Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ ТА ЗВІТНОСТІ В ОСВІТНЬОМУ ПРОЦЕСІ»

зі спеціальності 122 Комп’ютерні науки
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Кобель Максим Олександрович
_____ «____»_____ 2026 р.
(підпис)

Науковий керівник к.ф.-м. н., доцент, Парфьонова Тетяна Олексадрівна
_____ «____»_____ 2026 р.
(підпис)

Рецензент

ПОЛТАВА 2026 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Реалізація інформаційної системи для планування та звітності в освітньому процесі»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Кобель Максим Олександрович
Затверджена наказом ректора № 213-Н від «01» жовтня 2025 р.
Термін подання студентом роботи «___»_____ 2026 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки веб застосунків та сайтів.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Аналіз існуючих інформаційних систем управління освітнім процесом
2.2. Аналіз електронних журналів планування та обліку діяльності викладачів
2.3. Аналіз технологій розроблення веборієнтованих інформаційних систем
2.4. Формування вимог до системи
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проєктування архітектури інформаційної системи
3.2. Проєктування структури бази даних
3.3. Проєктування ролей користувачів і прав доступу
3.4. Проєктування функціональних модулів електронного журналу
3.5. Проєктування REST API інформаційної системи
3.6. Проєктування інтерфейсу користувача
4. ПРАКТИЧНА ЧАСТИНА
4.1. Обґрунтування вибору технологій розроблення
4.2. Реалізація серверної частини інформаційної системи
4.3. Реалізація клієнтської частини електронного журналу
4.4. Реалізація механізму авторизації та розмежування доступу
4.5. Реалізація модулів планування, обліку годин і формування звітності
4.6. Тестування та розгортання інформаційної системи
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 3 аркуші блок-схем, 26 ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Парфьонова Т.О.		
Інформаційний огляд	Парфьонова Т.О.		
Теоретична частина	Парфьонова Т.О.		
Практична частина	Парфьонова Т.О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2025 р.

Здобувач вищої освіти _____ Кобель Максим Олександрович
(підпис)

Науковий керівник _____ к.ф.-м.н., доц. Парфьонова Т.О.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2026 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2025 р.

Погоджено
Науковий керівник _____
к.пед.н., Тетяна ПАРФЬОНОВА
«____» _____ 2025 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Реалізація інформаційної системи для планування та звітності в освітньому процесі»
Прізвище, ім'я, по батькові Кобель Максим Олександрович

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Аналіз існуючих інформаційних систем управління освітнім процесом
2.2. Аналіз електронних журналів планування та обліку діяльності викладачів
2.3. Аналіз технологій розроблення веборієнтованих інформаційних систем
2.4. Формування вимог до системи
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проєктування архітектури інформаційної системи
3.2. Проєктування структури бази даних
3.3. Проєктування ролей користувачів і прав доступу
3.4. Проєктування функціональних модулів електронного журналу
3.5. Проєктування REST API інформаційної системи
3.6. Проєктування інтерфейсу користувача
4. ПРАКТИЧНА ЧАСТИНА
4.1. Обґрунтування вибору технологій розроблення
4.2. Реалізація серверної частини інформаційної системи
4.3. Реалізація клієнтської частини електронного журналу
4.4. Реалізація механізму авторизації та розмежування доступу
4.5. Реалізація модулів планування, обліку годин і формування звітності
4.6. Тестування та розгортання інформаційної системи
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ М.О. Кобель
(підпис)
«____» _____ 2025 р.

РЕФЕРАТ

Записка: 116 сторінок, 26 рисунків, 1 додаток, 22 літературних джерел.

ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННИЙ ЖУРНАЛ, ОСВІТНІЙ ПРОЦЕС, ІНДИВІДУАЛЬНИЙ ПЛАН, ЗВІТНІСТЬ, FASTAPI, REACT, POSTGRESQL.

Дипломна робота присвячена розробці електронного журналу планування та звітності діяльності науково-педагогічних працівників закладу вищої освіти. Актуальність роботи зумовлена необхідністю автоматизації процесів планування, обліку та контролю різних видів діяльності викладачів, підвищення ефективності управління освітнім процесом та спрощення формування звітної документації.

Метою роботи є розробка електронного журналу планування та звітності діяльності науково-педагогічних працівників закладу вищої освіти.

Об’єктом дослідження є процес автоматизації планування, обліку та контролю діяльності науково-педагогічних працівників у закладі вищої освіти.

Предметом дослідження є методи, засоби та програмні технології розробки веборієнтованих інформаційних систем планування та звітності в освітньому процесі.

У ході виконання дипломної роботи проведено аналіз існуючих інформаційних систем управління освітнім процесом та електронних журналів обліку діяльності викладачів. На основі проведеного аналізу сформовано функціональні вимоги до розроблюваної системи.

У роботі спроєктовано архітектуру інформаційної системи, структуру бази даних, функціональні модулі, ролі користувачів та REST API. Розроблено серверну та клієнтську частини електронного журналу, реалізовано механізми авторизації та розмежування доступу між викладачами, завідувачами кафедр і адміністраторами.

Для реалізації проєкту використано Python, FastAPI, SQLAlchemy, PostgreSQL, React, Docker та Docker Compose. Обраний стек технологій

забезпечив створення сучасної веборієнтованої інформаційної системи з можливістю подальшого розширення функціональних можливостей.

У системі реалізовано модулі планування діяльності, автоматичного розрахунку нормативних годин, обліку виконаних робіт та формування звітів у форматі Microsoft Excel. Передбачено можливість централізованого керування нормативами часу та автоматизацію підготовки звітної документації.

Проведено тестування розробленого програмного забезпечення та виконано його розгортання із застосуванням контейнерної технології Docker. Результати тестування підтвердили коректність роботи основних функціональних модулів та відповідність системи поставленим вимогам.

Отже, розроблена інформаційна система може бути використана як практичний інструмент для автоматизації планування та звітності діяльності науково-педагогічних працівників у закладах вищої освіти.

ЗМІСТ

ВСТУП 8
1. ПОСТАНОВКА ЗАДАЧІ 12
2. ІНФОРМАЦІЙНИЙ ОГЛЯД 14
2.1. Аналіз існуючих інформаційних систем управління освітнім процесом 14
2.2. Аналіз електронних журналів планування та обліку діяльності викладачів 21
2.3. Аналіз технологій розроблення веборієнтованих інформаційних систем 23
2.4. Формування вимог до системи 26
3. ТЕОРЕТИЧНА ЧАСТИНА 29
3.1. Проєктування архітектури інформаційної системи 29
3.2. Проєктування структури бази даних 33
3.3. Проєктування ролей користувачів і прав доступу 36
3.4. Проєктування функціональних модулів електронного журналу 39
3.5. Проєктування REST API інформаційної системи 42
3.6. Проєктування інтерфейсу користувача 46
4. ПРАКТИЧНА ЧАСТИНА 50
4.1. Обґрунтування вибору технологій розроблення 50
4.2. Реалізація серверної частини інформаційної системи 58
4.3. Реалізація клієнтської частини електронного журналу 66
4.4. Реалізація механізму авторизації та розмежування доступу 70
4.5. Реалізація модулів планування, обліку годин і формування звітності 74
4.6. Тестування та розрогртання інформаційної системи 76
ВИСНОВКИ 92
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 95
ДОДАТОК А 98

ВСТУП

Інформатизація діяльності закладів вищої освіти супроводжується активним впровадженням програмних засобів для автоматизації управлінських, навчальних та організаційних процесів. Важливе місце серед таких процесів займає планування та облік діяльності науково-педагогічних працівників, що охоплює навчальну, методичну, наукову й організаційну роботу. Використання традиційних підходів до ведення індивідуальних планів і звітної документації, які ґрунтуються на роботі з електронними таблицями та текстовими документами, ускладнює контроль виконання запланованих показників, обробку даних та формування звітів. У зв'язку з цим актуальним напрямом розвитку інформаційних технологій у сфері освіти є створення спеціалізованих веборієнтованих систем, призначених для автоматизації процесів планування, обліку та аналізу результатів діяльності викладачів.

Одним із важливих документів, що використовується для організації діяльності викладача, є індивідуальний план роботи науково-педагогічного працівника. Його ведення передбачає облік великої кількості показників, пов’язаних із виконанням навчального навантаження, наукової, методичної та організаційної діяльності. В умовах зростання обсягів інформації та необхідності оперативного контролю виконання планів виникає потреба у впровадженні спеціалізованих інформаційних систем, здатних забезпечити централізоване зберігання даних, автоматизацію розрахунків та формування звітності.

Актуальність теми зумовлена необхідністю підвищення ефективності процесів планування та звітності в освітньому процесі шляхом створення електронного журналу діяльності викладачів. Використання таких систем дає змогу автоматизувати ведення індивідуальних планів, забезпечити контроль виконання запланованих показників, організувати взаємодію між викладачами, завідувачами кафедр та адміністраторами системи, а також спростити

процедуру підготовки звітних документів. Особливої актуальності набуває забезпечення централізованого обліку діяльності викладачів із використанням веборієнтованих технологій, що надають можливість працювати із системою незалежно від місця перебування користувача та використовуваного пристрою.

