предметної області, проєктування клієнт-серверної архітектури, структури бази даних, REST API та інтерфейсу користувача з календарним представленням подій. Четвертий розділ містить опис реалізації функціоналу системи засобами Python, інтеграції клієнтської та серверної частин, а також тестування і аналіз ефективності роботи системи в умовах практичного використання.

У висновках підсумовано результати виконаної роботи, підтверджено відповідність реалізованої системи поставленим завданням та визначено перспективи її подальшого розвитку. У додатках наведено фрагменти програмного коду, структуру бази даних та матеріали інтерфейсу користувача.

ABSTRACT

The master’s thesis is devoted to the research and development of the information and reference system Optima using the Python programming language. The system is designed for storing, processing, and visualizing reference data about events in a calendar-based format with client–server interaction. The thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices. The introduction substantiates the relevance of the topic, defines the purpose of the research, the main objectives, and the methodological basis.

The first chapter presents the theoretical foundations of information and reference systems, their purpose, functions, and requirements for modern applied solutions of this type. Special attention is paid to the organization of reference data and principles of user-oriented data presentation. The second chapter analyzes the Python programming language and compares it with other programming languages. It also justifies the selection of frameworks and tools used for system implementation, including Django, Django REST Framework, JWT authentication, and PyQt6.

The third chapter focuses on the design of the Optima information and reference system. It covers the analysis of the subject domain, the design of the client–server architecture, database structure, REST API, and the user interface based on calendar visualization. The fourth chapter describes the implementation of the system functionality using Python, the integration of client and server components, as well as testing and evaluation of system efficiency under practical usage conditions.

The conclusions summarize the results of the research, confirm that the developed system meets the defined objectives, and outline prospects for further development. The appendices contain source code fragments, database structure descriptions, and user interface materials.

ЗМІСТ

ВСТУП

У сучасних умовах розвитку інформаційних технологій все більше організацій використовують програмні засоби для зберігання, обробки та надання довідкової інформації. Інформаційно-довідкові системи відіграють важливу роль у діяльності підприємств, навчальних закладів та державних установ, оскільки дозволяють впорядковувати дані, спрощувати робочі процеси та підвищувати ефективність прийняття управлінських рішень. Використання таких систем дає можливість зменшити кількість ручної роботи, уникнути помилок, пов’язаних з людським фактором, а також забезпечити швидкий доступ до необхідної інформації.

Особливу увагу в процесі створення інформаційно-довідкових систем приділяють вибору мови програмування та інструментів розробки. Від цього залежить зручність реалізації функціоналу, надійність системи, її продуктивність та можливість подальшого розвитку. Однією з мов програмування, яка активно використовується для створення прикладного програмного забезпечення, є мова Python. Вона здобула популярність завдяки простому синтаксису, широкому набору бібліотек та можливості застосування у різних сферах, зокрема для розробки настільних і веб-застосунків, роботи з базами даних та створення інформаційних систем.

Актуальність даної дипломної роботи зумовлена зростаючою потребою у простих, зручних та надійних інформаційно-довідкових системах, які можуть бути швидко впроваджені у робочий процес організацій. Мова Python дозволяє реалізувати такі системи з меншими затратами часу та ресурсів у порівнянні з багатьма іншими мовами програмування. Завдяки наявності великої кількості готових рішень та активної спільноти розробників Python є ефективним інструментом для створення програмних продуктів різного рівня складності.

Метою дипломної роботи є дослідження можливостей використання мови програмування Python для розробки інформаційно-довідкових систем, а також

практична реалізація такої системи на прикладі програмного продукту Optima. Для досягнення поставленої мети в роботі необхідно виконати низку завдань: проаналізувати поняття та особливості інформаційно-довідкових систем, розглянути основні переваги та недоліки мови Python, дослідити інструменти та бібліотеки, які можуть бути використані для розробки подібних систем, спроєктувати архітектуру інформаційно-довідкової системи та реалізувати її основний функціонал.

Об’єктом дослідження у даній роботі є процес розробки інформаційно-довідкових систем. Предметом дослідження є використання мови програмування Python та відповідних програмних засобів для створення інформаційно-довідкових систем. У ході виконання дипломної роботи застосовуються такі методи дослідження, як аналіз наукових джерел та технічної документації, порівняння програмних засобів, проєктування програмної архітектури, а також тестування розробленого програмного продукту.

Практичне значення роботи полягає у створенні інформаційно-довідкової системи Optima, яка може бути використана для обліку та перегляду довідкових даних у межах організації. Результати дослідження можуть бути корисними для студентів, початківців у програмуванні та розробників, які планують використовувати мову Python для створення прикладних інформаційних систем. Запропоновані підходи та реалізований функціонал можуть бути використані як основа для подальшого розвитку та вдосконалення подібних програмних рішень.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ІНФОРМАЦІЙНО-ДОВІДКОВИХ СИСТЕМ

1.1. Поняття та призначення інформаційно-довідкових систем

У сучасному суспільстві інформація відіграє важливу роль у всіх сферах діяльності людини. Кількість даних, з якими доводиться працювати організаціям, постійно зростає, що ускладнює їх зберігання, обробку та використання без застосування спеціалізованих програмних засобів. Саме тому все більшого значення набувають інформаційні системи, зокрема інформаційно-довідкові системи, які забезпечують упорядкування даних та швидкий доступ до необхідної інформації.

Інформаційно-довідкова система – це програмно-апаратний комплекс, призначений для збирання, зберігання, обробки та надання довідкової інформації користувачам у зручному для них вигляді. Основною метою таких систем є забезпечення оперативного доступу до актуальних даних, що використовуються для прийняття рішень, виконання службових обов’язків або отримання довідкових відомостей. На відміну від складних аналітичних систем, інформаційно-довідкові системи орієнтовані насамперед на пошук, перегляд та систематизацію інформації.

Інформаційно-довідкові системи широко застосовуються у різних сферах діяльності. У державних установах вони використовуються для обліку персоналу, ведення реєстрів, зберігання нормативних документів та довідкових матеріалів. У навчальних закладах такі системи застосовуються для управління навчальним процесом, обліку студентів та викладачів, формування розкладів і збереження методичних матеріалів. У комерційних організаціях інформаційно-довідкові системи використовуються для роботи з клієнтськими базами, обліку

співробітників, управління внутрішніми процесами та контролю інформаційних потоків. [4]

Однією з основних причин впровадження інформаційно-довідкових систем є необхідність зменшення обсягу ручної роботи. Використання паперових носіїв або неструктурованих електронних документів призводить до втрати часу, підвищує ймовірність помилок та ускладнює пошук потрібної інформації. Інформаційно-довідкові системи дозволяють автоматизувати рутинні процеси, забезпечити єдину структуру даних та зменшити навантаження на працівників.

Важливим аспектом інформаційно-довідкових систем є централізоване зберігання даних. У таких системах інформація зазвичай зберігається у базах даних, що забезпечує її цілісність, актуальність та захищеність. Централізований підхід дозволяє уникнути дублювання інформації, спростити її оновлення та забезпечити доступ до даних для різних категорій користувачів відповідно до їх прав.

Призначення інформаційно-довідкових систем полягає не лише у зберіганні даних, але й у забезпеченні зручного механізму їх пошуку та перегляду. Користувач повинен мати можливість швидко знайти потрібну інформацію за заданими параметрами, переглянути її у зрозумілому вигляді та, за потреби, сформувати звіт або отримати узагальнені відомості. Саме тому при розробці таких систем значну увагу приділяють інтерфейсу користувача та логіці взаємодії з даними.

Інформаційно-довідкові системи можуть бути як автономними, так і інтегрованими з іншими програмними продуктами. Автономні системи зазвичай використовуються у невеликих організаціях або для вирішення вузькоспеціалізованих задач. Інтегровані системи, у свою чергу, можуть взаємодіяти з бухгалтерськими програмами, системами управління персоналом або іншими інформаційними ресурсами, утворюючи єдиний інформаційний простір організації.

Ще одним важливим призначенням інформаційно-довідкових систем є підтримка управлінських рішень. Наявність актуальної та впорядкованої інформації дозволяє керівникам швидко оцінювати поточну ситуацію, аналізувати дані та приймати обґрунтовані рішення. Навіть прості довідкові системи можуть значно підвищити ефективність управління за рахунок скорочення часу на пошук необхідних відомостей.

З розвитком інформаційних технологій змінюються і вимоги до інформаційно-довідкових систем. Сучасні користувачі очікують від програмного забезпечення зручності, надійності та швидкої роботи. Система повинна бути зрозумілою навіть для користувачів без спеціальної підготовки, мати логічну структуру та мінімальну кількість складних налаштувань. Важливим фактором є також можливість масштабування системи та її адаптації до змін у діяльності організації.

Окрему увагу слід приділити питанням безпеки інформації. Інформаційно-довідкові системи часто містять персональні дані, службову або конфіденційну інформацію, що потребує належного захисту. Це передбачає використання механізмів авторизації та автентифікації користувачів, розмежування прав доступу, а також захисту даних від несанкціонованого доступу та втрати.

Таким чином, інформаційно-довідкові системи є важливим елементом сучасного інформаційного середовища. Вони забезпечують ефективне зберігання та використання даних, підвищують продуктивність праці та сприяють оптимізації робочих процесів. Завдяки своїй універсальності та відносній простоті реалізації такі системи можуть бути адаптовані до потреб різних організацій та сфер діяльності.[9]

У контексті даної дипломної роботи інформаційно-довідкові системи розглядаються як програмні рішення, що можуть бути реалізовані з використанням мови програмування Python. Поєднання простоти реалізації, широких можливостей роботи з даними та наявності готових бібліотек робить

Python зручним інструментом для створення інформаційно-довідкових систем різного рівня складності. Подальші розділи роботи присвячені аналізу можливостей мови Python та практичній реалізації інформаційно-довідкової системи Optima.

1.2. Класифікація інформаційно-довідкових систем

Інформаційно-довідкові системи є різновидом інформаційних систем і застосовуються для зберігання, обробки та надання користувачам довідкової інформації. Залежно від сфери використання, технічної реалізації та функціонального призначення такі системи можуть суттєво відрізнятися між собою. Саме тому для кращого розуміння їх можливостей та особливостей доцільно розглянути основні підходи до класифікації інформаційно-довідкових систем.

Класифікація інформаційно-довідкових систем дозволяє визначити їх місце серед інших програмних рішень, а також спрощує вибір оптимального типу системи для конкретних умов використання. Поділ таких систем здійснюється за різними ознаками, серед яких найбільш поширеними є спосіб доступу, масштаб використання, тип збереження даних, функціональне призначення та рівень інтеграції з іншими інформаційними ресурсами.

Однією з основних ознак класифікації є спосіб доступу користувачів до інформаційно-довідкової системи. За цією ознакою виділяють локальні, клієнт-серверні та веб-орієнтовані системи. Локальні інформаційно-довідкові системи встановлюються та використовуються на одному комп’ютері без необхідності підключення до мережі. Вони зазвичай застосовуються у невеликих організаціях або для вирішення вузькоспеціалізованих задач. Основними перевагами таких систем є простота впровадження та відсутність залежності від мережевого з’єднання, проте їх недоліком є обмежена можливість спільної роботи кількох користувачів.

Клієнт-серверні інформаційно-довідкові системи передбачають наявність центрального сервера, на якому зберігаються дані, та клієнтських програм, через які користувачі отримують доступ до інформації. Такий підхід дозволяє забезпечити централізоване зберігання даних, контроль доступу та синхронну роботу кількох користувачів. Клієнт-серверні системи широко застосовуються у середніх та великих організаціях, де важливо забезпечити єдиний інформаційний простір та актуальність даних.

Веб-орієнтовані інформаційно-довідкові системи працюють через веб-браузер і не потребують встановлення спеціального програмного забезпечення на комп'ютерах користувачів. Доступ до таких систем здійснюється через мережу Інтернет або локальну мережу. Основною перевагою веб-систем є зручність використання та можливість доступу з будь-якого пристрою. Разом з тим, такі системи залежать від стабільності мережевого з'єднання та потребують додаткових заходів безпеки.

Ще однією важливою ознакою класифікації є масштаб використання інформаційно-довідкових систем. За цією ознакою виділяють персональні, групові та корпоративні системи. Персональні системи призначені для використання однією особою і зазвичай мають обмежений функціонал. Вони можуть застосовуватися для зберігання особистих нотаток, контактів або іншої довідкової інформації. Групові системи орієнтовані на роботу невеликого колективу та дозволяють кільком користувачам спільно працювати з даними. Корпоративні інформаційно-довідкові системи використовуються у великих організаціях та охоплюють значні обсяги інформації, забезпечуючи доступ до неї для різних підрозділів.

За типом збереження та обробки даних інформаційно-довідкові системи поділяються на файлові та бази даних. Файлові системи зберігають інформацію у вигляді окремих файлів, що може бути зручно для невеликих обсягів даних. Однак із зростанням кількості інформації та користувачів такий підхід стає менш ефективним. Системи, що використовують бази даних, дозволяють зберігати великі обсяги структурованої інформації, забезпечувати швидкий

пошук та підтримувати цілісність даних. Саме цей підхід є найбільш поширеним у сучасних інформаційно-довідкових системах.

За функціональним призначенням інформаційно-довідкові системи можуть бути універсальними або спеціалізованими. Універсальні системи призначені для роботи з різними типами довідкової інформації та можуть використовуватися у різних сферах діяльності. Спеціалізовані системи орієнтовані на конкретну предметну область, наприклад, облік персоналу, медичні довідники, навчальні бази даних або інформаційні системи для органів влади. Такий поділ дозволяє створювати програмні продукти, максимально адаптовані до потреб конкретних користувачів.

Окремо слід виділити класифікацію за рівнем інтеграції з іншими інформаційними системами. Ізольовані інформаційно-довідкові системи працюють незалежно та не обмінюються даними з іншими програмними продуктами. Інтегровані системи, навпаки, можуть взаємодіяти з бухгалтерськими програмами, системами управління персоналом або іншими інформаційними ресурсами. Такий підхід дозволяє уникнути дублювання даних та забезпечити їх актуальність у різних системах.

Також інформаційно-довідкові системи можна класифікувати за характером оновлення інформації. Статичні системи містять інформацію, яка змінюється рідко або не змінюється взагалі. Динамічні системи передбачають регулярне оновлення даних та підтримують внесення змін у реальному часі. У сучасних умовах більшість інформаційно-довідкових систем належать до динамічних, оскільки потреба в актуальній інформації є надзвичайно високою.

За рівнем доступу користувачів інформаційно-довідкові системи поділяються на відкриті та обмежені. Відкриті системи надають доступ до інформації широкому колу користувачів, наприклад, через мережу Інтернет. Обмежені системи використовуються всередині організацій та передбачають обов’язкову авторизацію користувачів. Такий підхід дозволяє забезпечити захист інформації та розмежувати права доступу.

Отже, інформаційно-довідкові системи можуть бути класифіковані за багатьма ознаками, що свідчить про їх різноманітність та універсальність. Розуміння основних типів таких систем дозволяє більш обґрунтовано підходити до вибору технологій та засобів їх реалізації. Вимоги до сучасних інформаційно-довідкових систем

Сучасні інформаційно-довідкові системи є важливим інструментом у діяльності організацій різного типу. Вони використовуються для зберігання, обробки та надання довідкової інформації, що безпосередньо впливає на ефективність роботи користувачів. У зв’язку з цим до таких систем висувається низка вимог, які визначають їх якість, зручність використання та можливість подальшого розвитку. Дотримання цих вимог є необхідною умовою створення надійного та практичного програмного продукту.

Вимоги до інформаційно-довідкових систем формуються з урахуванням потреб користувачів, умов експлуатації та технічних можливостей сучасних інформаційних технологій. Вони охоплюють як функціональні, так і нефункціональні аспекти роботи системи, включаючи зручність інтерфейсу, швидкодію, безпеку та надійність збереження даних.