Метою роботи є розробка електронного журналу планування та звітності діяльності науково-педагогічних працівників закладу вищої освіти.

Об’єктом дослідження є процес автоматизації планування, обліку та контролю діяльності науково-педагогічних працівників у закладі вищої освіти.

Предметом дослідження є методи, засоби та програмні технології розробки веборієнтованих інформаційних систем планування та звітності в освітньому процесі.

Для досягнення поставленої мети необхідно вирішити такі завдання: провести аналіз існуючих інформаційних систем управління освітнім процесом та електронних журналів аналогічного призначення; визначити функціональні та нефункціональні вимоги до інформаційної системи; обґрунтувати вибір технологій і засобів розробки; спроєктувати архітектуру програмного забезпечення та структуру бази даних; реалізувати серверну частину системи; розробити клієнтський вебінтерфейс; реалізувати механізми автентифікації та авторизації користувачів; забезпечити розмежування прав доступу відповідно до ролей користувачів; реалізувати модулі ведення індивідуальних планів, обліку виконаних робіт та формування звітності; забезпечити автоматичний розрахунок трудомісткості робіт на основі нормативів часу; реалізувати формування звітних документів у форматі Excel; провести тестування програмного забезпечення та оцінити його відповідність поставленим вимогам.

Під час виконання роботи використано методи аналізу предметної області, методи проєктування інформаційних систем, принципи побудови багаторівневих вебзастосунків, технології клієнт-серверної взаємодії, а також підходи до розробки програмного забезпечення із розподілом функціональності між серверною та клієнтською частинами. Архітектура розробленої системи

побудована за трирівневим принципом та включає рівень представлення, рівень бізнес-логіки та рівень зберігання даних.

Серверна частина системи реалізована мовою програмування Python із використанням фреймворку FastAPI, який забезпечує створення REST API, маршрутизацію запитів, валідацію даних та взаємодію між компонентами системи. Для роботи з базою даних використано ORM-бібліотеку SQLAlchemy, що забезпечує об’єктно-реляційне відображення сутностей предметної області та реалізацію операцій створення, читання, оновлення і видалення даних. Для опису схем даних і перевірки коректності вхідної інформації використано Pydantic. Автентифікація користувачів реалізована на основі технології JSON Web Token із використанням бібліотеки python-jose, а для захисту паролів застосовано алгоритм хешування bcrypt. Формування звітних документів реалізовано за допомогою бібліотеки openpyxl.

Для зберігання інформації використано систему керування базами даних PostgreSQL. Основними сутностями системи є користувачі, індивідуальні плани, записи про виконані роботи та нормативи часу. Реалізована структура даних забезпечує підтримку ролей викладача, завідувача кафедри та адміністратора, а також можливість централізованого керування нормативами часу та контролю виконання індивідуальних планів.

Клієнтська частина застосунку розроблена із використанням бібліотеки React. Для організації навігації між сторінками використано React Router, а для взаємодії із серверною частиною застосовано бібліотеку Axios. Реалізований користувацький інтерфейс забезпечує роботу з планами, записами діяльності, нормативами часу, звітністю та механізмами керування користувачами. Побудова клієнтської частини за принципом односторінкового застосунку забезпечує швидку взаємодію користувача із системою та зручність роботи з даними.

Практичне значення роботи полягає у створенні програмного засобу, який забезпечує автоматизацію процесів планування та звітності діяльності науково-

педагогічних працівників, спрощує контроль виконання індивідуальних планів, зменшує обсяг рутинної роботи та підвищує ефективність управління освітнім процесом.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі сформульовано постановку задачі та визначено основні вимоги до програмного забезпечення. У другому розділі проведено аналіз існуючих інформаційних систем планування та звітності в освітньому процесі, а також розглянуто сучасні технології розроблення веборієнтованих інформаційних систем. У третьому розділі виконано проєктування електронного журналу, розроблено архітектуру системи, структуру бази даних, моделі взаємодії користувачів та функціональні модулі. У четвертому розділі описано реалізацію серверної та клієнтської частин застосунку, механізми авторизації та розмежування доступу, функціональні можливості системи, а також результати тестування та розгортання програмного забезпечення.

.

1. ПОСТАНОВКА ЗАДАЧІ

Планування та облік діяльності науково-педагогічних працівників є важливою складовою організації освітнього процесу в закладах вищої освіти. Виконання цих функцій передбачає ведення індивідуальних планів роботи, фіксацію результатів навчальної, наукової, методичної та організаційної діяльності, а також підготовку звітної документації. Використання електронних таблиць і текстових документів для виконання таких завдань не завжди забезпечує зручність роботи з даними, оперативність їх опрацювання та ефективний контроль виконання планових показників.

У межах дипломної роботи передбачається створення електронного журналу планування та звітності діяльності науково-педагогічних працівників. Розроблювана інформаційна система повинна забезпечувати ведення індивідуальних планів роботи, облік виконаних видів діяльності, автоматичний розрахунок годин відповідно до встановлених нормативів часу та формування звітних документів. Особливістю системи є підтримка декількох категорій користувачів, зокрема викладачів, завідувачів кафедр та адміністраторів, для яких передбачено різні права доступу до функціональних можливостей програмного забезпечення.

Результатом виконання роботи має стати веборієнтована інформаційна система для планування та звітності в освітньому процесі, яка забезпечуватиме автоматизацію основних операцій, пов'язаних із веденням індивідуальних планів та контролем їх виконання.

Для досягнення поставленого результату необхідно проаналізувати особливості предметної області та існуючі програмні рішення аналогічного призначення, сформувати вимоги до системи, спроєктувати її архітектуру та структуру бази даних, реалізувати серверну і клієнтську частини застосунку, забезпечити механізми автентифікації та авторизації користувачів, розробити модулі планування діяльності, обліку виконаних робіт і формування звітності, а також провести тестування розробленого програмного забезпечення.

Під час створення системи необхідно забезпечити централізоване зберігання даних, можливість керування нормативами часу, підтримку статусів планів, розмежування прав доступу між користувачами та зручний інтерфейс для роботи з інформацією. Важливим аспектом є автоматизація розрахунку трудомісткості окремих видів робіт і формування звітів на основі накопичених даних.

Отже, у межах дипломної роботи планується розробити електронний журнал планування та звітності діяльності науково-педагогічних працівників, який дозволить підвищити ефективність обліку виконаних робіт, спростити контроль виконання індивідуальних планів та автоматизувати процес підготовки звітної документації.

.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Аналіз існуючих інформаційних систем управління освітнім процесом

На етапі проєктування інформаційної системи для планування та звітності в освітньому процесі важливим є аналіз існуючих програмних рішень, що використовуються у закладах вищої освіти для автоматизації управління навчальною діяльністю, обліку результатів навчання, ведення документації та контролю освітнього процесу. Проведення такого аналізу дає змогу визначити основні підходи до організації інформаційних систем, структури функціональних модулів, способів взаємодії між користувачами та реалізації механізмів доступу до даних. Особливу увагу доцільно приділити системам, які підтримують електронні журнали, індивідуальні плани викладачів та інші інструменти цифрового супроводу освітньої діяльності.

У науковій праці О. Двірної «Управління розкладом та логістикою освітнього процесу засобами автоматизованої системи планування, організації, управління та контролю навчального процесу» розглянуто особливості функціонування автоматизованих систем управління освітньою діяльністю закладів вищої освіти. Автором проаналізовано питання організації розкладу занять, логістики освітнього процесу та використання інформаційних технологій для підтримки управлінських рішень. У роботі зазначається, що сучасні автоматизовані системи повинні забезпечувати інтеграцію різних підсистем освітнього середовища, підтримувати централізоване зберігання даних та автоматизувати процеси планування і контролю навчальної діяльності. Особливу увагу приділено автоматизованим системам університетського рівня, які поєднують функції управління навчальними планами, розкладом занять та інформаційним супроводом освітнього процесу. Результати дослідження підтверджують актуальність використання комплексних інформаційних систем для підвищення ефективності організації освітньої діяльності закладів вищої

освіти [13].

У роботі М. М. Косіюка «Автоматизована інформаційна система управління закладом вищої освіти» досліджуються принципи побудови сучасних інформаційних систем управління університетами та особливості їх впровадження в освітньому середовищі. Автор розглядає структуру інтегрованої інформаційної системи закладу вищої освіти, яка охоплює підсистеми роботи зі студентами, кадрового забезпечення, документообігу, освітнього процесу та аналітичної звітності. У дослідженні наголошується, що використання централізованої інформаційної системи дозволяє забезпечити узгодженість даних між структурними підрозділами університету, підвищити оперативність обробки інформації та спростити формування звітних документів. Автор також підкреслює важливість реалізації єдиного інформаційного середовища, у межах якого різні категорії користувачів працюють із спільною базою даних відповідно до визначених прав доступу [17].

У монографії «Розвиток інформаційних систем управління освітою як інструмент реалізації державної освітньої політики» за редакцією С. Л. Лондара розглянуто сучасні тенденції розвитку інформаційних систем у сфері освіти та їх роль у цифровій трансформації управлінських процесів. У роботі підкреслюється, що інформаційні системи управління освітою повинні забезпечувати підтримку процесів планування, моніторингу, аналізу та формування звітності на різних рівнях управління. Значна увага приділяється питанням інтеграції інформаційних ресурсів, використанню централізованих баз даних та створенню єдиного цифрового середовища для взаємодії учасників освітнього процесу. Автори зазначають, що розвиток інформаційних систем управління освітою є одним із ключових чинників підвищення ефективності функціонування закладів освіти та забезпечення якісного інформаційного супроводу управлінської діяльності [14].

Одним із прикладів комплексної інформаційної системи управління

освітнім процесом є автоматизована система керування навчальним процесом iZETA, яка використовується в Полтавський університет економіки і торгівлі. Дана система призначена для автоматизації значної кількості процесів, пов’язаних з організацією освітньої діяльності закладу вищої освіти. Її функціональні можливості охоплюють роботу з навчальними планами, формування розкладу занять, облік контингенту студентів, ведення електронних журналів успішності, підтримку документообігу та забезпечення доступу до різних інформаційних ресурсів університету. Система використовується як інструмент централізованого управління даними, що дозволяє зменшити кількість розрізнених інформаційних ресурсів та забезпечити єдиний інформаційний простір закладу освіти.

Особливістю iZETA є реалізація модульного підходу до організації функціональності системи. Окремі підсистеми відповідають за різні напрямки діяльності університету, зокрема планування освітнього процесу, облік результатів навчання, роботу з розкладом, формування звітності та підтримку адміністративних процесів. Така структура дозволяє забезпечити незалежний розвиток окремих модулів і водночас підтримувати їх взаємодію в межах єдиного програмного середовища. Фактично система реалізує концепцію інтегрованої автоматизованої системи управління освітнім процесом, у якій дані використовуються спільно різними категоріями користувачів.

З технічної точки зору подібні університетські системи будуються за клієнт-серверним принципом та передбачають використання централізованої бази даних, у якій зберігається інформація про студентів, викладачів, навчальні плани, дисципліни, результати навчання та інші об’єкти освітнього процесу. Доступ до функціональних можливостей системи здійснюється через вебінтерфейс із використанням механізмів автентифікації та розмежування прав доступу. Такий підхід забезпечує можливість роботи різних категорій користувачів у межах одного програмного середовища із доступом лише до дозволених функцій та даних.

Важливою характеристикою iZETA є орієнтація на підтримку управлінських процесів університету. Система не обмежується лише функціями електронного журналу успішності, а використовується як засіб комплексного супроводу освітньої діяльності. Це дозволяє автоматизувати підготовку звітної документації, зменшити обсяг ручного введення інформації та підвищити оперативність отримання аналітичних даних. У результаті система виступає не лише інструментом обліку, а й засобом підтримки управлінських рішень у сфері організації освітнього процесу.

Разом із тим аналіз функціональних можливостей iZETA показує, що основний акцент у системі зроблено на автоматизації загальноуніверситетських процесів, пов’язаних із навчальною діяльністю студентів та організацією освітнього процесу. Водночас окремі функції, пов’язані з веденням індивідуальних планів роботи викладачів, обліком різних видів науково-педагогічної діяльності та автоматизованим формуванням звітності за результатами їх виконання, не є центральною складовою системи. Саме тому розробка спеціалізованого електронного журналу планування та звітності діяльності науково-педагогічних працівників залишається актуальним напрямом удосконалення інформаційного забезпечення освітнього процесу.

Значний інтерес становить електронний журнал, що використовується у Київський національний економічний університет імені Вадима Гетьмана. Особливістю цього рішення є наявність електронного журналу поточної успішності та електронного індивідуального плану роботи викладача. На сайті університету розміщено окремі інструкції для викладачів щодо роботи з електронним індивідуальним планом, що свідчить про використання спеціалізованого програмного інструменту для ведення планування та звітності діяльності науково-педагогічних працівників.

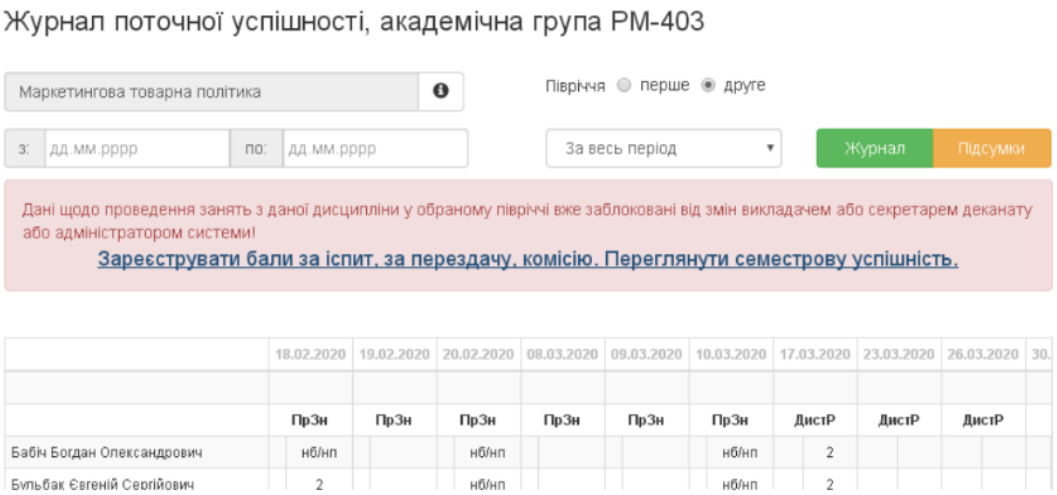
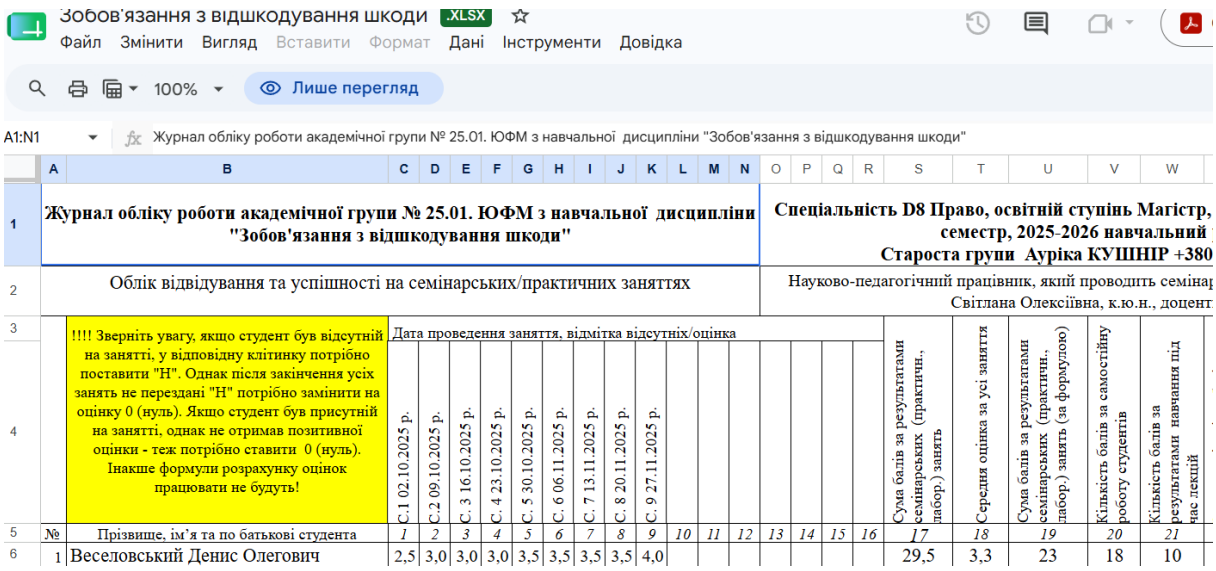


Рисунок 2.1. Фрагмент інтерфейсу електронного журналу успішності

У порівнянні з більшістю університетських систем, орієнтованих переважно на студентську успішність, даний ресурс демонструє приклад інтеграції функцій електронного журналу та індивідуального планування діяльності викладача [14].

Ще одним прикладом є інформаційна система «Електронний університет», що використовується у Хмельницький національний університет. Дане програмне рішення забезпечує автоматизацію різних процесів управління освітньою діяльністю та надає доступ до електронних сервісів для різних категорій користувачів. Подібні системи реалізують концепцію єдиного інформаційного середовища університету, у межах якого поєднуються засоби роботи зі студентськими даними, навчальними планами, розкладом та іншими складовими освітнього процесу [16].

Окрему категорію становлять електронні журнали успішності, які використовуються для обліку результатів навчання студентів. Прикладом є система електронних журналів успішності, впроваджена у Хмельницькому університеті управління та права імені Леоніда Юзькова, де журнали організовано за факультетами та академічними підрозділами. Основним призначенням таких систем є контроль навчальної діяльності студентів, облік оцінок і відвідування занять [15].