Функціональні вимоги визначають, які саме задачі повинна виконувати інформаційно-довідкова система. До основних функціональних вимог належать:

1. Зберігання довідкової інформації у структурованому вигляді;
2. Пошук інформації за заданими параметрами;
3. Перегляд та оновлення даних відповідно до прав користувачів;
4. Підтримка роботи з різними типами довідкової інформації;
5. Формування звітів або узагальнених відомостей.

Інформаційно-довідкова система повинна забезпечувати швидкий доступ до актуальної інформації та мінімізувати кількість дій, необхідних для виконання основних операцій. Надмірно складна логіка або перевантажений функціонал можуть негативно впливати на зручність використання системи.

Однією з найважливіших вимог до сучасних інформаційно-довідкових систем є зручність використання. Інтерфейс користувача повинен бути інтуїтивно зрозумілим та логічно побудованим. Користувач повинен мати змогу працювати із системою без тривалого навчання або спеціальної підготовки.

До основних вимог у цій групі належать:

1. Простота та зрозумілість інтерфейсу;
2. Логічне розташування елементів управління;
3. Зрозумілі назви кнопок, полів та повідомлень;
4. Мінімальна кількість дій для виконання типових операцій.

Зручний інтерфейс сприяє підвищенню продуктивності праці та зменшує кількість помилок під час роботи з системою.

Інформаційно-довідкова система повинна працювати стабільно та без збоїв упродовж тривалого часу. Надійність системи полягає у її здатності коректно виконувати задані функції навіть за умов підвищеного навантаження або виникнення помилок.

Основними вимогами до надійності є:

1. Захист від втрати даних;
2. Коректна обробка помилок;
3. Можливість відновлення роботи після збоїв;
4. Збереження цілісності даних.

Недостатній рівень надійності може призвести до втрати важливої інформації та негативно вплинути на діяльність організації.

Швидкодія інформаційно-довідкової системи безпосередньо впливає на комфорт роботи користувачів. Система повинна швидко реагувати на дії користувача, виконувати пошук інформації та обробляти запити без значних затримок.

Продуктивність системи особливо важлива у випадках роботи з великими обсягами даних або при одночасному доступі кількох користувачів. Для

забезпечення належної швидкодії необхідно використовувати ефективні алгоритми обробки даних та оптимізовані засоби зберігання інформації.

Інформаційно-довідкові системи часто містять персональні або службові дані, що потребують захисту від несанкціонованого доступу. Забезпечення інформаційної безпеки є однією з ключових вимог до таких систем.

До основних вимог безпеки належать:

1. Авторизація та автентифікація користувачів;
2. Розмежування прав доступу;
3. Захист даних від несанкціонованих змін;
4. Резервне копіювання інформації.

Реалізація цих вимог дозволяє зменшити ризик витоку або втрати інформації.

Сучасні інформаційно-довідкові системи повинні мати можливість подальшого розвитку та адаптації до змін. Масштабованість передбачає здатність системи працювати з більшими обсягами даних або більшою кількістю користувачів без суттєвого зниження продуктивності.

Гнучкість системи полягає у можливості додавання нового функціоналу, зміни структури даних або інтеграції з іншими програмними продуктами без повної переробки системи.

Таблиця 1.1 – Узагальнена таблиця вимог до інформаційно-довідкових систем

Група вимог	Коротка характеристика
Функціональні	Забезпечують виконання основних задач системи
Зручність використання	Гарантують простоту та зрозумілість інтерфейсу
Надійність	Забезпечують стабільну роботу та збереження даних
Швидкодія	Визначають швидкість обробки інформації

Безпека	Захищають дані від несанкціонованого доступу
Масштабованість	Дозволяють розширювати систему у майбутньому

Отже, сучасна інформаційно-довідкова система повинна відповідати комплексу вимог, які охоплюють усі аспекти її роботи – від зручності інтерфейсу до безпеки та продуктивності. Дотримання цих вимог дозволяє створити ефективний програмний продукт, здатний забезпечити якісну роботу з довідковою інформацією.

РОЗДІЛ 2. АНАЛІЗ МОВИ PYTHON ТА ЗАСОБІВ РОЗРОБКИ

2.1. Загальна характеристика мови програмування Python

Мова програмування Python є однією з найпопулярніших мов у сучасній розробці програмного забезпечення. Вона широко використовується для створення настільних застосунків, веб-систем, інформаційних систем, автоматизації процесів, аналізу даних та багатьох інших завдань. Популярність Python пояснюється простотою синтаксису, зрозумілою логікою побудови програм та наявністю великої кількості готових бібліотек і інструментів.

Python був створений наприкінці 1980-х років і вперше представлений у 1991 році. Його розробником є Гвідо ван Россум, який ставив за мету створити мову програмування, що буде простою у вивченні та зручною у використанні. Основною ідеєю Python є читабельність коду, що дозволяє розробникам швидко розуміти логіку програм навіть без детального вивчення їх внутрішньої структури. Саме ця особливість зробила Python популярним як серед початківців, так і серед досвідчених програмістів.[12]

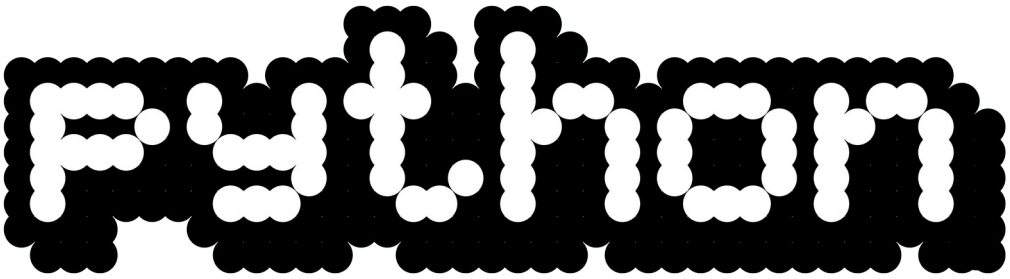

Рисунок 2.1 – Перший логотип мови Python

Python належить до мов програмування високого рівня, що означає його орієнтацію на зручність розробника, а не на роботу з апаратним забезпеченням. Код, написаний мовою Python, є незалежним від конкретної операційної системи, що забезпечує його портативність. Програми, створені на Python, можуть працювати на різних платформах без значних змін у вихідному коді.

Однією з ключових характеристик Python є його простий та зрозумілий синтаксис. У цій мові відсутні складні конструкції, характерні для деяких інших мов програмування. Наприклад, для визначення блоків коду використовується відступ, а не спеціальні символи. Такий підхід сприяє написанню акуратного та логічно структурованого коду, що особливо важливо при розробці великих програмних проєктів, зокрема інформаційно-довідкових систем.

Python є інтерпретованою мовою програмування. Це означає, що програмний код виконується без попередньої компіляції, що спрощує процес розробки та тестування програм. Інтерпретація коду дозволяє швидко знаходити та виправляти помилки, що позитивно впливає на швидкість створення програмного продукту. Разом з тим, сучасні реалізації Python забезпечують достатню продуктивність для більшості прикладних задач.

Ще однією важливою характеристикою Python є підтримка різних парадигм програмування. Мова дозволяє використовувати процедурний, об’єктно-орієнтований та функціональний підходи. Об’єктно-орієнтоване програмування у Python реалізоване досить просто, що дозволяє створювати зрозумілі та масштабовані програмні системи. Це є важливим фактором при розробці інформаційно-довідкових систем, які часто мають складну структуру та велику кількість взаємопов’язаних компонентів.

Python має велику стандартну бібліотеку, яка містить готові модулі для роботи з файлами, мережею, базами даних, датами та часом, а також для обробки тексту та даних. Наявність стандартної бібліотеки значно зменшує потребу у використанні сторонніх рішень та прискорює процес розробки програмного забезпечення. Крім того, Python підтримує використання сторонніх бібліотек, які можна легко підключати за допомогою спеціальних інструментів керування пакетами.

Важливою перевагою Python є активна спільнота розробників. Завдяки цьому постійно створюються нові бібліотеки, фреймворки та інструменти, які розширюють можливості мови. Для більшості типових задач можна знайти готові рішення або приклади реалізації, що значно спрощує процес розробки. Наявність великої кількості навчальних матеріалів також робить Python доступним для самостійного вивчення.[14]

Python широко використовується для створення інформаційно-довідкових систем завдяки зручним засобам роботи з даними. Мова підтримує різні типи баз даних, включаючи локальні та серверні рішення. За допомогою відповідних бібліотек можна легко організувати збереження, пошук та оновлення довідкової інформації. Це робить Python придатним для створення систем, орієнтованих на роботу з великими обсягами структурованих даних.

Рисунок 2.2 – Перелік галузей де використовується Python

Окрему увагу слід приділити можливостям Python у сфері створення користувацьких інтерфейсів. Існує низка бібліотек, які дозволяють розробляти графічні інтерфейси для настільних застосунків. Це дає змогу створювати інформаційно-довідкові системи з зручним та зрозумілим інтерфейсом, що відповідає вимогам сучасних користувачів.

Ще однією важливою характеристикою Python є простота супроводу та розвитку програмного коду. Завдяки читабельності та логічній структурі коду, внесення змін або розширення функціоналу не потребує значних зусиль. Це особливо важливо для інформаційно-довідкових систем, які з часом можуть потребувати оновлення або доповнення новими можливостями.

Python також підтримує інтеграцію з іншими мовами програмування та технологіями. За потреби можна використовувати компоненти, написані на інших мовах, що дозволяє поєднувати простоту Python із продуктивністю

низькорівневих рішень. Така гнучкість розширює можливості застосування Python у складних програмних проєктах.[21]

Таким чином, мова програмування Python має низку характеристик, які роблять її ефективним інструментом для розробки інформаційно-довідкових систем. Простий синтаксис, підтримка різних парадигм програмування, велика кількість бібліотек та активна спільнота розробників дозволяють створювати надійні та зручні програмні продукти. Саме ці особливості зумовили вибір Python як основного засобу реалізації інформаційно-довідкової системи Optima, що розглядається у даній дипломній роботі.

2.2. Порівняння Python з іншими мовами програмування

Вибір мови програмування є одним з ключових етапів у процесі розробки інформаційно-довідкових систем. Від цього вибору залежить швидкість створення програмного продукту, його надійність, зручність супроводу та можливість подальшого розвитку. Сучасний ринок програмного забезпечення пропонує велику кількість мов програмування, кожна з яких має свої особливості, переваги та обмеження. У цьому контексті доцільно здійснити порівняння мови Python з іншими популярними мовами програмування, які часто використовуються для створення інформаційних систем.

Python суттєво відрізняється від багатьох мов програмування за своєю філософією. Основний акцент у Python зроблено на простоті, читабельності та швидкості розробки. На відміну від мов, у яких значна увага приділяється суворому контролю типів або складним синтаксичним конструкціям, Python дозволяє розробнику зосередитися безпосередньо на логіці роботи програми. Це є важливою перевагою при створенні інформаційно-довідкових систем, де часто необхідно швидко реалізовувати бізнес-логіку та працювати з великими обсягами даних.

Порівнюючи Python з мовою Java, слід зазначити, що Java є статично типізованою мовою, яка вимагає чіткого визначення типів даних на етапі написання коду. Такий підхід підвищує надійність програм, однак ускладнює процес розробки та збільшує обсяг коду. Python, будучи динамічно типізованою мовою, дозволяє значно скоротити кількість службових конструкцій, що робить код компактнішим і зрозумілішим. У випадку інформаційно-довідкових систем це дає змогу швидше реалізувати основний функціонал та оперативно вносити зміни у програму.[27]

Ще однією відмінністю між Python та Java є підхід до розробки користувацьких інтерфейсів. У Java для цього часто використовуються складні фреймворки, які потребують значних зусиль для налаштування. Python пропонує простіші інструменти, що дозволяють швидко створювати графічні інтерфейси для настільних застосунків. Це є важливим фактором при розробці інформаційно-довідкових систем, орієнтованих на кінцевого користувача.

Порівняння Python з мовою C# також є доцільним, оскільки C# широко використовується для створення корпоративних інформаційних систем. C# має тісну інтеграцію з платформою .NET, що забезпечує високу продуктивність та зручність роботи у середовищі Windows. Водночас Python є більш універсальним у плані платформної незалежності. Програми, написані на Python, можуть працювати на різних операційних системах без суттєвих змін, що робить цю мову більш гнучкою у використанні.

З точки зору швидкості розробки Python має значну перевагу над C#. Завдяки простому синтаксису та великій кількості готових бібліотек, розробник може реалізувати необхідний функціонал за менший проміжок часу. Для інформаційно-довідкових систем, які часто створюються у стислі терміни або потребують швидкого прототипування, ця особливість є вирішальною.

Порівнюючи Python з мовами сімейства C та C++, слід зазначити, що ці мови орієнтовані на високу продуктивність та роботу з низькорівневими ресурсами. Вони широко використовуються для розробки системного

програмного забезпечення, драйверів та високопродуктивних застосунків. Проте складність синтаксису та необхідність ручного керування пам’яттю роблять їх менш зручними для створення інформаційно-довідкових систем. Python, у свою чергу, приховує більшість технічних деталей від розробника, що дозволяє зосередитися на реалізації логіки системи, а не на технічних нюансах.

Важливим аспектом порівняння є продуктивність виконання програм. Традиційно Python вважається повільнішим за компільовані мови, такі як C++ або Java. Однак для більшості інформаційно-довідкових систем цей недолік не є критичним, оскільки основні операції пов’язані з обробкою запитів до бази даних та роботою з інтерфейсом користувача. Крім того, сучасні бібліотеки та оптимізовані середовища виконання дозволяють значно підвищити ефективність Python-програм.

Порівнюючи Python з мовами веб-розробки, такими як JavaScript, слід зазначити, що JavaScript є основною мовою для створення клієнтської частини веб-застосунків. Водночас Python частіше використовується для реалізації серверної логіки або створення настільних застосунків. Для інформаційно-довідкових систем Python є більш універсальним рішенням, оскільки дозволяє створювати як серверну, так і клієнтську частину системи з використанням єдиної мови програмування.

Таблиця 2.1 – Порівняння Python з іншими мовами програмування за основними характеристиками

Характеристика	Python	Java	C#	JavaScript
Типізація	Динамічна	Статична	Статична	Динамічна
Синтаксис	Простий, читабельний	Формальний, об’ємний	Формальний	Гнучкий, але складніший
Швидкість розробки	Висока	Середня	Середня	Середня
Портативність	Висока	Висока	Обмежена платформою .NE	Висока

			T	
Продуктивність	Середня	Висока	Висока	Середня
Порог входження	Низький	Середній	Середній	Середній
Зручність супроводу	Висока	Середня	Середня	Середня
Сфера застосування	ІДС, веб, автоматизація	Корпоративні системи	Корпоративні системи	Веб-застосунки

JavaScript має асинхронну модель виконання, що є перевагою для веб-застосунків із великою кількістю одночасних користувачів. Проте ця особливість ускладнює логіку програм та може бути складною для розуміння початківцями. Python пропонує більш просту модель програмування, що робить його зручним для створення інформаційно-довідкових систем, де основна увага приділяється роботі з даними, а не обробці великої кількості паралельних запитів.

Ще одним важливим критерієм порівняння є зручність супроводу програмного коду. Python-код зазвичай є коротшим та читабельнішим у порівнянні з кодом, написаним на багатьох інших мовах. Це спрощує процес внесення змін, пошуку помилок та розширення функціоналу системи. Для інформаційно-довідкових систем, які часто використовуються протягом тривалого часу та потребують регулярного оновлення, ця характеристика має особливе значення.