Зобов'язання з відшкодування шкоди .XLSX

Файл Змінити Вигляд Вставити Формат Дані Інструменти Довідка

100% Лише перегляд

A1:N1 Журнал обліку роботи академічної групи № 25.01. ЮФМ з навчальної дисципліни "Зобов'язання з відшкодування шкоди"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		**Журнал обліку роботи академічної групи № 25.01. ЮФМ з навчальної дисципліни "Зобов'язання з відшкодування шкоди"**													**Спеціальність D8 Право, освітній ступінь Магістр, семестр, 2025-2026 навчальний** **Староста групи Ауріка КУШНІР +380**									
2		Облік відвідування та успішності на семінарських/практичних заняттях													Науково-педагогічний працівник, який проводить семінар Світлана Олексіївна, к.ю.н., доцент									
3		!!!! Зверніть увагу, якщо студент був відсутній на занятті, у відповідну клітинку потрібно поставити "Н". Однак після закінчення усіх занять не перездані "Н" потрібно замінити на оцінку 0 (нуль). Якщо студент був присутній на занятті, однак не отримав позитивної оцінки - теж потрібно ставити 0 (нуль). Інакше формули розрахунку оцінок працювати не будуть!	Дата проведення заняття, відмітка відсутніх/оцінка																Сума балів за результатами семінарських (практичн., лабор.) занять	Середня оцінка за усі заняття	Сума балів за результатами семінарських (практичн., лабор.) занять (за формулою)	Кількість балів за самостійну роботу студентів	Кількість балів за результатами навчання під час лекцій
4			С.1 02.10.2025 р.	С.2 09.10.2025 р.	С. 3 16.10.2025 р.	С. 4 23.10.2025 р.	С. 5 30.10.2025 р.	С. 6 06.11.2025 р.	С. 7 13.11.2025 р.	С. 8 20.11.2025 р.	С. 9 27.11.2025 р.												
5	№	Прізвище, ім'я та по батькові студента	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*
6	1	Веселовський Денис Олегович	2,5	3,0	3,0	3,0	3,5	3,5	3,5	3,5	4,0								29,5	3,3	23	18	10

Рисунок 2.2. Фрагмент електронного журналу успішності, впроваджена у Хмельницькому університеті управління та права імені Леоніда Юзькова

Подібний підхід використовується також у програмному комплексі «ПС-Журнал успішності-Web», який функціонує у низці українських закладів вищої освіти.

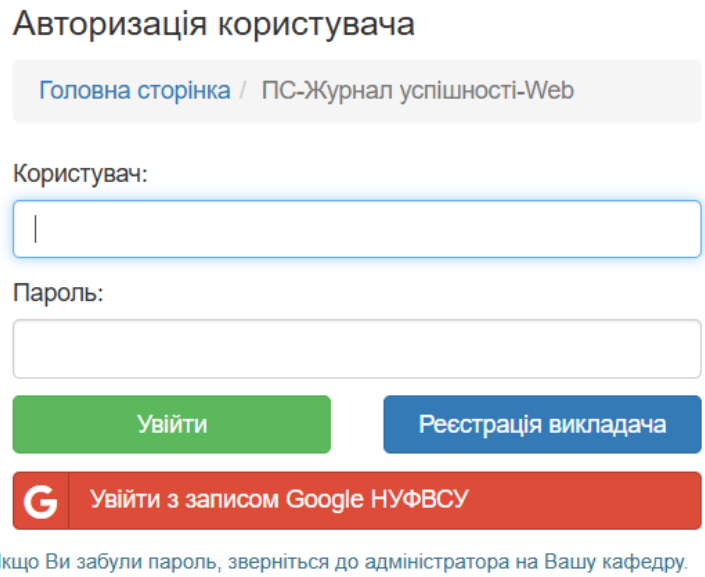


Рисунок 2.3. Авторизація у електронному журналі НУФВСУ

Система забезпечує електронний облік результатів навчання, підтримує авторизацію викладачів та організацію роботи з академічними групами.

Основний акцент таких рішень зроблено на контролі успішності студентів та формуванні відповідної звітності.

Показовим є також досвід Національний аерокосмічний університет імені М. Є. Жуковського «ХАІ», де електронний журнал розглядається як складова системи автоматизації управління університетом. У відповідному положенні зазначено, що система забезпечує облік академічної успішності студентів, навчальної діяльності викладачів, аналітичну обробку інформації та формування звітності.

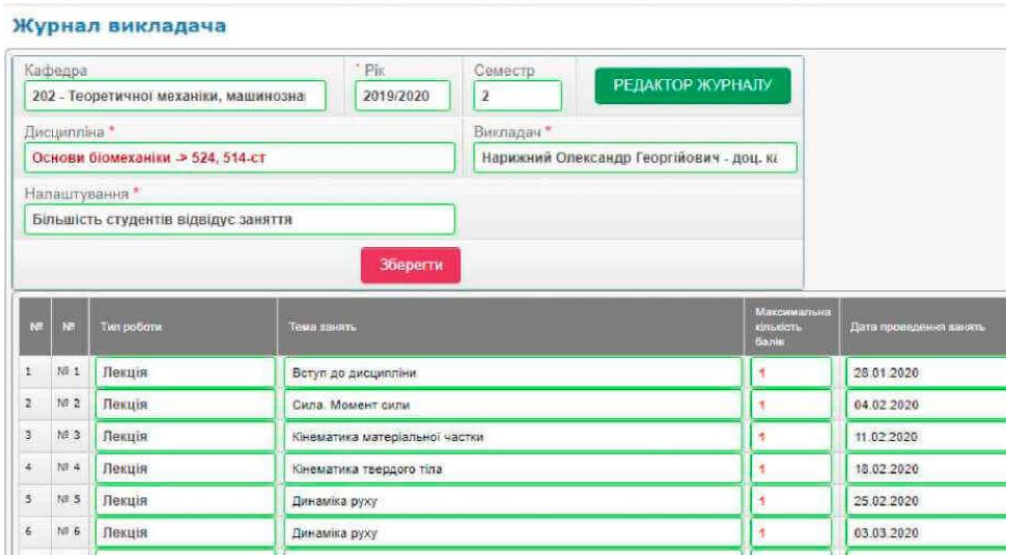


Рисунок 2.4. Фрагмент інтерфейсу журналу викладача ХАІ

Це демонструє тенденцію до інтеграції електронних журналів у більш масштабні інформаційні системи управління освітнім процесом [21].

Проведений аналіз показує, що більшість існуючих інформаційних систем закладів вищої освіти орієнтовані насамперед на автоматизацію навчального процесу, облік успішності студентів, ведення розкладу занять та підтримку документообігу. Водночас програмні рішення, призначені безпосередньо для ведення індивідуальних планів роботи викладачів, обліку різних видів діяльності науково-педагогічних працівників і формування відповідної звітності, поширені значно менше. У більшості випадків такі функції реалізуються як окремі модулі великих університетських

інформаційних систем або підтримуються за допомогою електронних таблиць та локальних програмних засобів.

Отже, аналіз існуючих інформаційних систем управління освітнім процесом свідчить про активне впровадження цифрових технологій у діяльність закладів вищої освіти. Разом із тим питання автоматизації планування та звітності діяльності викладачів залишається менш розвиненим порівняно із системами електронних журналів успішності студентів. Це підтверджує актуальність розробки спеціалізованого електронного журналу планування та звітності діяльності науково-педагогічних працівників, який забезпечуватиме централізоване ведення індивідуальних планів, облік виконаних робіт і формування звітної документації.

2.2. Аналіз електронних журналів планування та обліку діяльності викладачів

Поряд із системами управління освітнім процесом важливу роль у цифровізації діяльності закладів вищої освіти відіграють електронні журнали планування та обліку роботи науково-педагогічних працівників. Такі програмні засоби призначені для ведення індивідуальних планів викладачів, контролю виконання різних видів діяльності, обліку навчального навантаження та формування звітної документації. На відміну від традиційних електронних журналів успішності студентів, подібні системи орієнтовані насамперед на автоматизацію професійної діяльності викладача та підтримку управлінських процесів на рівні кафедри або закладу вищої освіти.

Одним із найбільш показових прикладів є система електронного індивідуального плану роботи викладача, яка використовується у Київський національний економічний університет імені Вадима Гетьмана. Система

забезпечує формування індивідуального плану науково-педагогічного працівника, внесення інформації про різні види діяльності та підготовку звітності за результатами виконаної роботи. Для користувачів розроблено окремі інструкції щодо роботи з електронним планом, що свідчить про його повноцінне використання в освітньому процесі. Особливістю даного рішення є інтеграція планування та звітності в єдиному програмному середовищі, що дозволяє здійснювати контроль виконання індивідуального плану протягом навчального року та спрощує підготовку підсумкових звітів [14].

Значний інтерес становить програмний комплекс «Індивідуальний план викладача», розроблений у Національному університеті біоресурсів і природокористування України. Система призначена для автоматизації процесів планування та обліку діяльності викладачів, забезпечує ведення індивідуальних планів, накопичення інформації про виконану роботу та формування відповідної документації. Подібні рішення демонструють тенденцію переходу від паперових форм обліку до централізованих електронних систем, що забезпечують зручність зберігання інформації та підвищують ефективність контролю виконання запланованих заходів.

У багатьох закладах вищої освіти функції електронного планування діяльності викладачів реалізуються як окремі модулі великих інформаційних систем управління освітнім процесом. Зокрема, автоматизовані системи університетського рівня можуть містити підсистеми управління навчальним навантаженням, формування індивідуальних планів, контролю виконання кафедральних показників та підготовки статистичної звітності. Такий підхід дозволяє використовувати спільну базу даних для різних напрямів діяльності університету та забезпечувати взаємозв'язок між освітнім процесом, кадровим обліком і плануванням роботи викладачів.

Аналіз існуючих рішень показує, що більшість систем планування діяльності викладачів реалізують схожий набір функціональних можливостей. До них належать ведення індивідуальних планів, облік виконання навчальної,

наукової, методичної та організаційної роботи, розрахунок годин відповідно до встановлених нормативів, формування звітів та забезпечення доступу до інформації відповідно до ролі користувача. Водночас рівень автоматизації окремих процесів суттєво відрізняється. Частина систем передбачає лише електронне зберігання даних, тоді як більш сучасні рішення реалізують автоматичні розрахунки, контроль виконання планів та генерацію звітної документації.

Проведений аналіз також свідчить про те, що більшість наявних систем створювалися для використання в межах конкретного закладу освіти та не набули широкого поширення за його межами. Унаслідок цього відсутні універсальні програмні рішення, які могли б бути адаптовані до потреб різних закладів вищої освіти без суттєвого доопрацювання. Крім того, значна частина існуючих систем має застарілий інтерфейс або обмежений набір функціональних можливостей, що не повністю відповідає сучасним вимогам до веборієнтованих інформаційних систем.

Отже, аналіз електронних журналів планування та обліку діяльності викладачів показує наявність потреби у створенні сучасних веборієнтованих рішень, здатних забезпечити автоматизацію процесів ведення індивідуальних планів, контролю виконання різних видів діяльності та формування звітності. Перспективним напрямом є розробка інформаційних систем, які поєднують функції планування, обліку та аналізу діяльності науково-педагогічних працівників у межах єдиного програмного середовища та забезпечують зручну взаємодію між викладачами, завідувачами кафедр і адміністраторами системи.

2.3. Аналіз технологій розроблення веборієнтованих інформаційних систем

Проведений аналіз інформаційних систем управління освітнім процесом показує, що сучасні програмні рішення даного класу реалізуються переважно

як веборієнтовані інформаційні системи. Такий підхід забезпечує централізоване зберігання даних, доступ до функціональних можливостей через веббраузер та можливість одночасної роботи великої кількості користувачів незалежно від їхнього місця перебування. Використання вебтехнологій дозволяє спростити супровід програмного забезпечення, оскільки оновлення системи виконується на сервері без необхідності встановлення нових версій програмного забезпечення на комп'ютери користувачів.

Аналіз розглянутих у попередньому підрозділі систем свідчить, що більшість із них побудовані на основі клієнт-серверної архітектури. Зокрема, система iZETA, яка використовується у Полтавському університеті економіки і торгівлі, функціонує як інтегроване інформаційне середовище закладу вищої освіти та забезпечує централізоване керування даними освітнього процесу. Аналогічний підхід використовується в системах електронного журналу КНЕУ та інформаційній системі «Електронний університет» Хмельницького національного університету. У всіх випадках користувач взаємодіє із системою через вебінтерфейс, тоді як основна обробка інформації виконується на сервері.

Важливою особливістю розглянутих рішень є використання централізованих баз даних. Такий підхід дозволяє уникнути дублювання інформації, забезпечує її цілісність та створює умови для одночасної роботи різних категорій користувачів. У більшості сучасних інформаційних систем закладів вищої освіти застосовуються реляційні системи керування базами даних, які підтримують складні зв'язки між сутностями предметної області та забезпечують надійне зберігання інформації.

Спільною характеристикою більшості проаналізованих систем є реалізація рольової моделі доступу. Користувачі отримують доступ лише до тих функцій і даних, які відповідають їхнім посадовим обов'язкам. Наприклад, викладачі можуть працювати з власними дисциплінами або індивідуальними планами, керівники підрозділів отримують можливість контролювати діяльність працівників відповідного структурного підрозділу, а адміністратори

виконують функції керування користувачами та налаштування системи. Такий підхід дозволяє підвищити рівень інформаційної безпеки та забезпечити впорядкованість роботи із даними.

Для реалізації взаємодії між окремими компонентами веборієнтованих інформаційних систем широко використовуються програмні інтерфейси прикладного програмування (API). Найбільш поширеним підходом є використання архітектурного стилю REST, який забезпечує обмін даними між клієнтською та серверною частинами за допомогою HTTP-запитів. Використання REST API дозволяє створювати незалежні клієнтські та серверні компоненти, що спрощує модернізацію системи та її інтеграцію з іншими програмними продуктами.

Для побудови користувацького інтерфейсу сучасних вебзастосунків активно використовуються JavaScript-фреймворки та бібліотеки, зокрема React, Angular і Vue.js. Їх застосування забезпечує створення інтерактивного інтерфейсу користувача та підтримку концепції односторінкових застосунків, що дозволяє виконувати більшість операцій без повного перезавантаження вебсторінки. Це підвищує швидкодію системи та покращує зручність роботи користувачів.

Для серверної частини інформаційних систем застосовуються різні програмні платформи, серед яких поширеними є ASP.NET Core, Spring Boot, Django та FastAPI. Використання сучасних серверних фреймворків дозволяє реалізувати механізми обробки запитів, автентифікації користувачів, керування бізнес-логікою та взаємодії з базами даних. Важливою вимогою до таких платформ є підтримка високої продуктивності, масштабованості та засобів захисту інформації.

Проведений аналіз показує, що сучасні інформаційні системи управління освітнім процесом будуються на основі спільних технологічних принципів: клієнт-серверної архітектури, централізованих баз даних, рольової моделі доступу та використання вебтехнологій для організації взаємодії користувачів

із системою. Саме такі підходи доцільно використовувати під час розроблення електронного журналу планування та звітності діяльності науково-педагогічних працівників.

З урахуванням проведеного аналізу для реалізації розроблюваної інформаційної системи доцільно використовувати сучасний технологічний стек, який включає бібліотеку React для побудови клієнтської частини, фреймворк FastAPI для реалізації серверної логіки, систему керування базами даних PostgreSQL для зберігання інформації та архітектурний стиль REST API для організації взаємодії між компонентами системи. Такий підхід забезпечує створення масштабованого, продуктивного та зручного у використанні програмного забезпечення.

2.4. Формування вимог до системи

Проведений аналіз існуючих інформаційних систем управління освітнім процесом та електронних журналів діяльності викладачів показав, що більшість наявних рішень орієнтовані переважно на автоматизацію навчального процесу, облік успішності студентів або виконання окремих управлінських функцій. Водночас процеси планування та звітності діяльності науково-педагогічних працівників часто реалізуються частково або залишаються недостатньо автоматизованими. У зв'язку з цим виникає необхідність розроблення спеціалізованої інформаційної системи, призначеної для ведення індивідуальних планів роботи викладачів, обліку виконаних видів діяльності та формування звітної документації.

Основним призначенням розроблюваної системи є автоматизація процесів планування, обліку та контролю діяльності науково-педагогічних працівників закладу вищої освіти. Система повинна забезпечувати

централізоване зберігання інформації про користувачів, індивідуальні плани роботи, нормативи часу та результати виконання різних видів діяльності.

Однією з ключових вимог є підтримка рольової моделі доступу. Передбачається використання трьох основних категорій користувачів: викладачів, завідувачів кафедр та адміністраторів. Викладач повинен мати можливість створювати та редагувати власні плани, вносити інформацію про виконані роботи та переглядати сформовані звіти. Завідувач кафедри повинен отримувати доступ до інформації про діяльність викладачів кафедри та здійснювати контроль виконання індивідуальних планів. Адміністратор має забезпечувати керування користувачами, нормативами часу та іншими довідковими даними системи.

Функціональні можливості системи повинні передбачати створення індивідуальних планів роботи викладачів на визначений період, ведення записів про виконання навчальної, наукової, методичної та організаційної діяльності, а також збереження інформації про обсяг виконаних робіт. Важливою вимогою є підтримка нормативів часу для різних видів діяльності та автоматичний розрахунок кількості годин на основі введених користувачем даних. Це дозволить мінімізувати кількість помилок під час підрахунків і забезпечити єдиний підхід до обліку навантаження.

Система повинна забезпечувати формування звітної документації на основі накопичених даних. Формування звітів має виконуватися автоматично з використанням інформації про виконані роботи та затверджені нормативи часу. Отримані звіти повинні бути придатними для подальшого використання в діяльності кафедри та закладу вищої освіти.

До важливих вимог належить забезпечення цілісності та достовірності інформації. Усі дані, що вводяться користувачами, повинні проходити перевірку коректності, а система має запобігати виникненню помилкових або неповних записів. Крім того, необхідно забезпечити захист персональних даних

користувачів та обмежити доступ до службової інформації відповідно до встановлених прав доступу.