Не менш важливим фактором є наявність бібліотек та інструментів для роботи з базами даних. Python підтримує широкий спектр баз даних і надає прості засоби для організації збереження та обробки інформації. У порівнянні з деякими іншими мовами, робота з базами даних у Python є менш складною та більш наочною, що позитивно впливає на швидкість розробки інформаційно-довідкових систем.

Таблиця 2.2 – Оцінка придатності мов програмування для розробки інформаційно-довідкових систем

Критерій оцінки	Python	Java	C#	JavaScript
Зручність роботи з даними	Висока	Висока	Висока	Середня
Робота з базами даних	Проста	Складніша	Складніша	Обмежена
Створення настільних застосунків	Висока	Середня	Висока	Низька
Швидке прототипування	Високе	Низьке	Середнє	Середнє
Масштабованість	Висока	Висока	Висока	Висока
Підтримка спільнотою	Дуже висока	Висока	Висока	Дуже висока
Доцільність для ІДС	Висока	Середня	Середня	Обмежена

Також слід звернути увагу на навчальну складову. Python часто використовується як перша мова програмування у навчальних закладах завдяки своїй простоті та зрозумілості. Це означає, що фахівців, які володіють Python, значно легше підготувати або залучити до роботи над проєктом. У порівнянні з мовами, які мають складніші концепції та синтаксис, Python є більш доступним для широкого кола користувачів.[22]

Таким чином, порівняльний аналіз Python з іншими мовами програмування показує, що Python є ефективним інструментом для розробки інформаційно-довідкових систем. Його простота, гнучкість, універсальність та широка підтримка спільнотою роблять цю мову зручним вибором для створення програмних продуктів, орієнтованих на роботу з довідковою інформацією. Незважаючи на окремі обмеження щодо продуктивності, Python повністю відповідає вимогам, які висуваються до сучасних інформаційно-довідкових систем, що підтверджує доцільність його використання у межах даної дипломної роботи.

2.3. Фреймворки та бібліотеки Python для розробки інформаційно-довідкових систем

Розробка сучасних інформаційно-довідкових систем неможлива без використання спеціалізованих програмних засобів, які спрощують роботу з даними, інтерфейсом користувача та взаємодією між різними компонентами системи. Мова програмування Python має широкий набір фреймворків і бібліотек, що дозволяють створювати інформаційно-довідкові системи різного рівня складності – від простих локальних програм до клієнт-серверних і веб-орієнтованих рішень. Саме наявність таких інструментів є однією з ключових причин популярності Python у сфері прикладного програмування.

Фреймворки та бібліотеки Python забезпечують готові механізми для реалізації типових задач, з якими стикається розробник під час створення інформаційно-довідкових систем. До таких задач належать робота з базами даних, формування користувацького інтерфейсу, обробка запитів користувачів, обмін даними між клієнтською та серверною частинами, а також забезпечення безпеки та стабільності роботи системи. Використання готових інструментів дозволяє суттєво скоротити час розробки та зменшити кількість помилок.

Одним з важливих напрямів використання Python у контексті інформаційно-довідкових систем є створення настільних застосунків. Для цього Python пропонує кілька бібліотек, які дозволяють реалізувати графічний інтерфейс користувача. Такі бібліотеки надають готові елементи управління, зокрема кнопки, поля введення, таблиці та календарі, що є необхідними компонентами інформаційно-довідкових систем. Завдяки цьому розробник може зосередитися на логіці роботи програми, а не на низькорівневих деталях взаємодії з операційною системою.

Однією з найбільш поширених бібліотек для створення графічних інтерфейсів у Python є PyQt. Вона надає широкий набір інструментів для розробки повноцінних настільних застосунків із сучасним інтерфейсом. PyQt

дозволяє створювати багатовіконні програми, реалізовувати складну логіку взаємодії між елементами інтерфейсу та працювати з подіями користувача. Це робить її зручним вибором для розробки інформаційно-довідкових систем, орієнтованих на щоденне використання працівниками організацій.

Важливою перевагою PyQt є можливість чіткого розділення логіки програми та інтерфейсу користувача. Такий підхід позитивно впливає на підтримуваність коду та спрощує внесення змін у майбутньому. Для інформаційно-довідкових систем, які можуть розвиватися та доповнюватися новим функціоналом, це має особливе значення. Крім того, PyQt підтримує роботу з таблицями та календарями, що є ключовими елементами багатьох довідкових систем.[22]

Окрім настільних застосунків, Python широко використовується для створення веб-орієнтованих інформаційно-довідкових систем. У цьому випадку важливу роль відіграють веб-фреймворки, які забезпечують обробку HTTP-запитів, взаємодію з базами даних та формування відповідей для клієнта. Серед таких фреймворків особливе місце займають Flask та FastAPI, які відрізняються простотою використання та гнучкістю.

Flask є легким веб-фреймворком, який надає базові можливості для створення веб-застосунків. Його головною перевагою є мінімалізм і відсутність жорстких обмежень щодо структури проєкту. Це дозволяє розробнику самостійно визначати архітектуру системи, що є зручним при створенні інформаційно-довідкових систем з нестандартною логікою. Flask часто використовується для реалізації серверної частини, яка відповідає за обробку запитів та надання довідкової інформації.

FastAPI є сучасним веб-фреймворком, орієнтованим на створення швидких і надійних API. Він активно використовується для побудови клієнт-серверних інформаційно-довідкових систем, де обмін даними між компонентами відбувається у форматі JSON. FastAPI забезпечує автоматичну перевірку даних, що надходять від клієнта, та спрощує реалізацію механізмів

взаємодії між різними частинами системи. Це підвищує стабільність роботи інформаційно-довідкових систем та зменшує кількість помилок.

Невід'ємною складовою будь-якої інформаційно-довідкової системи є база даних. Python підтримує роботу з різними системами управління базами даних, що дозволяє обирати оптимальне рішення залежно від масштабів і вимог проєкту. Для локальних та невеликих інформаційно-довідкових систем часто використовується SQLite, яка не потребує складного налаштування та дозволяє зберігати дані у вигляді одного файлу. Такий підхід є зручним для настільних застосунків та прототипів.

Для більших систем Python надає інструменти для роботи з серверними базами даних, такими як PostgreSQL або MySQL. Взаємодія з базами даних здійснюється за допомогою спеціалізованих бібліотек, які забезпечують виконання запитів, обробку результатів та управління транзакціями. Це дозволяє створювати інформаційно-довідкові системи, здатні працювати з великими обсягами даних та підтримувати одночасний доступ багатьох користувачів.

Важливу роль у розробці інформаційно-довідкових систем відіграють бібліотеки для роботи з форматами даних. Python надає зручні засоби для обробки текстової інформації, файлів, а також обміну даними між компонентами системи. Формат JSON є одним з найпоширеніших у клієнт-серверних архітектурах, і Python має вбудовані інструменти для роботи з ним. Це спрощує реалізацію взаємодії між сервером та клієнтом у межах інформаційно-довідкових систем.

Окрім цього, Python має бібліотеки для роботи з датами та часом, що є важливим для систем, які оперують подіями, графіками або календарями. Такі можливості дозволяють коректно обробляти часові інтервали, виконувати сортування та фільтрацію даних за датами, а також відображати інформацію у зручному для користувача вигляді.[27]

Ще однією важливою перевагою Python є можливість інтеграції різних бібліотек у межах одного проєкту. Це дозволяє поєднувати настільний інтерфейс користувача з серверною частиною або зовнішніми сервісами. Такий підхід є актуальним для інформаційно-довідкових систем, які потребують синхронізації даних або доступу до віддалених ресурсів.

Під час розробки інформаційно-довідкових систем також важливими є інструменти для тестування та налагодження програмного коду. Python має вбудовані засоби для перевірки коректності роботи програм, що дозволяє виявляти помилки на ранніх етапах розробки. Це підвищує якість програмного продукту та зменшує ризик виникнення збоїв під час експлуатації системи.

Важливу роль у розробці клієнт-серверних інформаційно-довідкових систем відіграє серверна частина, яка відповідає за обробку запитів, зберігання даних, реалізацію бізнес-логіки та забезпечення безпеки доступу до інформації. У межах мови програмування Python одним з найбільш потужних і поширених фреймворків для створення серверних застосунків є Django. Саме цей фреймворк був використаний для реалізації серверної частини інформаційно-довідкової системи Optima.

Django є високорівневим веб-фреймворком, який надає розробнику готову структуру для створення повноцінних веб-застосунків і серверних систем. Основною ідеєю Django є принцип швидкої розробки та повторного використання компонентів. Фреймворк орієнтований на створення надійних і масштабованих систем, що особливо важливо для інформаційно-довідкових систем, які можуть з часом розширюватися та обслуговувати велику кількість користувачів.

Однією з ключових особливостей Django є чітко визначена архітектура, побудована за принципом Model–View–Template. Такий підхід дозволяє розділити логіку роботи з даними, обробку запитів і представлення інформації. Для інформаційно-довідкових систем це має суттєве значення, оскільки сприяє підтримуваності коду та спрощує внесення змін у майбутньому. Завдяки такій

структурі серверна частина системи залишається зрозумілою навіть при збільшенні обсягу функціоналу.

Django надає потужні засоби для роботи з базами даних. Вбудований механізм об’єктно-реляційного відображення дозволяє працювати з даними у вигляді об’єктів мови Python без необхідності написання складних SQL-запитів. Це значно спрощує реалізацію логіки доступу до даних та зменшує ймовірність помилок. Для інформаційно-довідкових систем, які оперують структурованими довідковими даними, такий підхід є зручним і ефективним.[30]

Ще однією важливою перевагою Django є вбудована система керування користувачами та правами доступу. Інформаційно-довідкові системи часто передбачають різні рівні доступу до даних, наприклад, для адміністратора, звичайного користувача або керівника. Django дозволяє реалізувати такі механізми без використання сторонніх бібліотек, що підвищує безпеку системи та спрощує її налаштування.

Django також забезпечує засоби для захисту серверної частини від типових загроз, зокрема від несанкціонованого доступу та помилкових запитів. Наявність вбудованих механізмів безпеки є важливим фактором при розробці інформаційно-довідкових систем, які працюють з персональними або службовими даними. Це дозволяє зменшити ризики витоку інформації та забезпечити стабільну роботу системи.

Використання Django у серверній частині інформаційно-довідкової системи дозволяє легко реалізувати обмін даними між сервером та клієнтською частиною. Сервер може надавати доступ до довідкової інформації у вигляді структурованих даних, що спрощує інтеграцію з настільними або веб-клієнтами. Такий підхід є особливо актуальним для систем, у яких клієнтська частина реалізована окремо від сервера.

Ще однією важливою особливістю Django є можливість швидкого масштабування серверної частини. У разі збільшення кількості користувачів або обсягу даних серверна частина може бути адаптована без суттєвої переробки

коду. Це робить Django придатним для використання у середніх та великих інформаційно-довідкових системах.

Фреймворк Django є ефективним інструментом для створення серверної частини інформаційно-довідкових систем. Він поєднує у собі зручність розробки, високу надійність та можливість подальшого розвитку системи. Використання Django у проєкті Optima дозволило реалізувати стабільну серверну архітектуру, забезпечити роботу з базою даних та організувати взаємодію між клієнтською і серверною частинами системи.

Таблиця 2.3 – Порівняльна характеристика фреймворків Python для розробки інформаційно-довідкових систем

Характеристика	PyQt	Flask	FastAPI	Django
Тип застосування	Настільні	Веб	API	Повноцінні серверні системи
Основне призначення	GUI	Веб-логіка	REST API	Сервер + бізнес-логіка
Архітектура	Подійна	Довільна	API-орієнтована	Model–View–Template
Робота з БД	Локальна	Через бібліотеки	Через бібліотеки	Вбудована ORM
Безпека	Обмежена	Базова	Базова	Вбудовані механізми
Масштабованість	Середня	Висока	Дуже висока	Висока
Доцільність для ІДС	Висока	Висока	Висока	Дуже висока

Таблиця 2.4 – Роль бібліотек та фреймворків Python у системі Optima

Засіб	Частина системи	Призначення

PyQt	Клієнтська	Графічний інтерфейс користувача
Django	Серверна	Обробка запитів, логіка, безпека
SQLite	Локальна	Зберігання довідкових даних
PostgreSQL / MySQL	Серверна	Централізована база даних
JSON	Обмін даними	Передача інформації між клієнтом і сервером
datetime	Обробка часу	Робота з датами та подіями

У межах даної дипломної роботи розглянуті фреймворки та бібліотеки використовуються для реалізації інформаційно-довідкової системи Optima. Обрані інструменти дозволяють забезпечити зручний інтерфейс користувача, надійну роботу з базами даних та можливість подальшого розвитку системи, що підтверджує доцільність використання мови Python для розв’язання поставлених задач.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ OPTIMA

3.1. Аналіз предметної області та постановка задачі

Інформаційно-довідкові системи в організаціях використовуються для того, щоб впорядковувати дані та швидко надавати потрібну інформацію користувачам. У багатьох установах значна частина робочих процесів пов’язана з обліком подій, що стосуються працівників: відпустки, лікарняні, відрядження, тимчасова відсутність посадових осіб, призначення відповідальних осіб на період відсутності тощо. Якщо ці дані ведуться вручну або у вигляді несистемних таблиць, виникають типові проблеми: інформація дублюється, важко знайти актуальні записи, немає єдиного формату подій, складно контролювати перетини дат, а також зростає ризик помилок під час підготовки довідок і звітів.

Предметною областю даної роботи є процеси зберігання та надання довідкової інформації про кадрові події й організаційні відсутності працівників у межах установи. Особливість предметної області полягає в тому, що дані мають часову прив’язку (дата початку й дата завершення), можуть бути різних типів (відпустка, лікарняний, відрядження тощо), та в окремих випадках супроводжуються додатковими умовами (наприклад, призначення виконуючого обов’язки на період відсутності керівника). Для користувачів важливим є не лише збереження цих даних, а й їх наочне представлення, зручний пошук та швидке отримання довідкових відомостей у потрібному форматі.

У межах даної дипломної роботи розглядається та проєктується інформаційно-довідкова система Optima, призначена для ведення та перегляду подій, пов’язаних із працівниками, із використанням календарного подання. Система Optima орієнтована на практичні потреби організації: вона має

забезпечити швидкий доступ до даних про кадрові події, спростити пошук інформації, зменшити ручну роботу та підвищити актуальність довідкових відомостей.

У типовій організації інформація про відсутності працівників та інші кадрові події часто ведеться в паперовому вигляді, у наборі окремих електронних документів або в таблицях без єдиних правил заповнення. Такий підхід має низку недоліків. По-перше, дані можуть зберігатися в різних місцях, що ускладнює їх актуалізацію та перевірку. По-друге, відсутня єдина структура подій, тому одна й та сама подія може бути описана різними словами або форматами, що ускладнює пошук. По-третє, ручне ведення даних підвищує ризик помилок у датах, прізвищах, типах подій, а також призводить до втрати часу при формуванні довідок або узгодженні інформації між підрозділами.

Окремо слід зазначити проблему оперативності. Часто довідкові дані потрібні швидко: керівництву – для контролю ситуації, кадровій службі – для ведення документації, іншим працівникам – для планування робочих процесів. Якщо доступ до інформації ускладнений, це призводить до затримок, додаткових звернень та перевантаження відповідальних осіб.

Таким чином, актуальною задачею є створення інформаційно-довідкової системи, яка забезпечить централізований облік подій, їх зручне представлення у вигляді календаря, швидкий пошук і фільтрацію, а також коректне відображення типів подій і пов’язаних умов (наприклад, заміна посадової особи на період відсутності).

Система Optima проєктується як інструмент для накопичення та надання довідкових відомостей про події, які мають часові межі та прив’язку до працівників. У контексті інформаційно-довідкових систем її головним призначенням є забезпечення таких можливостей:

1. Ведення єдиного довідкового простору подій;
2. Уніфікація даних (стандартизовані типи подій та правила збереження);

3. Наочне відображення подій у календарі;

4. Швидкий пошук і перегляд інформації за датами, працівниками та типами подій;

5. Зменшення кількості помилок та дублювання інформації.

Optima повинна підтримувати сценарії, характерні для кадрової та організаційної діяльності: фіксація відпусток, відряджень, лікарняних, а також спеціальних подій, пов’язаних із відсутністю керівної особи та призначенням виконуючого обов’язки на цей період. Особливо важливою є вимога коректного відображення таких подій у календарному вигляді, оскільки календарне представлення найбільш наочно демонструє часові перетини, тривалість і послідовність подій.

У межах предметної області можна виділити кілька типових ролей користувачів, які мають різні потреби та рівні доступу до даних.

Кадровий працівник або відповідальна особа – користувач, який вносить і редагує інформацію про події. Для нього важливі зручність введення даних, наявність списків типів подій, швидкий вибір працівника, контроль коректності дат, а також можливість швидко перевірити, що подія відобразиться правильно.

Керівник або відповідальна посадова особа – користувач, який переважно переглядає інформацію для контролю ситуації. Для нього важливі наочність, швидкий доступ до календаря, фільтрація та можливість швидко знайти потрібну подію за датами.

Адміністратор системи – користувач, який налаштовує доступ, підтримує довідники (наприклад, типи подій), контролює стабільність роботи та цілісність даних. Для нього важливі механізми авторизації, розмежування прав та керування структурою даних.

Таким чином, система Optima повинна враховувати різні сценарії використання та забезпечити баланс між простотою інтерфейсу і достатнім рівнем функціональності.

Для коректного проєктування системи необхідно визначити основні об’єкти, з якими вона працює, та зв’язки між ними. У контексті інформаційно-довідкової системи Optima ключовими об’єктами є працівник, подія, тип події та додаткові атрибути, що уточнюють зміст події.

Працівник у системі розглядається як носій довідкової інформації: ПІБ, посада, підрозділ та інші дані, які потрібні для ідентифікації. У межах системи важливо забезпечити однозначність ідентифікації працівника та коректне відображення його прізвища в календарі.

Подія є основним елементом довідкової інформації. Подія має часові межі (початок і завершення), тип, прив’язку до конкретного працівника, а також можливий опис (коментар) або додаткові поля, що залежать від конкретного типу події.

Тип події є довідником, який забезпечує стандартизацію та уніфікацію даних. Саме тип визначає, як подія буде трактуватися, відображатися у календарі та оброблятися при фільтрації. Стандартизований довідник типів подій дозволяє уникнути ситуацій, коли одна й та сама подія вводиться у різних формулюваннях.

Для окремих ситуацій може виникати потреба у пов’язаних подіях або у додатковому відображенні контексту. Наприклад, якщо відсутня керівна особа, і на цей період призначено виконуючого обов’язки, у календарі користувачу важливо бачити не лише факт відсутності, а й інформацію про того, хто заміщує. Це вимагає або спеціальної структури подій, або механізму пов’язаності, який дозволить відобразити в одному часовому інтервалі дві логічно пов’язані сутності.

У предметній області можна виділити кілька базових процесів, що повинні підтримуватися системою Optima.

Перший процес – реєстрація події. Він включає вибір працівника, визначення типу події, введення дати початку та завершення, а за потреби –

опису. Важливо, щоб система мінімізувала помилки шляхом перевірок, наприклад, щоб дата завершення не була меншою за дату початку.

Другий процес – перегляд інформації. Він передбачає відображення подій у календарному форматі та можливість переходу між місяцями або тижнями. Користувач повинен мати можливість швидко визначити, хто і коли відсутній, а також переглянути деталі події.

Третій процес – пошук та фільтрація. На практиці користувачам часто потрібно знайти події певного працівника або події певного типу за вибраний період. Тому система повинна підтримувати пошук за основними параметрами та надавати результат у зрозумілому вигляді.

Четвертий процес – формування довідкової інформації для подальшого використання. У багатьох установах потрібні списки подій за період, дані для звітності або експорту. Навіть якщо система є інформаційно-довідковою, можливість формування структурованого результату є важливим доповненням, яке підвищує її практичну цінність.

На основі аналізу предметної області формулюється задача: необхідно спроєктувати інформаційно-довідкову систему, яка забезпечить централізований облік подій, пов’язаних із працівниками, та надання довідкової інформації у зручному вигляді. Результатом має стати система Optima, яка підтримує роботу з подіями, типами подій і календарним представленням.

Функціональні вимоги до системи можна сформулювати як обов’язкові можливості, які забезпечують виконання основних задач предметної області. Система повинна забезпечити роботу з довідником типів подій, підтримувати додавання подій із часовими межами, надавати календарне відображення та механізми пошуку. Важливою вимогою є коректне відображення назви типу події, а не лише його внутрішнього ідентифікатора, оскільки користувач сприймає подію через її назву. Також важливою є можливість відображення пов’язаних подій або додаткового інформаційного контексту у випадках заміщення посадових осіб.

Нефункціональні вимоги визначають загальні властивості системи, які впливають на якість її використання. Система повинна працювати стабільно, бути зрозумілою для користувача, забезпечувати швидке завантаження й перегляд подій, а також мати захист доступу до даних. Для організаційних систем важливою є можливість розвитку: система повинна допускати додавання нових типів подій, розширення довідників та підключення нових модулів без повної переробки програмного коду.

Враховуючи предметну область, доцільним є підхід, за якого дані зберігаються у структурованому вигляді, а клієнтська частина надає користувачу зручний інтерфейс для перегляду. Для системи Optima важливо забезпечити наочність календарного подання, оскільки воно дозволяє швидко оцінити ситуацію по датах та працівниках.

Клієнтська частина може бути реалізована як настільний застосунок з графічним інтерфейсом, що є практичним рішенням для внутрішнього використання в установі. Серверна частина, реалізована на основі веб-фреймворку Django, дозволяє централізовано зберігати дані, надавати API та забезпечувати контроль доступу. Такий поділ на клієнт і сервер відповідає реальним потребам інформаційно-довідкових систем, де важлива єдина база даних і можливість працювати з нею з різних робочих місць.

Отже, аналіз предметної області показує, що задача створення інформаційно-довідкової системи Optima є актуальною та практично значущою. У предметній області існує потреба в централізованому зберіганні кадрових та організаційних подій, уніфікації типів подій та зручному календарному представленні інформації. На основі цього сформульовано постановку задачі та визначено ключові вимоги до системи: підтримка довідників, робота з подіями та датами, наочне відображення у календарі, пошук і фільтрація, а також забезпечення надійності й можливості подальшого розвитку. Подальші пункти розділу 3 будуть присвячені проєктуванню архітектури системи Optima та проєктуванню інтерфейсу користувача.

3.2. Проєктування архітектури системи Optima

Проєктування архітектури інформаційно-довідкової системи Optima є одним із найважливіших етапів розробки, оскільки саме архітектура визначає, як система буде працювати в реальних умовах. Для довідкових систем, у яких дані використовуються різними користувачами та мають часову прив’язку, архітектура повинна забезпечити узгодженість інформації, передбачувану поведінку системи та зручний доступ до потрібних відомостей. У випадку Optima основою є події, що прив’язуються до працівників і мають інтервал дат, тип події та, за потреби, додатковий опис. Календарне подання робить систему наочною, але одночасно висуває вимоги до структури даних і логіки вибірок, щоб користувач бачив повну картину за обраний період.

Вибір клієнт-серверної архітектури для Optima є обґрунтованим, оскільки в організаційних умовах довідкова інформація має бути централізованою та однаковою для всіх. Якщо дані зберігаються локально на робочих місцях, виникає ризик розсинхронізації: різні користувачі бачать різні версії інформації, а оновлення виконуються неузгоджено. При клієнт-серверному підході сервер виступає джерелом істини, а клієнт отримує дані в момент запиту, що підтримує актуальність відомостей. Додатково спрощується реалізація безпеки, оскільки контроль доступу та перевірка коректності даних виконуються централізовано на сервері.

У проєкті Optima серверна частина проєктується на основі фреймворку Django, який забезпечує структуроване створення серверних застосунків, підтримку роботи з базами даних через ORM і готові механізми керування користувачами та правами доступу. Для довідкової системи це важливо, тому що значна частина ризиків і помилок пов’язана не з інтерфейсом, а з некоректними даними, неправильними правилами доступу або нестабільним

обміном між компонентами. Django дозволяє реалізувати ці аспекти системно та з меншими витратами часу.

Клієнтська частина Optima, орієнтована на календарне відображення, має забезпечити користувачу швидке отримання відповіді на практичні питання: хто відсутній у конкретні дні, які події заплановані на період, чи є перетини подій і хто виконує обов'язки у разі відсутності відповідальної особи. Для цього клієнт повинен мати стабільний канал отримання даних і логіку, яка перетворює отримані записи в елементи календаря. Водночас клієнт не повинен брати на себе функції, які належать серверу, зокрема забезпечення цілісності даних, виконання ключових перевірок і застосування прав доступу.

Щоб система працювала передбачувано, необхідно чітко визначити розподіл відповідальності. У межах архітектурного проєктування Optima доцільно зафіксувати такі ролі частин системи (перелік короткий, саме як узагальнення):

1. Клієнт відповідає за відображення, навігацію, зручність і взаємодію з користувачем;
2. Сервер відповідає за дані, перевірки, правила, доступ і формування відповідей api;
3. База даних є місцем централізованого збереження довідкової інформації.

Обмін даними доцільно організувати через API з передачею даних у форматі JSON. На практиці це означає, що клієнт відправляє запити на отримання подій або довідників, сервер обробляє запит, застосовує правила доступу та повертає результат у єдиній структурі. При такому підході важливо забезпечити стабільність формату відповідей, щоб клієнтський застосунок працював коректно і не потребував постійних змін через дрібні корекції API.

Для Optima принципово важливо, щоб клієнт отримував не лише числові ідентифікатори довідників, а й зрозумілі назви та значення. Наприклад, тип події в базі може зберігатися як id, але користувачу потрібна назва ("Відпустка",

“Відрядження”, “Лікарняний”). Аналогічно, подія прив’язується до працівника через ідентифікатор, але календар має відображати ПІБ або прізвище. Тому сервер повинен або повертати вкладені об’єкти типу “працівник” і “тип події”, або формувати на виході поля, які можна безпосередньо показати в інтерфейсі. Такий підхід не тільки спрощує клієнт, але й зменшує кількість додаткових запитів, що підвищує швидкодію системи.

Під час проєктування API необхідно враховувати, що основний сценарій роботи користувача пов’язаний з календарним переглядом подій. З практичної точки зору найважливішим є запит “події за період”, який формує календар на обраний місяць або діапазон дат. Для оптимальної роботи цей запит повинен повертати компактний набір даних, достатній для відображення у календарі. Деталі події доцільно отримувати окремо, наприклад при відкритті конкретного запису. Це дає баланс між швидкістю завантаження календаря та доступністю повної інформації.

В архітектурі Optima суттєве значення має довідник типів подій, оскільки саме він забезпечує єдині правила опису даних. Якщо типи подій вводяться довільним текстом, виникає неоднозначність і погіршується якість пошуку. Тому тип події проєктується як посилання на довідник, а не як вільне поле. Це спрощує фільтрацію, підтримує стандартизацію та дозволяє розширювати перелік типів без переробки системи. У межах довідкової системи це особливо важливо, оскільки актуальність і правильність назви типу безпосередньо впливають на сприйняття інформації користувачем.

Щоб структура даних відповідала предметній області, у системі виділяються базові сутності: працівник, тип події та подія. Працівник виступає носієм довідкової інформації, тип події – елементом стандартизації, подія – основним записом, який має часові межі та відображається у календарі. На цьому етапі проєктування важливо визначити обов’язкові поля та зв’язки між сутностями, щоб забезпечити коректність збереження та швидкість типових вибірок. Мінімальні вимоги до структури даних можна сформулювати так:

подія не може існувати без працівника, тип події має існувати у довіднику, а інтервал дат повинен бути логічно коректним.

Окремим моментом, характерним для Optima, є підтримка ситуацій, коли події мають пов’язаний контекст. Прикладом є відсутність керівної особи та призначення виконуючого обов’язки на цей період. Для користувача важливо бачити ці дані разом, а для системи важливо зберігати їх так, щоб вони були узгодженими. На архітектурному рівні це означає, що потрібно передбачити механізм зв’язку або представлення заміщення, який дає можливість коректно відображати обидві частини інформації у календарі. Практично доцільним є варіант, коли заміщення зберігається як окрема пов’язана подія або окрема сутність, що посилається на основну подію відсутності. Це дозволяє будувати наочне відображення (наприклад, дві смуги в один період) і зберігати логічну цілісність даних.

Щоб система залишалася надійною, у проєктуванні закладаються правила валідації. Частина перевірок може виконуватися на клієнті для зручності користувача, однак остаточний контроль має виконуватися на сервері. Це гарантує, що навіть у випадку помилкових дій або нестандартних запитів база даних не отримає некоректні записи. Основні правила валідації пов’язані з коректністю дат, обов’язковістю довідникових посилань та узгодженістю періодів для пов’язаних подій. Якщо організація вимагає суворішого контролю, можуть додаватися перевірки на конфлікти подій, однак такі правила залежать від конкретної політики використання.

З точки зору підтримуваності важливо, щоб клієнтська частина була структурована таким чином, щоб логіка обміну з сервером не змішувалася з логікою відображення. Тому доцільно виділяти окремий модуль, який відповідає за роботу з API, обробку відповідей і помилок, а інтерфейс використовує вже підготовлені дані. Це дозволяє змінювати API або правила обробки, не переписуючи весь інтерфейс, і навпаки.

Наприкінці частини 1 доцільно узагальнити архітектурне рішення Optima у вигляді короткого підсумку (перелік невеликий, без перевантаження):

1. Optima проєктується як клієнт-серверна інформаційно-довідкова система з централізованим збереженням даних;
2. Серверна частина на django реалізує роботу з бд, бізнес-логіку, валідацію та контроль доступу;
3. Клієнтська частина забезпечує календарне відображення подій і зручну взаємодію з користувачем;
4. Обмін даними організовано через api у форматі json із передачею користувачу зрозумілих значень (назви типів, піб тощо);
5. Структура даних побудована навколо сутностей працівника, типу події та події, з можливістю представлення пов’язаних сценаріїв, таких як заміщення.

Серверна частина Optima, реалізована на Django, повинна бути організована так, щоб дані та логіка були структурованими й зрозумілими. Django задає типову модель організації серверного застосунку, де дані описуються у вигляді моделей, правила обробки зосереджені в серверних компонентах, а API виступає інтерфейсом взаємодії з клієнтом. Для інформаційно-довідкової системи важливо, щоб структура серверного коду не перетворювалася на “моноліт”, де все змішано, оскільки це ускладнює підтримку та збільшує ризик помилок при внесенні змін.

У серверній частині Optima доцільно підтримувати модульний принцип. У дипломному проєктуванні це можна описати як логічне групування функціоналу за призначенням: окремо робота з працівниками, окремо робота з типами подій, окремо робота з подіями та окремо механізми доступу. Такий підхід не лише робить систему зрозумілішою, а й дозволяє поступово розширювати її. Наприклад, додавання нового довідника або нового типу подій не повинно вимагати переробки всіх модулів; достатньо внести зміни у відповідну частину та узгодити формат даних в API.