Окрему увагу слід приділити вимогам до користувацького інтерфейсу. Система повинна забезпечувати зручну навігацію між функціональними модулями, зрозуміле представлення інформації та швидкий доступ до основних операцій. Інтерфейс має бути орієнтований на повсякденне використання викладачами та працівниками кафедр, що передбачає простоту роботи з формами введення даних, переглядом планів і формуванням звітів.

Таким чином, розроблювана інформаційна система повинна поєднувати функції планування діяльності, обліку виконаних робіт, автоматизованого розрахунку навантаження та формування звітності в межах єдиного програмного середовища. Реалізація зазначених вимог дозволить підвищити ефективність ведення індивідуальних планів роботи науково-педагогічних працівників та спростити контроль виконання запланованих показників.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Проєктування архітектури інформаційної системи

Одним із ключових етапів розроблення інформаційної системи є проєктування її архітектури. Від обраної архітектурної моделі залежать ефективність взаємодії між компонентами системи, можливість подальшого розвитку програмного забезпечення, зручність супроводу та масштабованість рішення. Для розроблюваного електронного журналу планування та звітності діяльності науково-педагогічних працівників було обрано багаторівневу клієнт-серверну архітектуру, яка передбачає розподіл функціональності між клієнтською частиною, серверною логікою та системою зберігання даних.

Архітектура розробленої системи включає три основні рівні. Перший рівень представлений клієнтською частиною, яка забезпечує взаємодію користувача із системою. Другий рівень утворює серверна частина, відповідальна за реалізацію бізнес-логіки, обробку запитів та взаємодію з базою даних. Третій рівень складається із системи керування базою даних, яка забезпечує централізоване зберігання інформації про користувачів, плани роботи, записи діяльності та нормативи часу.

У розробленій системі взаємодія між клієнтською та серверною частинами здійснюється через REST API. Користувач виконує певні дії через вебінтерфейс, після чого клієнтська частина надсилає HTTP-запит до відповідного API-маршруту серверної частини. Сервер виконує необхідну обробку запиту, взаємодіє з базою даних та повертає результат у форматі JSON. Такий підхід забезпечує незалежність між компонентами системи та спрощує подальшу модернізацію програмного забезпечення.

Загальна структура інформаційної системи виглядає наступним чином:

```
edu-planner/
├── backend/
│   ├── app/
```

```
│   │   ├── api/
│   │   │   ├── auth.py          # POST /register, /login; GET /me
│   │   │   ├── plans.py         # CRUD планів та WorkEntry
│   │   │   ├── norms.py         # CRUD норм часу
│   │   │   ├── reports.py       # GET Excel-звіт
│   │   │   └── users.py         # Управління користувачами (admin)
│   │   ├── models/
│   │   │   ├── user.py          # Модель User
│   │   │   ├── plan.py          # Модель Plan
│   │   │   ├── work_entry.py    # Модель WorkEntry
│   │   │   └── norm.py          # Модель WorkNorm
│   │   ├── schemas/
│   │   │   ├── user.py          # Pydantic схеми користувача
│   │   │   ├── plan.py          # Pydantic схеми плану
│   │   │   └── norm.py          # Pydantic схеми норм
│   │   └── services/
│   │       ├── auth_service.py  # bcrypt, JWT
│   │       ├── calculator.py    # eval-калькулятор
│   │       └── report_generator.py  # openpyxl Excel
│   ├── config.py                # Pydantic Settings (.env)
│   ├── database.py              # SQLAlchemy engine, get_db()
│   ├── main.py                  # FastAPI app, роутери, міграції
│   ├── create_admin.py          # Скрипт першого адміна
│   ├── Dockerfile
│   └── requirements.txt
├── frontend/
│   ├── src/
│   │   ├── api/                 # client.js, auth.js, plans.js ...
│   │   ├── context/
│   │   │   └── AuthContext.jsx  # user, login(), logout()
│   │   ├── components/
│   │   │   ├── Layout.jsx       # Хедер, навігація
│   │   │   └── PrivateRoute.jsx # Захист маршрутів
│   │   ├── pages/
│   │   │   ├── LoginPage.jsx
│   │   │   ├── RegisterPage.jsx
│   │   │   ├── PendingPage.jsx  # Сторінка очікування
│   │   │   ├── PlansPage.jsx
│   │   │   ├── PlanDetailPage.jsx
│   │   │   ├── NormsPage.jsx
│   │   │   └── UsersPage.jsx
│   │   ├── utils/
│   │   │   └── validate.js      # Валідація форм, evalFormula()
│   │   ├── App.jsx
│   │   ├── main.jsx
│   │   └── index.css
│   ├── nginx.conf
│   ├── Dockerfile
│   └── package.json
├── docker-compose.yml
├── render.yaml                  # Render Blueprint
├── .env                         # Локальні змінні (не в git)
├── .gitignore
└── generate_docs.py             # Цей скрипт
```

Клієнтська частина системи реалізована за компонентним принципом. Основними функціональними елементами є сторінки авторизації, перегляду

планів, редагування записів діяльності, роботи з нормативами часу та формування звітності. Такий підхід дозволяє розділити інтерфейс на незалежні компоненти та спрощує подальший супровід програмного коду.

Серверна частина реалізована відповідно до принципів розподілу відповідальності між окремими модулями. Структура проєкту передбачає виділення моделей даних, API-маршрутів, схем валідації та сервісних компонентів. Основними функціональними модулями серверної частини є модуль автентифікації користувачів, модуль керування планами роботи, модуль керування нормативами часу, модуль роботи із записами діяльності та модуль формування звітності.

Важливим елементом архітектури є використання рольової моделі доступу. У системі реалізовано три основні ролі користувачів: teacher, head та admin. Після проходження автентифікації сервер визначає роль користувача та надає доступ лише до дозволених функціональних можливостей. Такий підхід забезпечує захист службової інформації та дозволяє реалізувати різні сценарії роботи для окремих категорій користувачів.

Для зберігання інформації використовується реляційна база даних PostgreSQL. Доступ до даних здійснюється через ORM SQLAlchemy, що дозволяє працювати із записами бази даних за допомогою програмних об'єктів. Такий підхід спрощує розроблення програмного забезпечення та забезпечує незалежність бізнес-логіки від конкретних SQL-запитів.

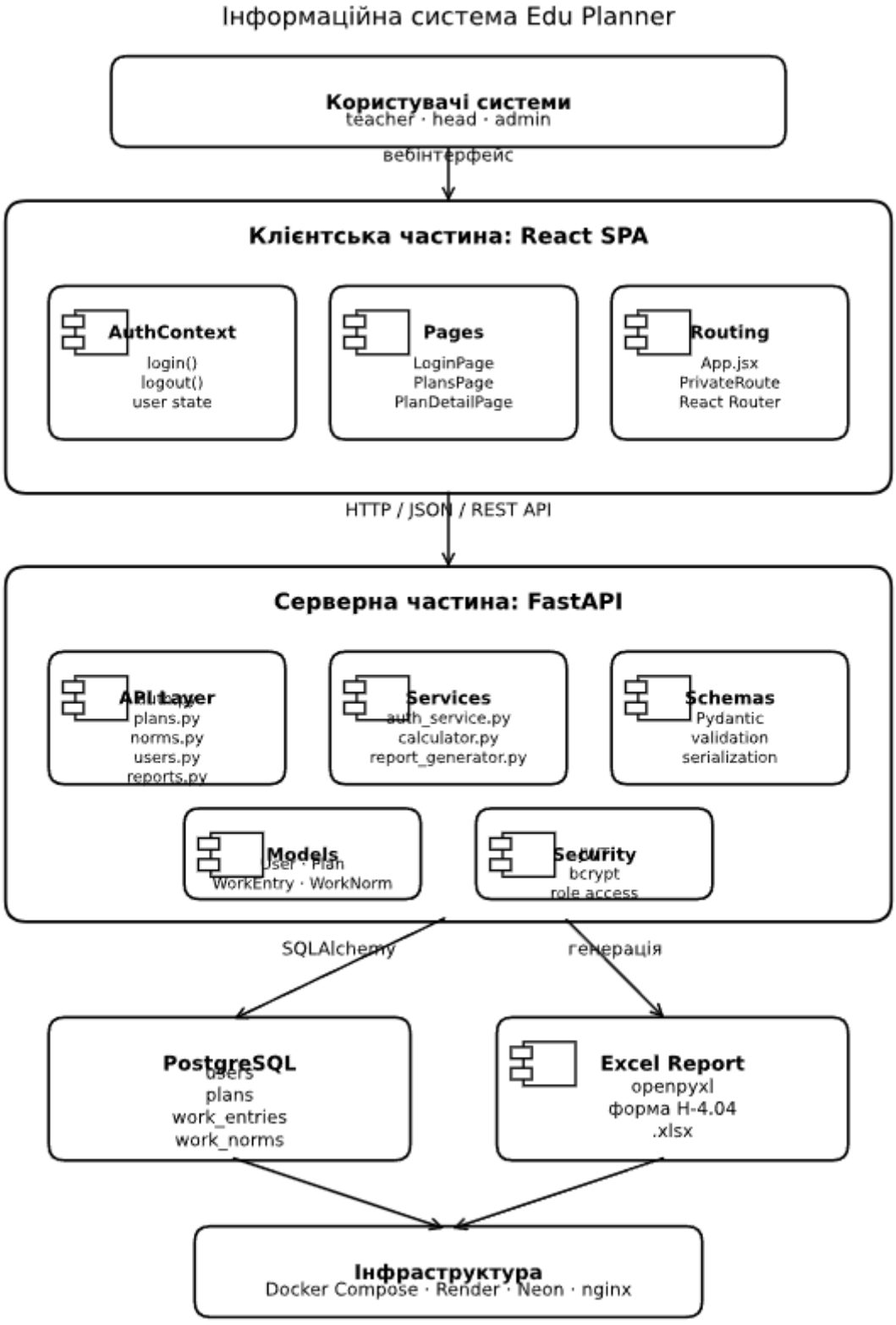


Рисунок 3.1. Архітектура інформаційної системи планування та звітності діяльності науково-педагогічних працівників