Для фіксації логіки серверної організації (у помірному переліку) можна виділити ключові елементи, які повинні бути передбачені в серверній структурі:

1. Моделі даних (працівники, типи подій, події, механізм пов’язаних сценаріїв);
2. Компоненти доступу до даних і вибірок (orm-запити, фільтрації);
3. Механізми валідації (перевірки дат, обов’язкових полів, узгодженості зв’язків);
4. Арі-рівень (прийом параметрів, формування json-відповідей);
5. Правила доступу (перевірка прав залежно від ролі користувача);
6. Журналювання та обробка помилок.

У практичній роботі довідкової системи важливим є питання узгодженості форматів. Якщо клієнт відображає календар, він має отримувати дані у вигляді, який легко перетворити у візуальні елементи. Саме тому серверна частина повинна забезпечувати стабільний формат: подія має містити дати, працівника, тип події та короткий текст для відображення. Якщо клієнту передаються лише ідентифікатори, він змушений робити додаткові запити або зберігати локальні мапінги, що ускладнює логіку і підвищує ризик помилок. У контексті Optima це особливо важливо, оскільки реальна проблема таких систем часто полягає в тому, що тип події приходить як числове значення, а відображатися має як текстова назва. Тому архітектурно правильним є підхід, за якого сервер повертає ідентифікатор і назву типу одночасно, або формує у відповіді окреме поле з готовим значенням для відображення.

Оскільки в системі використовується календарне подання, серверна логіка вибірки подій за період повинна враховувати інтервальний характер події. Подія не завжди “вкладається” у межі одного місяця: вона може початися раніше і закінчитися пізніше. Тому критерій відбору подій має спиратися на перетин інтервалів. Це є важливою деталлю проєктування, бо саме від неї залежить, чи побачить користувач у календарі всі актуальні події. У реальному використанні користувач очікує, що при перегляді місяця будуть показані всі події, які хоча б

частково тривають у цьому місяці, навіть якщо початок або завершення припадає на сусідній період.

Забезпечення цілісності даних в Optima є завданням серверної частини. Для інформаційно-довідкових систем характерно, що помилки введення або “роз’їхавші” довідники швидко накопичуються і роблять дані непридатними. Саме тому правила валідації повинні бути чітко визначені і застосовуватися на сервері для кожної операції, яка змінює дані. Доречно виділити основні перевірки, без яких система не може бути надійною (перелік невеликий, але показує контроль якості даних):

1. Коректність інтервалу дат (початок не пізніше завершення);
2. Наявність обов’язкових посилань (працівник і тип події);
3. Існування довідникових значень (тип події не може бути “невідомим”);
4. Узгодженість пов’язаних сценаріїв (наприклад, заміщення повинно відповідати періоду відсутності);
5. Коректність форматів і значень, що надходять через api (відсікання некоректних даних на вході).

Паралельно з валідацією важливо продумати обробку конфліктів подій. У деяких організаціях допускається накладення кількох подій, у деяких – ні. Тому архітектура повинна дозволяти або забороняти конфлікти залежно від вимог. Якщо конфлікти не допускаються, сервер може перевіряти, чи не існує вже подій, які перетинаються за датами для того самого працівника і є взаємовиключними. Якщо ж конфлікти допустимі, система може лише попереджати або відображати такі ситуації у календарі.

Не менш важливою частиною серверної архітектури є безпека. У системі Optima має бути реалізована автентифікація (перевірка користувача) та авторизація (перевірка прав). Важливо, щоб права контролювалися саме на сервері, а не лише через елементи інтерфейсу. Навіть якщо клієнт “не показує” кнопку видалення, це не гарантує безпеки, якщо сервер приймає такий запит

без перевірки прав. Django дає можливість організувати систему ролей і прав доступу, що є стандартним підходом для організаційних інформаційних систем.

Для опису підходу до доступу доречно навести типові ролі користувачів, які найчастіше потрібні в таких системах (перелік короткий, “чуть більше” як ти просив):

1. Користувач із доступом лише на перегляд довідкової інформації;
2. Оператор/відповідальна особа, яка може створювати та редагувати події;
3. Адміністратор, який керує довідниками, користувачами та правами.

Питання стабільності в реальних умовах також пов’язане з обробкою помилок і журналюванням. Для клієнт-серверної системи типовими є ситуації, коли сервер тимчасово недоступний, мережеве підключення нестабільне або дані в запиті містять помилки. Архітектура Optima повинна передбачити, що система не “падає” при таких ситуаціях, а коректно повідомляє користувача та зберігає можливість продовжити роботу. На клієнті це означає зрозумілі повідомлення без технічних подробиць, а на сервері – фіксацію події в логах, щоб можна було з’ясувати причину.

Для забезпечення керованості системи корисно визначити, які саме події є важливими для журналювання. У практичному плані до таких подій належать: помилки API, помилки доступу, спроби виконати заборонені операції, створення або видалення подій, а також критичні помилки сервера. Таке журналювання дозволяє підтримувати систему, аналізувати інциденти та підвищувати надійність.

У клієнтській частині Optima архітектура також повинна бути структурованою, щоб інтерфейс не перетворився на складну і нестабільну систему. Клієнт має бути орієнтований на відображення і взаємодію, тому обмін із сервером доцільно винести в окремий компонент. Це зменшує складність інтерфейсу та робить код зрозумілішим. Клієнт повинен мати механізм кешування довідників (наприклад, типів подій) на час сесії, щоб не робити

повторні запити без потреби, але при цьому “джерело істини” має залишатися на сервері. Для довідкових систем це важливий баланс: з одного боку, швидкодія інтерфейсу, з іншого – актуальність даних.

Логіка відображення календаря потребує чіткої прив’язки даних до інтерфейсу. Події повинні перетворюватися на мітки/смуги в календарі, і цей процес має бути передбачуваним. Для цього клієнту необхідно знати, який текст показати та як візуально відрізняти типи подій. Найчастіше це вирішується через правила відображення залежно від типу (наприклад, колір або стиль). У випадку Optima важливо, щоб правила відображення не залежали від “жорстко зашитих” значень на клієнті, а спиралися на довідник типів, що надходить із сервера. Це зменшує ризик помилок і спрощує зміну довідника.

Оскільки Optima може розвиватися, важливо закласти архітектурну готовність до розширення. Розширення може проявлятися у додаванні нових типів подій, створенні додаткових модулів, підключенні нового інтерфейсу або реалізації експорту. Клієнт-серверний підхід дає можливість розвивати систему поступово, не змінюючи її основу: сервер додає нові можливості через API, а клієнт оновлюється за потреби.

Таблиця 3.1 – Характеристика клієнтської частини інформаційно-довідкової системи Optima (PyQt6, модулі, UI-компоненти, інтеграція з API)

Компонент	Як зроблено в клієнті Optima	Для чого в системі
Стек / платформа	Python + PyQt6 (GUI), requests (HTTP), openpyxl (Excel)	Десктоп-клієнт з формами, календарем і обміном з сервером
Точка входу	app.py → створює MainWindow, відкриває AuthWindow	Запуск програми та старт з авторизації
Архітектура UI	MainWindow + QStackedWidget (екрани: Calendar/Main/Settings)	Перемикання сторінок без відкриття купи вікон
Авторизація	gui/auth_window.py: логін через POST /api/v1/token/, токен в data/auth.json	Доступ до захищених ендпоінтів сервера

		(events/logs)
Налаштування IP	IP сервера зберігається в data/settings.json (можна змінити у вікні входу ⚙)	Щоб клієнт підключався до потрібного сервера/порта
API-клієнт	core/client.py: build_url("http://{ip}/api/v1/..."), заголовок Authorization: Bearer ... + X-API-KEY	Уніфікований спосіб робити GET/POST/PUT/DELETE до сервера
Офлайн/резерв	Якщо сервер недоступний – MainWindow підхоплює події з локальної БД ../events.db	Щоб програма не була “мертва” без сервера
Локальна БД	core/db.py: SQLite-таблиця events (ПІБ полями, type, start/end, comment, vo)	Кеш/резерв подій для локального перегляду
Календар	gui/calendar_page.py + CustomCalendar	Основний екран: показ подій по днях + список подій
Відображення “ВО”	У календарі: якщо Чепелюк і є vo – показує жовту кнопку/рядок “В.о.”	Твоя фішка “синя смуга відсутності + жовта смуга ВО”
Типи подій	Частково: MainPage.load_event_types() бере з GET /event-types/ в type_combo; але CalendarPage ще має локальний EVENT_TYPE_NAMES={1..3}	Щоб у UI були нормальні назви типів (і тут є ризик розсинхрону)
Експорт	calendar_page.py використовує openpyxl для експорту в Excel	Вивантаження подій у таблицю (зручно для звітів)
BugReport	gui/bug_report_dialog.py: відправка POST /api/v1/bugreport/ з описом + файлом	Збір помилок/скрінів прямо з клієнта

Можна виділити основні висновки щодо деталізації архітектури:

1. Серверна частина на Django має модульну організацію, що спрощує розвиток і супровід;

2. API повинно повертати дані в узгодженому форматі, включаючи зрозумілі назви типів подій та дані працівника;
3. Валідація даних і контроль логічної цілісності реалізуються на сервері як обов’язкова умова надійності;
4. Автентифікація та авторизація забезпечують контроль доступу до даних відповідно до ролі користувача;
5. Журналювання та коректна обробка помилок підвищують стабільність системи в реальних умовах;
6. Клієнтська частина має бути відокремлена від мережевої логіки та орієнтована на календарне відображення;
7. Архітектура повинна допускати розвиток системи без повної переробки основних компонентів.

У межах системи Optima серверна частина реалізована як веб-застосунок на Django з використанням Django REST Framework. Такий підхід дозволяє побудувати централізований рівень доступу до даних, у якому виконуються ключові перевірки, керування доступом, формування довідкових відповідей для клієнтів і збереження інформації в базі даних. Важливою особливістю фактичної реалізації є те, що сервер вже надає готовий API-шар із версією /api/v1/, що спрощує узгодження між клієнтською частиною та сервером і дозволяє розвивати інтерфейси незалежно від внутрішньої структури бази.

Серверний проєкт має стандартну структуру Django: конфігураційний модуль проєкту (api_project) та прикладний модуль (додаток) api, який містить моделі, представлення (views), маршрути (urls) і допоміжні компоненти. Для реалізації API використано DRF, а для автентифікації – SimpleJWT. Така зв’язка є типовою для клієнт-серверних інформаційних систем, де клієнт звертається до сервера через HTTP-запити та отримує дані у форматі JSON.

Фактичний склад основних технологічних компонентів серверної частини можна стисло зафіксувати так (щоб це було “видно” в дипломі):

1. Django як базовий веб-фреймворк та каркас проєкту;

2. Django REST Framework для реалізації API та роботи з HTTP-запитами;

3. rest_framework_simplejwt для JWT-автентифікації;

4. django-cors-headers для підтримки CORS (дозвіл запитів з інших джерел);

5. python-dotenv для завантаження параметрів із .env;

6. SQLite як база даних у поточній конфігурації (dev/локальний режим);

7. підтримка MEDIA_ROOT/MEDIA_URL для збереження файлів (звіти про помилки).

У поточній реалізації сервер оперує кількома ключовими сутностями, які відповідають задачам інформаційно-довідкової системи.

По-перше, використовується довідник типів подій EventType (поле name). Він забезпечує стандартизацію назв типів і дозволяє клієнту відображати тип події в “людському” вигляді.

По-друге, використовується модель подій Event, яка зберігає дані про конкретний часовий інтервал і особу. Подія включає ПІБ у розбитому вигляді (прізвище, ім’я, по батькові), тип події, дати початку/завершення, коментар, а також поле vo, яке може використовуватися для відображення виконуючого обов’язки (що напряму пов’язано з вимогами предметної області).

По-третє, реалізовано журнал дій UserActionLog, який фіксує ключові операції користувачів із подіями (створення, оновлення, видалення). Це важливий елемент для службових систем, оскільки дозволяє відстежувати зміни та підвищує керованість.

Окремо реалізовано модель BugReport для прийому повідомлень про помилки з прикріпленим файлом. Це можна розглядати як елемент підтримки та експлуатаційної надійності системи.

У проєкті налаштовано базову маршрутизацію з префіксом /api/v1/, що є зручним для версіонування. Реалізовані ендпоїнти покривають як роботу з подіями, так і сервісні функції (реєстрація, логування, звіти).

Основні ендпоїнти серверної частини Optima:

1. /api/v1/events/ – універсальний ендпоїнт для роботи з подіями (GET/POST/PUT/DELETE) з вимогою автентифікації;
2. /api/v1/event-types/ – отримання довідника типів подій (id, name), доступне без автентифікації;
3. /api/v1/token/ – отримання JWT-токенів (із кастомним case-insensitive логіном);
4. /api/token/refresh/ – оновлення JWT-токена;
5. /api/v1/register/ – реєстрація користувача (перевірка пароля через стандартні валідатори Django);
6. /api/v1/logs/ – перегляд журналу дій (доступ обмежено: потрібна автентифікація та права staff);
7. /api/v1/bugreport/ – прийом звіту про помилку з описом та файлом (доступ без автентифікації).

Таке розділення відповідає логіці інформаційно-довідкової системи: основні довідкові дані (події) захищені авторизацією, а довідник типів може бути доступний ширше для коректного відображення на клієнті. Наявність окремого ендпоїнта для логів дозволяє реалізувати контрольовану адміністративну функцію без перевантаження основного API.

У конфігурації DRF встановлено JWT-автентифікацію як механізм за замовчуванням. Це означає, що для доступу до захищених ендпоїнтів клієнт має передавати токен у заголовку Authorization: Bearer <token>. У поточній реалізації ендпоїнт /api/v1/events/ працює лише для автентифікованих користувачів, а /api/v1/logs/ має додаткове обмеження – доступ лише для користувачів зі статусом is_staff.

Окремою особливістю є реалізація отримання токена через CaseInsensitiveTokenObtainPairView: логін користувача перевіряється без урахування регістру (case-insensitive). Це спрощує реальне використання системи, зменшуючи кількість помилок під час входу.

З погляду дипломного проєктування це можна сформулювати так: серверна частина реалізує централізований контроль доступу, де правила задаються на рівні API (через permission-класи), а клієнт виступає лише інтерфейсом, який передає токен і отримує дозволені дані.

Під час проєктування інформаційно-довідкових систем важливим є те, щоб клієнт отримував дані, придатні для відображення без “здогадок”. У твоїй реалізації це враховано у відповіді /api/v1/events/: сервер повертає не лише “сирий” тип події, а також формує type_name. Якщо поле type містить числове значення, сервер підтягує назву з довідника EventType. Якщо ж тип події переданий як текст (не число), повертається текстове значення. Такий підхід частково вирішує проблему узгодження типів і дозволяє клієнту стабільно показувати тип події як назву.

У відповідь події також включено ключові поля, які потрібні для календарного відображення та деталізації:

1. Ідентифікатор події;
2. Тип і назва типу;
3. Дати початку й завершення;
4. Піб у розбитих полях;
5. Коментар (за наявності);
6. Поле vo для відображення заміщення/виконання обов’язків.

Таким чином, серверна частина не лише зберігає дані, а й виконує роль “постачальника довідкових представлень”, підготовлених до показу в інтерфейсі.

У реалізації /api/v1/events/ передбачені базові перевірки наявності обов’язкових полів при створенні та оновленні подій. Це дозволяє відсікти

некоректні запити на вході та підтримувати мінімальну якість даних у базі. Крім того, реалізовано ведення журналу дій: при створенні, оновленні та видаленні події формується текстовий запис, який зберігається у UserActionLog. Це забезпечує просте відстеження змін і може бути використано як адміністративна функція контролю.

Таблиця 3.2 – Характеристика серверної частини інформаційно-довідкової системи Optima (Django/DRF, API, моделі)

Компонент	Реалізація у сервері Optima	Для чого в системі
Стек	Django + DRF + SimpleJWT	Серверний каркас + REST API + авторизація
API версія	/api/v1/...	Стабільність інтеграції з клієнтом, можливість розвитку
База даних	SQLite (db.sqlite3)	Зберігання довідкових даних (події/типи/логи) у dev-режимі
Ключові моделі	Event, EventType, UserActionLog, BugReport	Події, довідник типів, аудит, звіти про помилки
Події (API)	/api/v1/events/ (GET/POST/PUT/DELETE), доступ: IsAuthenticated	Головний ресурс системи: отримання та керування подіями
Типи подій (API)	/api/v1/event-types/, доступ: AllowAny	Довідник типів для коректного відображення назв на клієнті
Авторизація	/api/v1/token/ (логін без регістру), /api/token/refresh/	Вхід у систему та оновлення токенів
Реєстрація	/api/v1/register/ + validate_password	Створення користувачів із перевіркою пароля
Підготовка даних	В GET /events/ формується type_name і дати YYYY-MM-DD, поле vo	Щоб клієнт показував календар правильно (назви типів + заміщення)
Логи дій	UserActionLog + /api/v1/logs/ (тільки	Контроль хто/що міняв у подіях

	is_staff)	
BugReport	/api/v1/bugreport/ (AllowAny) + файл у media/	Збір помилок з вкладеннями для підтримки системи
CORS + .env	CORS_ALLOW_ALL_ORIGINS=True, SECRET_KEY/API_KEY з .env	Інтеграція з клієнтами та безпечніша конфігурація

З практичної точки зору та з погляду дипломного опису важливо зафіксувати, що в системі реалізовані три експлуатаційні механізми, які підвищують надійність:

1. Контроль доступу до критичних операцій через jwt і permission-перевірки;
2. Перевірка обов’язкових полів для запобігання “порожнім” записам;
3. Журналювання змін, що дає можливість аналізувати історію операцій.

Окремим доповненням до експлуатаційної надійності є ендпоїнт bugreport, який приймає опис проблеми та файл. Файли зберігаються у медіасховищі (MEDIA_ROOT), що дозволяє накопичувати матеріали для аналізу помилок та покращення системи без втручання в основні дані подій.

У серверній частині використовується підхід із параметризацією через .env (зокрема SECRET_KEY і API_KEY). Це відповідає практиці відокремлення конфігурації від коду. Також увімкнено CORS-доступ для всіх джерел і розширено список дозволених заголовків (у тому числі x-api-key). У рамках дипломного проєктування це можна пояснити як підготовку до роботи з різними клієнтами (настільний, веб-клієнт) та спрощення інтеграції під час розробки і тестування.

Оскільки зараз як база використовується SQLite, система зручна для локального розгортання та демонстрації. Водночас сама архітектура Django дозволяє без зміни прикладної логіки перейти на серверну СУБД (наприклад,

PostgreSQL) у разі потреби масштабування та багатокористувацького режиму роботи.

3.3. Проєктування інтерфейсу користувача інформаційно-довідкової системи Optima

Інтерфейс користувача в інформаційно-довідковій системі відіграє ключову роль, оскільки саме через нього користувач отримує доступ до довідкових даних та виконує типові дії: перегляд подій, пошук інформації, додавання або редагування записів. Для системи Optima інтерфейс має забезпечити зручне календарне подання подій і максимально швидке отримання відповіді на практичні питання: хто відсутній у певний період, який тип події, які дати охоплює подія, та чи призначено виконуючого обов’язки на час відсутності посадової особи.

Під час проєктування інтерфейсу враховуються особливості предметної області: події мають часові межі, події повинні бути стандартизовані за типами, а відображення має бути наочним і зрозумілим без зайвих переходів між екранами. Тому базовим елементом UI обрано календар, який дозволяє користувачеві швидко орієнтуватися у датах і бачити події у прив’язці до конкретних днів.

Інтерфейс Optima проєктується так, щоб підтримувати щоденне використання без складного навчання. У цьому контексті важливими є такі цілі:

1. Наочність: події відображаються безпосередньо на календарі, щоб користувач бачив ситуацію за період;
2. Простота дій: основні операції виконуються через кілька кроків;
3. Однозначність даних: у ui показуються “людські” назви типів подій, а не внутрішні ідентифікатори;

4. Зменшення помилок: при введенні даних застосовуються перевірки (обов’язкові поля, коректність дат);

5. Стабільна робота: користувач отримує зрозумілі повідомлення у разі помилок мережі чи сервера;

6. Підтримка специфічних сценаріїв: окреме коректне відображення заміщення (во) під час відсутності керівної особи.

Проєктування інтерфейсу спирається також на загальні принципи ергономіки: логічна структура екранів, узгоджені назви полів, однакове розташування елементів керування, мінімальна кількість дій для отримання інформації, та зрозумілий зворотний зв’язок після натискання кнопок або виконання операції.

Інтерфейс клієнтської частини Optima реалізовано як настільний застосунок з графічним інтерфейсом, у якому навігація між основними функціональними екранами організована логічно та послідовно. Користувач починає роботу з екрана авторизації, після чого переходить до основного вікна із календарним відображенням подій та додатковими розділами.

Узагальнена структура UI включає такі основні елементи:

1. Вікно авторизації (вхід у систему, налаштування параметрів підключення);

2. Головне вікно (контейнер інтерфейсу та перемикання сторінок);

3. Сторінка календаря (перегляд подій, навігація по датах, перегляд деталей);

4. Форма створення/редагування події (внесення або корекція довідкових даних);

5. Сторінка налаштувань (параметри системи та підключення);

6. Діалог звіту про помилку (відправка опису проблеми та файлу на сервер).

Такий поділ дозволяє уникнути перевантаження одного екрана великою кількістю елементів і забезпечує зрозумілу логіку: користувач завжди знаходить потрібну дію в очікуваному місці.

Екран авторизації призначений для введення облікових даних та отримання доступу до захищених ресурсів сервера. Основне завдання цього екрана – виконати вхід у систему та зберегти токен доступу, необхідний для подальших запитів до API.

На екрані авторизації доцільно передбачати:

1. Поля введення логіна і пароля;
2. Кнопку входу;
3. Елемент доступу до налаштувань підключення (наприклад, ір/адреса сервера);
4. Повідомлення про помилки (неправильний логін/пароль, сервер недоступний).

У випадку помилки інтерфейс повинен давати коротке і зрозуміле повідомлення, яке пояснює проблему без технічних деталей. Після успішної авторизації користувач переходить до головного вікна, де доступні основні довідкові функції.

Календарна сторінка є центральною частиною інтерфейсу Optima, оскільки саме тут користувач проводить більшість часу. Календар повинен забезпечити швидку орієнтацію у датах і наочне представлення подій.

Для календарної сторінки ключовими є такі функції:

1. Перегляд подій за обраний місяць (або інший період);
2. Перемикання між періодами (вперед/назад);
3. Перегляд подій конкретного дня (після вибору дати);
4. Відкриття деталей події (піб, тип, дати, коментар);
5. Запуск створення/редагування події (якщо дозволено правами);
6. Фільтрація або швидкий пошук (за потреби реалізації).

Події в календарі мають відображатися коротко: користувачу важливо швидко “зчитати” інформацію, не відкриваючи кожну подію окремо. Тому подія в межах календаря подається як короткий маркер або смуга з прізвищем (або скороченим ПІБ), а тип події та деталі доступні при перегляді.

Окремою вимогою для Optima є підтримка ситуації, коли у період відсутності керівника призначається виконуючий обов’язки. Для користувача важливо бачити це одразу на календарі. Тому у візуальному представленні закладається принцип подвійного відображення:

1. Основна смуга відсутності працівника відображається базовим стилем (наприклад, синім);
2. Якщо подія має значення «во», під основною смугою додатково відображається другий маркер (наприклад, жовтим) із прізвищем особи, що заміщає.

Такий підхід підвищує наочність і зменшує кількість уточнювальних дій: користувач бачить не лише “хто відсутній”, але й “хто відповідальний”.

Оскільки Optima працює з довідковими даними, внесення подій повинно бути максимально контрольованим. Форма створення/редагування події проєктується як окремий екран або діалогове вікно, де користувач вводить дані та підтверджує збереження.

Обов’язковими елементами форми є:

1. Дані працівника (ПІБ або вибір зі списку, залежно від реалізації);
2. Тип події (вибір із довідника типів);
3. Дата початку та дата завершення;
4. Коментар (необов’язкове поле);
5. Поле для “ВО” (якщо подія передбачає заміщення).

Щоб зменшити кількість помилок, у формі застосовуються прості правила перевірки: заборона порожніх обов’язкових полів, перевірка коректності дат, а також логічна узгодженість значень (наприклад, дата завершення не раніше за

дату початку). Після успішного збереження користувач повинен отримати зрозумілий результат: або повідомлення про успіх, або оновлення календаря без додаткових дій.

У практичному використанні важливим є можливість швидко змінити параметри підключення до сервера (наприклад, IP або адресу), особливо якщо система розгортається в локальній мережі або має кілька середовищ. Тому в інтерфейсі передбачається сторінка або розділ налаштувань, де користувач може змінити параметри та зберегти їх.

Налаштування мають зберігатися локально у вигляді конфігураційного файлу, щоб після перезапуску програми параметри не потрібно було вводити повторно. Це підвищує зручність і зменшує час старту роботи з системою.

Для підвищення надійності та підтримуваності системи Optima важливо передбачити механізм збору інформації про помилки під час експлуатації. У клієнтській частині реалізовується діалог, який дозволяє користувачу описати проблему і, за потреби, прикріпити файл (наприклад, скріншот). Після відправки дані надходять на серверний ендпоїнт звіту про помилку, що створює основу для технічної підтримки та вдосконалення системи.

Користувачу при цьому має відображатися простий результат: повідомлення про успішну відправку або повідомлення про неможливість відправки (наприклад, якщо сервер недоступний).

Надійний інтерфейс повинен коректно реагувати на типові проблеми: відсутність з’єднання, помилки авторизації, некоректні дані, або внутрішні помилки сервера. Для користувача важливо, щоб програма не “зависала” і не закривалася без пояснення. Тому в інтерфейсі передбачаються:

1. Повідомлення при недоступності сервера;
2. Повідомлення при помилці входу або відсутності прав доступу;
3. Повідомлення при некоректно заповнених полях у формах;
4. Повідомлення при невдалій операції створення/оновлення/видалення.

Також важливо, щоб критичні помилки не призводили до втрати вже введених даних без попередження, а інтерфейс давав користувачу можливість повторити дію.

Таблиця 3.3 – Основні екрани клієнтського інтерфейсу Optima та їх призначення

Екран / модуль інтерфейсу	Основне призначення	Ключові елементи
Авторизація	Вхід у систему та отримання токена доступу	Логін/пароль, кнопка входу, налаштування підключення, повідомлення про помилки
Головне вікно	Контейнер інтерфейсу та навігація між розділами	Меню/кнопки переходу, область відображення сторінок
Календар	Основний перегляд довідкових подій у прив’язці до дат	Календар, список подій дня, відкриття деталей, навігація по періодах
Форма події	Створення/редагування подій і перевірка введення	ПІБ, тип події, дати, коментар, поле “ВО”, кнопки збереження/скасування
Налаштування	Зміна параметрів підключення та збереження конфігурації	Поле IP/адреси, збереження, перевірка коректності
BugReport	Відправка опису проблеми та файлу на сервер	Поле опису, вибір файлу, кнопка відправки, повідомлення результату

Інтерфейс користувача Optima спроєктовано з орієнтацією на календарне представлення довідкової інформації про події. Основною ідеєю UI є швидкий доступ до даних за датами та наочне відображення подій із мінімальною кількістю дій з боку користувача. В інтерфейсі передбачено екран авторизації, головне вікно навігації, календарну сторінку як основний робочий простір, форму створення/редагування подій, сторінку налаштувань та механізм звітування про помилки.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ OPTIMA

4.1. Реалізація функціоналу системи засобами Python

Реалізація системи Optima виконана мовою Python у вигляді клієнт-серверного рішення. Такий підхід дозволяє розділити систему на два рівні: серверний рівень, який зберігає та обробляє дані, і клієнтський рівень, який показує ці дані користувачу та дає зручні інструменти для роботи з ними. У контексті інформаційно-довідкової системи це особливо важливо: довідкові дані повинні бути централізованими, узгодженими і доступними різним користувачам, а інтерфейс – простим і зрозумілим.

Серверна частина Optima реалізована на Django з використанням Django REST Framework для створення API. Клієнтська частина реалізована як настільний застосунок на Python (графічний інтерфейс), який працює з API через HTTP-запити. Обмін даними організовано у форматі JSON, а доступ до основних ресурсів закрито JWT-автентифікацією, що дозволяє контролювати, хто може переглядати та змінювати події.

На сервері ключові дані описані у вигляді моделей Django. У твоїй реалізації це довідник типів подій, події, журнал дій користувачів і модель звітів про помилки. Наприклад, модель події Event містить ПІБ, тип події, дати та додаткові поля коментаря і vo (для відображення виконуючого обов’язки):

```
# api/models.py (фрагмент)
class Event(models.Model):
    last_name = models.CharField(max_length=100)
    first_name = models.CharField(max_length=100)
    middle_name = models.CharField(max_length=100, blank=True)
```

```
type = models.CharField(max_length=100)

start = models.DateTimeField()

end = models.DateTimeField()

comment = models.TextField(blank=True)

vo = models.CharField(max_length=100, blank=True)
```

Саме поле vo важливе для специфіки Optima: система повинна показувати не лише факт відсутності, а й того, хто заміщує (ВО). Тобто вже на рівні моделі закладено підтримку сценарію, який потім відображається у календарі.

Доступ до подій реалізовано через один ендпоїнт /api/v1/events/, який підтримує кілька HTTP-методів. У твоєму коді це зроблено однією функцією events_api з декоратором DRF, яка обробляє GET, POST, PUT, DELETE. Ключовий момент – ресурс захищено IsAuthenticated, тому без токена доступу клієнт не отримає список подій і не зможе нічого змінити.

При отриманні подій (GET) сервер не просто віддає “сирі” поля, а формує зручне представлення для клієнта. Зокрема, сервер додає поле type_name, щоб клієнт показував нормальну назву типу, а не внутрішнє значення. Це видно у фрагменті формування відповіді:

```
# api/views.py (фрагмент GET)

formatted.append({

   "id": ev.id,

   "type": int(ev.type),

        "type_name": EventType.objects.filter(id=int(ev.type)).first().name if ev.type.isdigit() else ev.type,

   "start": ev.start.strftime('%Y-%m-%d'),

   "end": ev.end.strftime('%Y-%m-%d'),

   "last_name": ev.last_name,
```

```
        "first_name": ev.first_name,
        "middle_name": ev.middle_name,
        "comment": ev.comment,
        "vo": ev.vo
    })
```

Це практично важлива річ саме для довідкової системи: клієнт отримує вже “підготовлені” дані для UI. Дати повертаються у стабільному форматі YYYY-MM-DD, що зручно для календаря.

Створення та оновлення подій реалізовано через POST і PUT. Перед створенням/оновленням є проста серверна перевірка наявності обов’язкових полів (щоб у базі не з’являлися “биті” записи). Це реалізовано через допоміжну функцію:

```
# api/views.py (фрагмент)
def get_required_fields(data, fields):
    for field in fields:
        if field not in data:
                return JsonResponse({'error': f"Поле '{field}' є обов'язковим"}, status=400)
    return None
```

Після успішних змін сервер створює запис у журналі дій (UserActionLog). Тобто система не лише зберігає довідкові дані, а й фіксує “хто і що зробив”, що підсилює контроль і прозорість у службовій системі:

```
# api/views.py (фрагмент)
def create_action_log(user, text):
    UserActionLog.objects.create(user=user, action=text)
```

Доступ до логів винесено в окремий ендпоїнт /api/v1/logs/, який додатково обмежений: потрібна авторизація і права is_staff. Це вже схоже на реальну адмін-функцію, а не просто “для галочки”.