Архітектура системи також передбачає використання окремого сервісного рівня для реалізації допоміжної функціональності. Зокрема, модуль

формування звітності виконує автоматичне створення звітних документів на основі накопичених даних про виконані роботи та нормативів часу. Виділення такої функціональності в окремий сервісний компонент дозволяє уникнути дублювання коду та підвищує рівень структурованості програмного забезпечення.

Для реалізації механізмів захисту інформації використовується автентифікація на основі JSON Web Token. Після успішного входу до системи користувач отримує токен доступу, який використовується для подальшої взаємодії із серверною частиною. Це дозволяє забезпечити контроль доступу до ресурсів системи та підтримувати безпечну взаємодію між клієнтом і сервером.

Таким чином, архітектура розробленої інформаційної системи базується на клієнт-серверному підході та забезпечує розподіл функцій між інтерфейсом користувача, бізнес-логікою та рівнем зберігання даних. Використання багаторівневої структури, REST API, рольової моделі доступу та централізованої бази даних створює основу для надійного функціонування електронного журналу планування та звітності діяльності науково-педагогічних працівників.

3.2. Проєктування структури бази даних

Одним із найважливіших етапів проєктування інформаційної системи є розроблення структури бази даних, оскільки саме вона забезпечує зберігання, впорядкування та обробку інформації, необхідної для функціонування всіх програмних модулів. Для розроблюваного електронного журналу планування та звітності діяльності науково-педагогічних працівників була спроєктована реляційна база даних, яка забезпечує зберігання відомостей про користувачів,

індивідуальні плани роботи, записи про виконані види діяльності та нормативи часу.

Під час проєктування структури бази даних було враховано особливості предметної області та функціональні вимоги до системи. Основною метою проєктування стало забезпечення цілісності даних, підтримка зв'язків між об'єктами предметної області та можливість подальшого розширення функціональних можливостей системи без суттєвої зміни її структури.

Центральне місце в базі даних займає сутність користувача, яка представлена таблицею User. У ній зберігаються облікові дані користувачів системи, необхідні для автентифікації та авторизації. До основних атрибутів належать ідентифікатор користувача, логін, пароль у зашифрованому вигляді, прізвище та ім'я, електронна адреса, роль користувача та службова інформація, необхідна для роботи системи. Кожен користувач може належати до однієї з ролей: teacher, head або admin.

Для зберігання інформації про індивідуальні плани роботи використовується таблиця Plan. Кожен план належить конкретному викладачу та містить відомості про навчальний рік, дату створення, поточний статус та інші службові дані. Реалізований підхід дозволяє одному користувачу мати декілька планів для різних навчальних періодів, що забезпечує накопичення історичних даних та можливість аналізу діяльності за попередні роки.

Основною таблицею для обліку діяльності викладачів є WorkEntry. Вона містить записи про виконані види робіт та безпосередньо пов'язана з таблицею Plan. Для кожного запису зберігається інформація про категорію діяльності, назву роботи, її кількісні характеристики, розраховану трудомісткість та інші параметри, необхідні для формування звітності. Саме на основі даних цієї таблиці виконується автоматичний підрахунок загального навантаження та підготовка звітних документів.

Для забезпечення автоматизації розрахунків у системі передбачена таблиця WorkNorm. У ній зберігаються нормативи часу для різних видів

діяльності науково-педагогічних працівників. Кожен норматив містить назву виду роботи, одиницю вимірювання та кількість годин, що відповідає одиниці виконаної діяльності. Використання окремої таблиці нормативів дозволяє централізовано керувати правилами розрахунку навантаження та забезпечує єдиний підхід до обліку діяльності всіх користувачів системи.

Між таблицями бази даних встановлено відповідні зв'язки. Один користувач може мати багато планів, тому між таблицями User та Plan реалізовано зв'язок типу «один до багатьох». Аналогічно один план може містити багато записів про виконані роботи, що реалізовано через зв'язок «один до багатьох» між таблицями Plan та WorkEntry. Таблиця WorkNorm використовується як довідник нормативів часу та пов'язана із записами діяльності логічно через види робіт, які використовуються під час виконання розрахунків.

Загальну структуру бази даних наведено на рисунку 3.2.

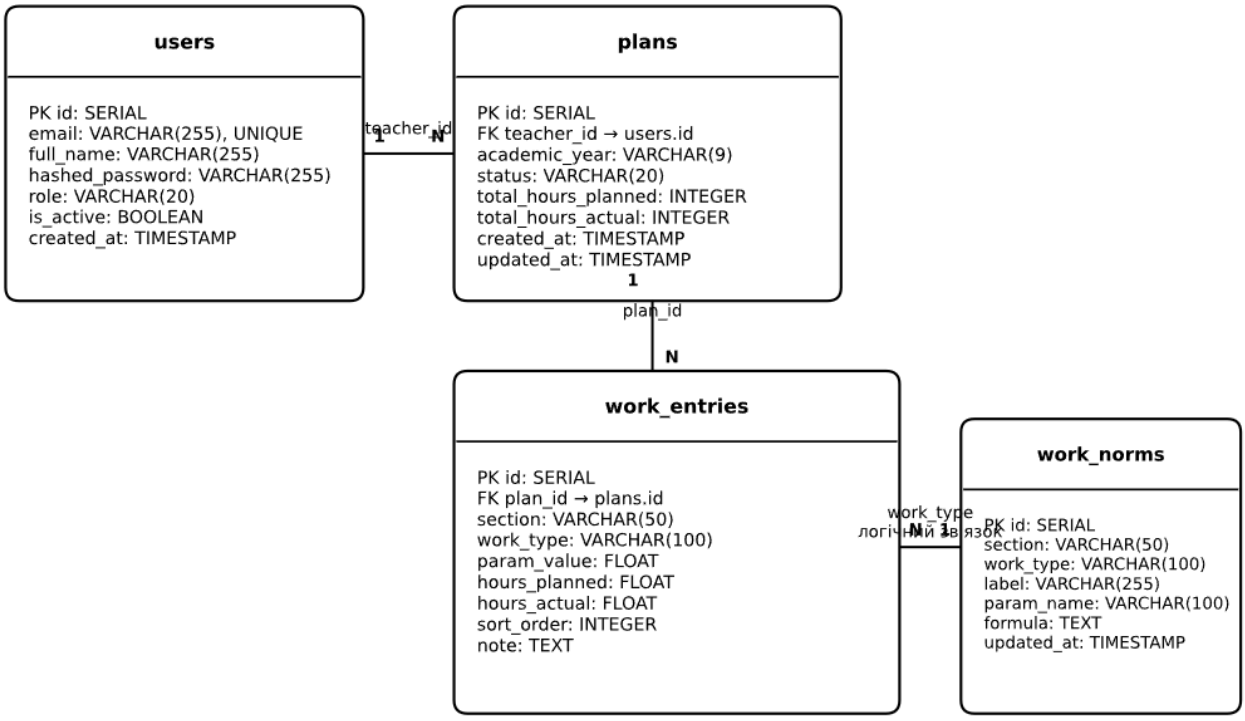


Рисунок 3.2. ER діаграма бази даних розробленого веб застосунку

Розроблена структура бази даних відповідає вимогам нормалізації та дозволяє уникнути дублювання інформації. Розподіл даних між окремими таблицями забезпечує підвищення продуктивності системи, спрощує підтримку цілісності інформації та створює умови для подальшого розширення функціональних можливостей програмного забезпечення.

Таким чином, спроєктована база даних забезпечує зберігання всіх даних, необхідних для функціонування електронного журналу планування та звітності діяльності науково-педагогічних працівників. Використання взаємопов'язаних таблиць User, Plan, WorkEntry та WorkNorm дозволяє реалізувати механізми обліку діяльності, автоматичного розрахунку навантаження, формування звітності та розмежування доступу між користувачами системи.

3.3. Проєктування ролей користувачів і прав доступу

Однією з важливих вимог до розроблюваної інформаційної системи є забезпечення контрольованого доступу до функціональних можливостей та даних відповідно до посадових обов'язків користувачів. Оскільки електронний журнал використовується різними категоріями працівників закладу вищої освіти, необхідно реалізувати механізм розмежування прав доступу, який забезпечує захист інформації та визначає перелік доступних операцій для кожного користувача.