Окремо у тебе реалізовано механізм “підтримки користувача” – прийом звітів про помилки /api/v1/bugreport/. Сервер приймає description і файл (multipart), перевіряє, що опис не пустий, і зберігає дані у BugReport. Це корисно в реальній експлуатації: замість “ой воно зламалось” є опис і доказ (скрін/лог/файл).

```
# api/views.py (фрагмент)

@api_view(['POST'])

@permission_classes([AllowAny])

def bug_report_view(request):

    description = request.POST.get('description', '').strip()

    file = request.FILES.get('file')

    if not description:

            return JsonResponse({'error': 'Опис помилки обов’язковий.'}, status=400)

    report = BugReport.objects.create(description=description, file=file)

        return JsonResponse({'message': 'Звіт збережено', 'id': report.id}, status=201)
```

Також реалізовано довідник типів подій /api/v1/event-types/, який віддає список id і name. Це дозволяє клієнту підтягувати список типів і показувати його користувачу у випадаючому списку або фільтрі.

Для входу клієнт звертається до /api/v1/token/ і отримує пару токенів (access/refresh). У тебе використано кастомний

CaseInsensitiveTokenObtainPairView, тобто логін можна вводити без урахування регістру.

Приклад клієнтського запиту:

```
import requests

base = "http://192.168.0.1:800"

resp = requests.post(f"{base}/api/v1/token/", json={

   "username": "Admin",

   "password": "password123"

})

tokens = resp.json()

access = tokens["access"]
```

Після цього доступ до /api/v1/events/ виконується з заголовком Authorization: Bearer <token>:

```
headers = {"Authorization": f"Bearer {access}"}

events = requests.get(f"{base}/api/v1/events/", headers=headers).json()
```

На клієнті основна задача – зручно показати довідкові дані та дати інструменти для типових дій: переглянути календар, додати/оновити/видалити подію, переглянути деталі, експортувати інформацію, а також відправити bugreport. Через те, що сервер віддає дані вже у “читабельному” форматі (наприклад type_name і дати YYYY-MM-DD), клієнту не потрібно здогадуватися, як перетворити id на назву – він просто бере поле і показує.

Додавання події з клієнта під твій POST /api/v1/events/ виглядає так :

```
payload = {

   "last_name": "Іваненко",

   "first_name": "Іван",
```

```
    "middle_name": "Іванович",
    "type": "1
    "start": "2025-12-01",
    "end": "2025-12-10",
    "comment": "Відпустка за графіком",
    "vo": "Петренко"
}
r = requests.post(f"{base}/api/v1/events/", json=payload, headers=headers)
print(r.status_code, r.json())
```

Оновлення (PUT) вимагає id і той самий набір обов’язкових полів:

```
payload_update = dict(payload)
payload_update["id"] = 15
payload_update["comment"] = "Коментар оновлено"
requests.put(f"{base}/api/v1/events/", json=payload_update, headers=headers)
```

Bugreport з файлом (під твій POST /api/v1/bugreport/) робиться multipart-запитом:

```
files = {"file": open("screenshot.png", "rb")}
data = {"description": "При створенні події з VO інколи не відображається смужка в календарі."}
requests.post(f"{base}/api/v1/bugreport/", data=data, files=files)
```

У клієнтській частині також реалізуються “побутові” функції, які реально потрібні користувачу в установі: наприклад, експорт подій в Excel (це часто просять для звітів/погоджень). У Python це зазвичай роблять через openpyxl, формуючи таблицю з колонками ПІБ/тип/дати/коментар/ВО.

Python у цій системі реально зіграв роль “універсального клею”, бо одна й та сама мова використана і для серверної логіки, і для клієнта. У результаті простіше тримати єдиний стиль даних, швидше робити правки, легше налагоджувати (debug) і простіше розширювати функції. Django дав структурований сервер (моделі, API, доступ), а клієнт на Python – швидку розробку інтерфейсу і інтеграції (HTTP-запити, робота з файлами, експорт).

Optima реалізована засобами Python як клієнт-серверна інформаційно-довідкова система. Сервер на Django/DRF забезпечує зберігання подій та довідників, захищений доступ через JWT, підготовку даних для зручного відображення (зокрема type_name і формати дат), журналювання дій користувачів та прийом повідомлень про помилки з файлами. Клієнтський застосунок, взаємодіючи з API через HTTP-запити, надає користувачу інтерфейс для перегляду календаря, роботи з подіями, підтримки сценарію BO та виконання прикладних дій на кшталт експорту і bugreport. Така реалізація відповідає призначенню інформаційно-довідкової системи: швидко та наочно надавати актуальні дані і забезпечувати контрольоване внесення змін.

4.2. Інтеграція клієнтської та серверної частин

Інтеграція клієнтської та серверної частин системи Optima реалізована за принципом клієнт-серверної взаємодії через HTTP-запити. Серверна частина на Django/DRF надає REST API з префіксом /api/v1/, а клієнтська частина (настільний застосунок) використовує API для отримання довідкових даних і виконання операцій з подіями. Передача даних здійснюється у форматі JSON, а доступ до основних ресурсів (події, журнали) захищений JWT-автентифікацією.

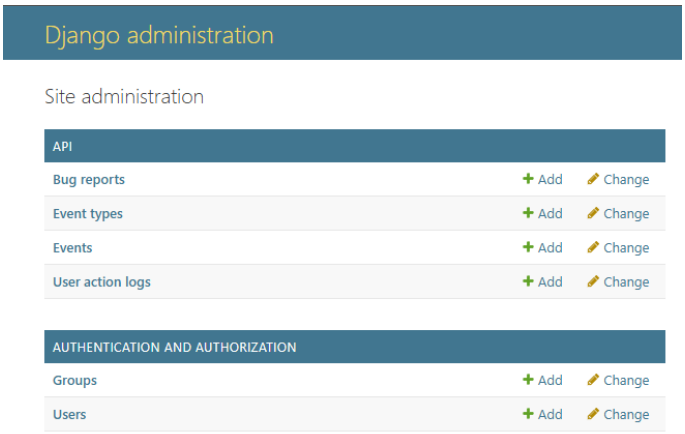

Рисунок 4.1 – Структура API системи Optima

Для коректної роботи інтеграції клієнту необхідно знати адресу сервера (IP/host і порт). Практично це вирішується через параметр підключення, який користувач може змінити у налаштуваннях або у вікні входу. Такий підхід дозволяє запускати клієнт у різних середовищах (локально, у мережі установи, тестовий сервер) без перекомпіляції чи зміни коду.

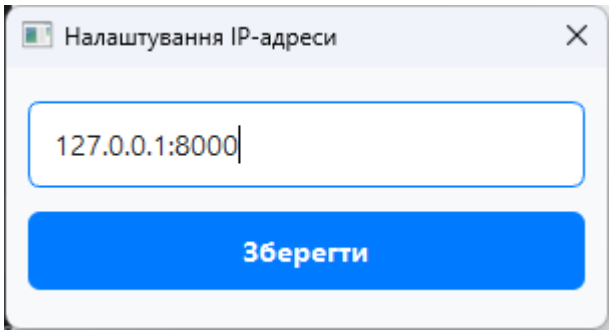

Рисунок 4.2 – Вікно налаштування підключення (IP/host сервера) у клієнті.

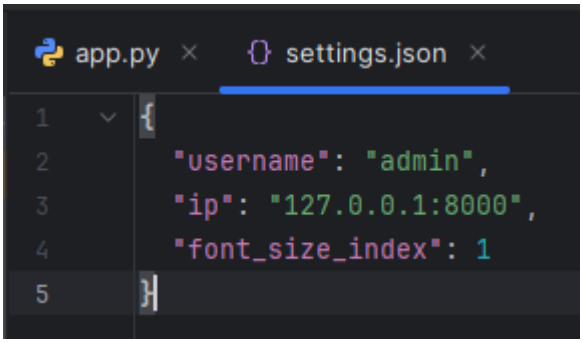

Рисунок 4.3 – Приклад файлу налаштувань клієнта з адресою сервера.

Інтеграція починається з автентифікації: клієнт відправляє запит на ендпоїнт отримання токена (/api/v1/token/) із логіном і паролем. Сервер

перевіряє дані і повертає JWT-токени (access/refresh). Після цього клієнт використовує access-токен для доступу до захищених ресурсів, передаючи його в заголовку Authorization.

Приклад запиту (логіка клієнта):

```
import requests
base_url = "http://192.168.0.4:8090"
resp = requests.post(f"{base_url}/api/v1/token/", json={
    "username": "Admin",
    "password": "password123"
})
tokens = resp.json()
access_token = tokens["access"]
```

Далі токен додається в заголовки кожного запиту до захищених ендпоїнтів:

```
headers = {"Authorization": f"Bearer {access_token}"}
events = requests.get(f"{base_url}/api/v1/events/", headers=headers).json()
```

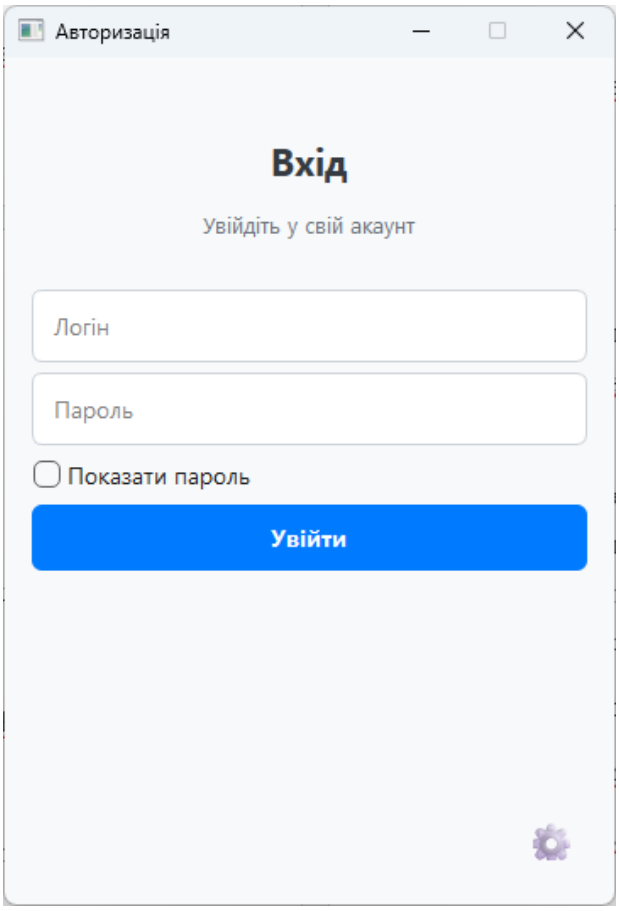

Рисунок 4.4 – Вікно входу в клієнті

```
[14/Dec/2025 18:45:02] "GET /admin/api/ HTTP/1.1" 200 7226
[14/Dec/2025 18:45:03] "GET /admin/api/bugreport/ HTTP/1.1" 200 11476
[14/Dec/2025 18:45:04] "GET /admin/ HTTP/1.1" 200 12674
[14/Dec/2025 18:46:56] "GET /api/v1/events/ HTTP/1.1" 200 20655
[14/Dec/2025 18:46:56] "GET /api/v1/event-types/ HTTP/1.1" 200 140
[14/Dec/2025 18:49:34] "POST /api/v1/token/ HTTP/1.1" 200 483
[14/Dec/2025 18:49:34] "GET /api/v1/events/ HTTP/1.1" 200 20655
[14/Dec/2025 18:49:38] "GET /admin/ HTTP/1.1" 200 12674
```

Рисунок 4.5 – Успішна відповідь сервера

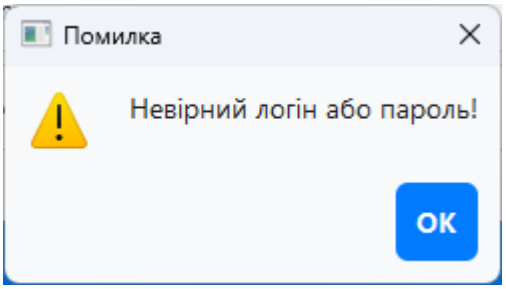

Рисунок 4.6 – Приклад помилки авторизації (неправильний пароль) і повідомлення в клієнті.

Для коректного відображення в інтерфейсі клієнт підвантажує довідник типів подій через GET /api/v1/event-types/. Це дає змогу показувати користувачу не “id типу”, а нормальні назви типів подій у формах і в календарі. Доступ до цього ендпоїнта в реалізації відкритий (без токена), що спрощує ініціалізацію інтерфейсу.

Після авторизації клієнт виконує GET /api/v1/events/ і отримує список подій. Сервер повертає дані в “готовому для UI” вигляді: дати у форматі YYYY-MM-DD, а також поле type_name, яке містить зрозумілу назву типу. Це спрощує клієнт: йому не потрібно самостійно перетворювати ідентифікатори в назви, достатньо відобразити type_name.

Приклад отримання:

```
headers = {"Authorization": f"Bearer {access_token}"}

events = requests.get(f"{base_url}/api/v1/events/", headers=headers).json()
```

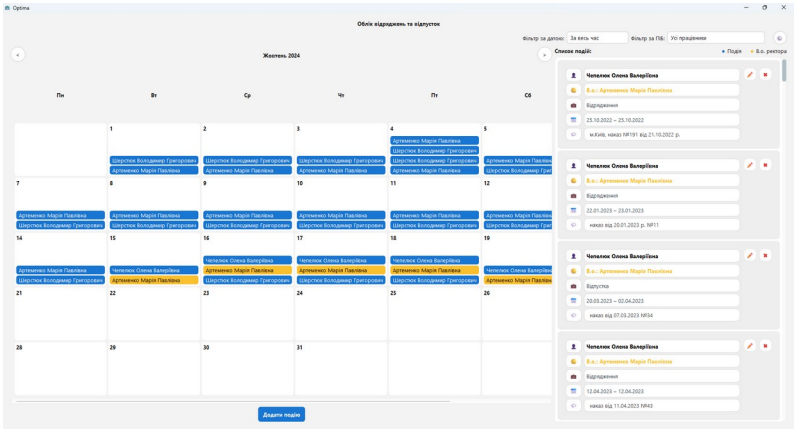

Рисунок 4.7 – Головний екран клієнта

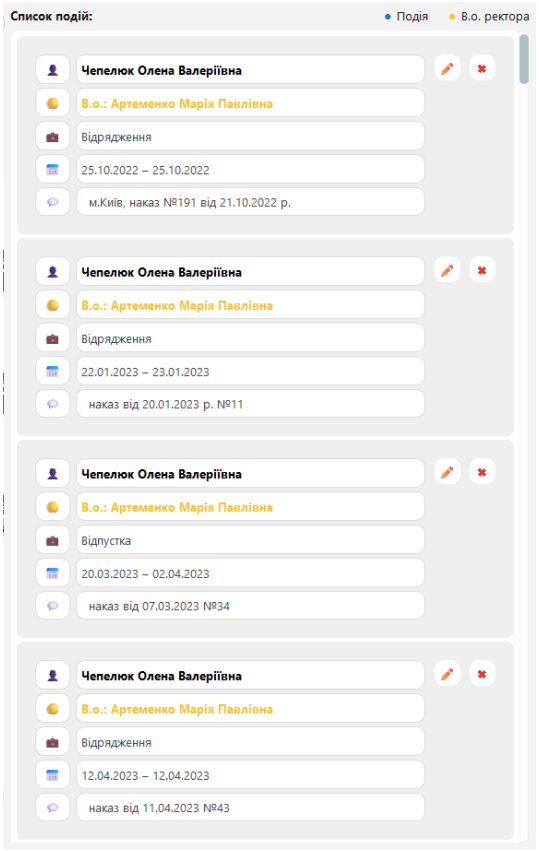

Рисунок 4.8 – Список подій обраного дня

Для додавання нової події клієнт формує JSON-дані (ПІБ, тип, дати, коментар, vo) і надсилає POST /api/v1/events/. Сервер виконує перевірку обов’язкових полів і, у разі успіху, створює запис у базі. Після цього клієнт оновлює календар (повторно запитує події або локально додає подію до списку).

Приклад:

```
payload = {
    "last_name": "Іваненко",
    "first_name": "Іван",
    "middle_name": "Іванович",
    "type": "1",
    "start": "2025-12-01",
    "end": "2025-12-10",
```

```
    "comment": "Відпустка",
    "vo": "Петренко"
}
r = requests.post(f"{base_url}/api/v1/events/", json=payload, headers=headers)
```

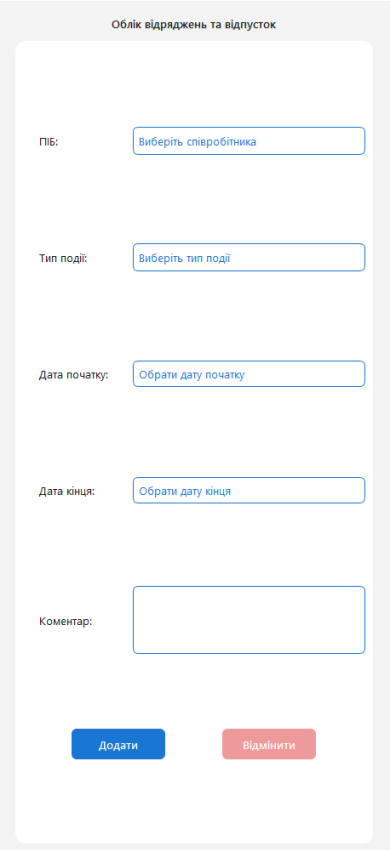

Рисунок 4.9 – Форма створення події у клієнті

Редагування події реалізується через PUT /api/v1/events/ із передачею id події та обов’язкових полів. Видалення – через DELETE із параметром id. Після виконання операції клієнт також оновлює відображення календаря, щоб користувач одразу бачив актуальні дані.

Специфіка Optima полягає в тому, що подія може містити поле vo – виконуючого обов’язки. Сервер передає це поле в JSON-відповіді, а клієнт використовує його для візуального відображення додаткового маркера під основною подією відсутності. Завдяки цьому інтеграція “дані → UI” забезпечує

не просто показ факту відсутності, а й показ пов’язаного контексту відповідальності.

Після створення, оновлення або видалення подій сервер формує запис у журналі дій. Доступ до журналу обмежено користувачами з правами is_staff через /api/v1/logs/. Це забезпечує адміністративний контроль і дозволяє перевіряти факт змін. З позиції інтеграції це означає, що клієнт (або адмін-інструмент) може окремо отримувати журнал дій і використовувати його для аналізу.

Для підтримки користувачів і збору помилок реалізовано інтеграцію через /api/v1/bugreport/, куди клієнт надсилає опис проблеми та файл. Це спрощує супровід, бо розробник отримує матеріали одразу в системі, а не “десь у месенджері”.

У реальному використанні інтеграція повинна адекватно реагувати на типові проблеми: сервер недоступний, токен протух/невалідний, некоректні дані запиту, помилки прав доступу. У таких випадках клієнт має показувати користувачу коротке повідомлення (наприклад, “Сервер недоступний” або “Потрібен вхід у систему”), а технічні деталі можуть зберігатися в логах для налагодження.

Таким чином, інтеграція в Optima реалізована як взаємодія клієнта з сервером через REST API. Клієнт отримує адресу сервера з налаштувань, виконує авторизацію через JWT-токени, підвантажує довідники та події, а також виконує операції створення, оновлення та видалення подій через стандартні HTTP-методи. Сервер повертає дані у стабільному форматі, включаючи підготовлені для відображення поля (наприклад, type_name), що спрощує клієнтський інтерфейс. Додатково інтеграція охоплює адміністративні механізми контролю (журнал дій) і механізми підтримки (BugReport), що підвищує придатність системи до експлуатації.

4.3. Тестування та аналіз ефективності роботи системи

Тестування системи Optima виконувалося з метою перевірки коректності реалізованих функцій, стабільності інтеграції клієнтської та серверної частин, а також оцінки того, наскільки зручно і швидко система дає користувачу потрібну довідкову інформацію. Оскільки Optima є інформаційно-довідковою системою, основний акцент робився не на складних обчисленнях, а на правильності даних, надійності операцій із подіями, передбачуваності роботи інтерфейсу та якості відображення календарної інформації. Додатково перевірялася специфічна вимога предметної області – показ виконуючого обов'язки (ВО) для періодів відсутності відповідальної особи, оскільки саме цей сценарій найбільше впливає на практичну корисність календаря.

Тестування проводилося у змішаному форматі. Серверну частину перевіряли через прямі HTTP-запити до API, щоб переконатися у правильності статус-кодів, структури JSON-відповідей, наявності обов'язкових полів і коректній реакції на помилки. Клієнтську частину перевіряли через реальні сценарії роботи користувача: вхід у систему, завантаження календаря, перегляд подій у вибрані дні, додавання/редагування/видалення подій та контроль того, що внесені зміни одразу відображаються в інтерфейсі. Такий підхід дозволив оцінити систему як єдине рішення, де важливо не лише "що повертає сервер", але й "як це виглядає і працює для користувача".

Окремим етапом було тестування механізму автентифікації та контролю доступу. Перевірялося, що запити до захищених ресурсів (наприклад, робота з подіями) недоступні без токена, і стають доступними після успішного отримання JWT-токена на ендпоїнті авторизації. Також перевірялися типові помилкові ситуації: неправильний пароль, відсутній або невалідний токен, спроба виконати дію без прав доступу. Результати показали, що сервер коректно обмежує доступ до основних функцій і повертає відповідні відповіді (помилки доступу або відмови у виконанні запиту). Для системи внутрішнього

використання це важливо, оскільки гарантує, що довідкові дані не можуть змінюватися довільно та залишаються під контролем авторизованих користувачів.

Після цього тестувалась коректність довідника типів подій і відповідність типів у подіях. Перевірка показала, що сервер повертає список типів подій у стандартизованому вигляді, а у відповідях подій додатково формується назва типу у полі type_name. Це суттєво підвищує якість інтеграції, оскільки клієнту не потрібно самостійно зіставляти ідентифікатор типу з назвою, що зменшує кількість помилок відображення. У практичному сенсі це означає, що користувач бачить в інтерфейсі чіткі й однакові назви (“Відпустка”, “Відрядження”, “Лікарняний” тощо), а не технічні значення, які не мають сенсу без довідника.

Наступним кроком перевірялися основні операції роботи з подіями: створення, оновлення та видалення. На сервері тестувалося, що при створенні події з коректними даними запис з’являється у базі та повертається у списку подій, а при оновленні змінюються саме ті поля, які редагувалися користувачем. Окремо перевірялася реакція на некоректні запити, зокрема відсутність обов’язкових полів. У цих випадках сервер повинен відмовляти у виконанні операції та повертати повідомлення про помилку. Така поведінка є критичною для інформаційно-довідкових систем, тому що саме сервер відповідає за цілісність даних і не повинен дозволяти появу “порожніх” або логічно неправильних записів. На клієнті тестувалося, що після створення або редагування події календар оновлюється, а користувач одразу бачить результат своєї дії без необхідності перезапуску програми або ручного “оновлення” списку.

У рамках календарного відображення додатково перевірялися ситуації, які найчастіше створюють проблеми: події з різними тривалостями та події, що перетинають межі періоду перегляду. Для календаря важливо, щоб подія відображалась у потрібні дні згідно інтервалу, а користувач міг швидко зрозуміти, які дати охоплює відсутність. Також перевірялося, що деталізація

події (ПІБ, тип, дати, коментар) доступна з інтерфейсу і не вимагає складних дій. У результаті було підтверджено, що календарне представлення є зручним для сприйняття і відповідає завданню довідкової системи – швидко показувати стан “хто і коли відсутній” у наочному форматі.

Окреме місце в тестуванні займав специфічний сценарій ВО. Перевірялося, що сервер коректно зберігає та повертає поле vo у подіях, де воно заповнене, а клієнт використовує ці дані для відображення додаткової інформації у календарі та в деталях події. Практична цінність цього механізму полягає в тому, що система показує не лише сам факт відсутності, а й пов’язаний контекст відповідальності. У щоденній роботі це зменшує кількість уточнень і запитань між працівниками, оскільки відповідальна особа в період відсутності визначена і відображена одразу.

Для оцінки керованості системи було протестовано журналювання дій користувача. Перевірялося, що після створення, оновлення та видалення подій на сервері з’являються записи у журналі, а доступ до перегляду журналу має лише користувач із відповідними правами. Наявність журналу дій є важливою для організаційного застосування: вона дає можливість перевіряти історію змін, знаходити причини неправильних даних та підвищує дисципліну роботи із системою. У технічному сенсі це також спрощує супровід, оскільки при виникненні проблеми можна відновити послідовність дій, які до неї призвели.

Також перевірявся механізм збору повідомлень про помилки (bugreport). Тестування показало, що сервер приймає опис проблеми і файл, зберігає їх і повертає підтвердження. Для реальної експлуатації це підвищує ефективність підтримки: замість усного опису “щось не працює” з’являється структуроване повідомлення з матеріалами, які можна використати для аналізу та виправлення. Такий підхід підвищує загальну надійність системи, оскільки дозволяє швидше реагувати на інциденти та накопичувати інформацію про типові помилки.

Аналіз ефективності роботи Optima проводився з точки зору практичних результатів для користувача. Календарний формат подання забезпечує швидке

отримання довідкової інформації: користувач одразу бачить ситуацію за вибраний період, а не шукає дані у текстових списках або документах. Використання довідника типів і формування type_name зменшує неоднозначність і помилки у трактуванні подій, що особливо важливо для кадрових даних. Централізоване збереження на сервері та контроль доступу через JWT роблять систему більш надійною, оскільки дані захищені і змінюються контрольовано. Журналювання дій підвищує прозорість і дозволяє аналізувати зміни, а механізм bugreport зменшує час на пошук причин проблем у разі їх виникнення. У сукупності ці особливості показують, що система не лише виконує заявлені функції, але й є придатною до експлуатації в умовах організації, де важливі стабільність, контрольованість та швидкий доступ до довідкових даних.

Таким чином, тестування підтвердило коректність реалізації основних сценаріїв роботи Optima на рівні API, інтеграції та користувацького інтерфейсу. Система стабільно виконує операції з подіями, забезпечує однозначне відображення типів, підтримує специфічний сценарій ВО, надає механізми контролю змін через журналювання та забезпечує зручний канал для збору повідомлень про помилки. Проведений аналіз показав, що обране технічне рішення є ефективним для задач інформаційно-довідкової системи, оскільки дозволяє швидко отримувати актуальні дані у наочному вигляді та підтримувати їхню якість і цілісність у процесі використання.

ВИСНОВКИ

У результаті виконання дипломної роботи було реалізовано та досліджено підхід до розробки інформаційно-довідкової системи на мові Python на прикладі системи Optima. Метою роботи було показати, що Python може бути повноцінною основою для створення прикладних систем, які надають користувачам швидкий і наочний доступ до довідкових даних, підтримують централізоване збереження інформації та забезпечують контроль доступу і цілісність даних. Поставлені завдання загалом виконано: сформовано вимоги до системи, спроєктовано архітектуру, реалізовано серверну і клієнтську частини, організовано інтеграцію між ними та проведено тестування, яке підтвердило працездатність ключових сценаріїв.

Під час розробки було проаналізовано та випробувано різні підходи і інструменти, після чого обрано ті, що найкраще відповідають завданням інформаційно-довідкової системи. Для серверної частини було розглянуто варіанти реалізації веб-рівня на Python, і як найбільш придатний інструмент обрано Django у поєднанні з Django REST Framework, оскільки це рішення забезпечує структурований підхід до розробки, зручну роботу з даними через ORM, можливість швидко створити й підтримувати REST API, а також має готові механізми безпеки та керування користувачами. Для авторизації обрано JWT-механізм, тому що він є зручним для клієнт-серверної моделі й дозволяє надійно обмежити доступ до ресурсів системи. Для клієнтської частини було обрано підхід настільного застосунку на Python, орієнтованого на календарне подання даних, що є практичним для користувача: інформація сприймається швидше, а типові питання (“хто відсутній”, “на які дати”, “хто виконує обов’язки”) вирішуються без зайвих переходів по меню. Додатково враховано прикладні потреби експлуатації: журналювання дій користувачів на сервері та механізм надсилання bugreport із файлом, що підвищує контрольованість системи та спрощує її підтримку.

У межах роботи також виконано порівняння Python з іншими мовами програмування, яке дозволило обґрунтувати вибір технологій. У порівнянні з C/C++ Python суттєво швидший у розробці прикладних систем, оскільки не потребує великої кількості низькорівневого коду та дає змогу зосередитись на бізнес-логіці, а не на деталях керування пам’яттю чи складності збірки. У порівнянні з Java Python зазвичай дає коротший і більш читабельний код, швидше дозволяє робити прототипи та вносити зміни, що важливо для систем, які розвиваються під потреби організації. У порівнянні з C# (особливо в контексті Windows-десктопу) Python може поступатися частині “коробкових” можливостей екосистеми .NET, але компенсує це простішим входом, гнучкістю та великою кількістю бібліотек для інтеграції, автоматизації та роботи з даними. У порівнянні з JavaScript/Node.js Python виграє тим, що одна й та сама мова може використовуватися і для сервера, і для настільного клієнта, а також має зрілу екосистему для швидкого створення інформаційних систем з базою даних, авторизацією та API. У підсумку Python показав себе як практичний вибір саме для задач інформаційно-довідкового класу, де важливі швидкість розробки, простота підтримки, доступність фреймворків та інтеграційні можливості.

Реалізована система Optima підтвердила переваги обраного підходу на практиці. Серверна частина забезпечує централізоване збереження подій і довідників, контроль доступу через JWT, формування даних у зручному для клієнта вигляді (зокрема передавання назв типів подій), журналювання змін та прийом повідомлень про помилки з файлами. Клієнтська частина забезпечує зручний календарний інтерфейс, який дозволяє швидко отримувати довідкову інформацію, виконувати типові операції з подіями, а також підтримує специфічний сценарій відображення виконуючого обов’язки (ВО), що підвищує інформативність і практичну корисність системи. Тестування показало, що основні сценарії роботи виконуються коректно: авторизація, отримання довідників, завантаження і відображення подій, операції створення/оновлення/видалення, контроль доступу до журналів і робота механізму bugreport. Аналіз ефективності підтвердив, що календарне

представлення скорочує час на отримання інформації, стандартизація типів подій зменшує неоднозначність, а журналювання і bugreport підвищують керованість та підтримуваність системи в експлуатації.

Загалом результати роботи дозволяють зробити висновок, що Python є доцільним і ефективним інструментом для розробки інформаційно-довідкових систем, а поєднання Django/DRF для серверної частини та Python-клієнта для інтерфейсу дає практичне, гнучке та масштабоване рішення. Подальший розвиток системи може включати розширення довідників, деталізацію ролей і прав доступу, оптимізацію вибірок при рості обсягу даних, підключення більш потужної СУБД для багатокористувацької роботи, а також розробку веб-інтерфейсу як додаткового клієнта без зміни серверної логіки.