У розробленій системі використовується рольова модель доступу (Role-Based Access Control, RBAC), відповідно до якої кожному користувачу призначається певна роль. Права доступу визначаються не індивідуально для кожного користувача, а на рівні ролі, що спрощує адміністрування системи та забезпечує єдиний підхід до керування доступом.

Відповідно до вимог предметної області в системі реалізовано три

основні ролі: викладач (teacher), завідувач кафедри (head) та адміністратор (admin). Інформація про роль користувача зберігається у відповідному полі таблиці User та використовується під час перевірки прав доступу до функціональних модулів системи.

Роль викладача є базовою роллю системи. Користувач із цією роллю має можливість працювати лише з власними даними. Викладач може створювати індивідуальні плани роботи, вносити записи про виконану навчальну, наукову, методичну та організаційну діяльність, переглядати результати розрахунків навантаження та формувати звіти на основі власних даних. Доступ до інформації інших користувачів для цієї ролі обмежений.

Роль завідувача кафедри призначена для здійснення контролю за діяльністю викладачів кафедри. Користувачі з даною роллю отримують розширені права доступу до інформації про плани та результати роботи викладачів. Завідувач кафедри може переглядати дані підпорядкованих працівників, контролювати виконання індивідуальних планів та аналізувати сформовану звітність. Це дозволяє використовувати систему як інструмент моніторингу виконання запланованих показників на рівні структурного підрозділу.

Найвищий рівень доступу має адміністратор системи. Дана роль забезпечує керування користувачами, підтримку довідкової інформації та налаштування параметрів функціонування системи. Адміністратор має можливість створювати нові облікові записи, змінювати ролі користувачів, редагувати нормативи часу та виконувати інші адміністративні операції. Крім того, адміністратор отримує доступ до всіх даних системи незалежно від їх належності конкретним користувачам.

Реалізація рольової моделі доступу здійснюється на серверній частині інформаційної системи. Після проходження автентифікації користувач отримує токен доступу, який містить інформацію про його обліковий запис та роль. Під час виконання кожного запиту сервер перевіряє наявність необхідних прав

доступу та визначає можливість виконання відповідної операції. Такий підхід забезпечує захист від несанкціонованого доступу до даних навіть у випадку спроби прямого звернення до API.

Для реалізації механізму контролю доступу використовується централізована перевірка ролей користувачів перед виконанням критичних операцій. Зокрема, окремі маршрути API можуть бути доступними лише адміністраторам або користувачам із розширеними правами. Це дозволяє забезпечити додатковий рівень безпеки та виключити можливість несанкціонованої зміни даних.

Схема взаємодії користувачів із функціональними можливостями системи може бути представлена у вигляді UML-діаграми варіантів використання (Use Case Diagram), наведеної на рисунку 3.3. На діаграмі відображено основні ролі користувачів та перелік доступних для них функцій.

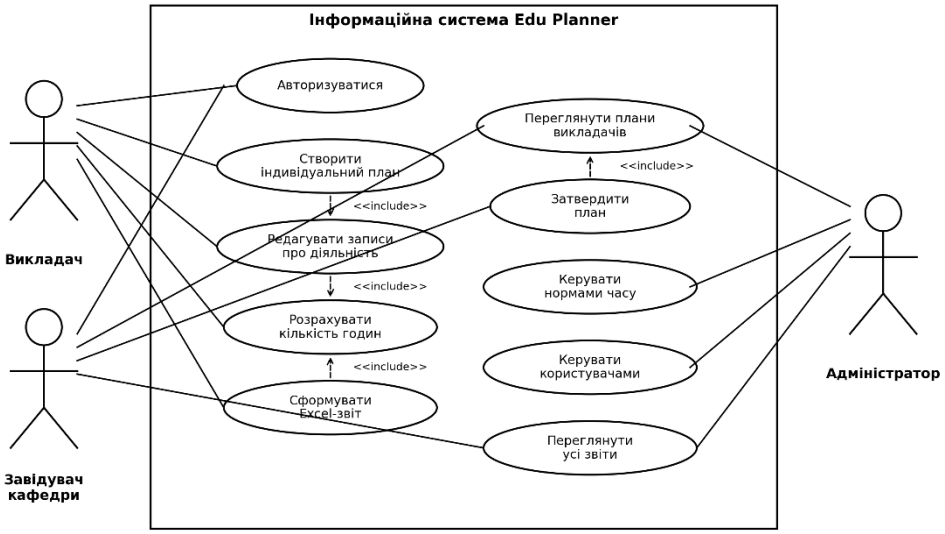


Рисунок 3.3. Діаграма варіантів використання інформаційної системи

Таким чином, використання рольової моделі доступу дозволяє забезпечити безпечну роботу інформаційної системи та реалізувати розмежування функціональних можливостей відповідно до посадових обов'язків користувачів. Реалізований механізм контролю доступу відповідає

вимогам предметної області та створює умови для надійного функціонування електронного журналу планування та звітності діяльності науково-педагогічних працівників.

3.4. Проєктування функціональних модулів електронного журналу

Функціональна структура інформаційної системи визначає перелік програмних компонентів, необхідних для реалізації поставлених завдань, а також принципи їх взаємодії між собою. Під час проєктування електронного журналу планування та звітності діяльності науково-педагогічних працівників функціональні можливості системи було розподілено між окремими модулями відповідно до їх призначення. Такий підхід дозволяє забезпечити логічну організацію програмного забезпечення, спрощує супровід системи та створює передумови для її подальшого розширення.

Центральним модулем системи є модуль автентифікації та авторизації користувачів. Його призначення полягає у забезпеченні безпечного доступу до функціональних можливостей електронного журналу. Модуль виконує перевірку облікових даних користувача, визначає його роль та забезпечує доступ лише до дозволених функцій системи. Результатом роботи модуля є формування сеансу користувача та надання відповідних прав доступу до інформаційних ресурсів.

Наступним функціональним компонентом є модуль керування користувачами. Даний модуль використовується для створення, редагування та перегляду облікових записів користувачів системи. Крім того, він забезпечує призначення ролей та підтримку актуальної інформації про викладачів, завідувачів кафедр і адміністраторів. Використання окремого модуля для роботи з користувачами дозволяє централізувати процес адміністрування системи та забезпечити контроль доступу до її функціональних можливостей.

Основу електронного журналу становить модуль керування

індивідуальними планами роботи. Його призначення полягає у створенні та веденні планів діяльності викладачів на визначений навчальний період. Модуль забезпечує перегляд інформації про заплановані види робіт, їх редагування та збереження в базі даних. Кожен план пов'язаний із конкретним користувачем, що дозволяє здійснювати персоніфікований облік діяльності науково-педагогічних працівників.

Для фіксації результатів виконаної роботи передбачено модуль обліку діяльності викладача. За допомогою цього модуля користувач може вносити інформацію про виконані навчальні, наукові, методичні та організаційні роботи. Усі записи зберігаються в системі та використовуються для подальшого розрахунку навантаження й формування звітності. Реалізація окремого модуля дозволяє відокремити процес ведення індивідуального плану від процесу накопичення фактичних результатів діяльності.

Важливою складовою системи є модуль керування нормативами часу. Його основним завданням є зберігання та підтримка нормативів, що використовуються під час розрахунку трудомісткості різних видів робіт. Модуль забезпечує можливість додавання нових нормативів, редагування існуючих значень та їх використання під час автоматичного обчислення навантаження. Такий підхід дозволяє централізувати правила розрахунку та забезпечити однакові умови обліку діяльності для всіх користувачів системи.

Для автоматизації підрахунків у системі реалізовано модуль розрахунку навантаження. Він використовує інформацію про нормативи часу та введені користувачем параметри виконаних робіт для визначення кількості годин за кожним видом діяльності. Результати обчислень автоматично відображаються у відповідних розділах індивідуального плану та використовуються під час формування звітних документів. Автоматизація даного процесу дозволяє мінімізувати кількість помилок, які можуть виникати під час ручного підрахунку.

Окремий функціональний модуль призначений для формування звітності.

Його робота базується на аналізі накопичених у системі даних про діяльність викладача та результатах розрахунку навантаження. Модуль забезпечує автоматичне створення звітних документів у встановленому форматі та дозволяє отримувати актуальну інформацію про виконання індивідуальних планів роботи. Завдяки цьому значно скорочується час, необхідний для підготовки звітної документації.

Для завідувачів кафедр передбачено модуль контролю та моніторингу діяльності викладачів. Його функціональність дозволяє переглядати плани підпорядкованих працівників, аналізувати стан їх виконання та отримувати узагальнену інформацію щодо результатів роботи кафедри. Використання такого модуля підвищує ефективність контролю та спрощує процес прийняття управлінських рішень.

Взаємодія між функціональними модулями здійснюється через серверну частину системи, яка забезпечує обробку запитів, перевірку прав доступу та взаємодію з базою даних. Завдяки цьому всі модулі працюють у межах єдиного інформаційного середовища та використовують спільні дані, що виключає дублювання інформації та забезпечує її цілісність.

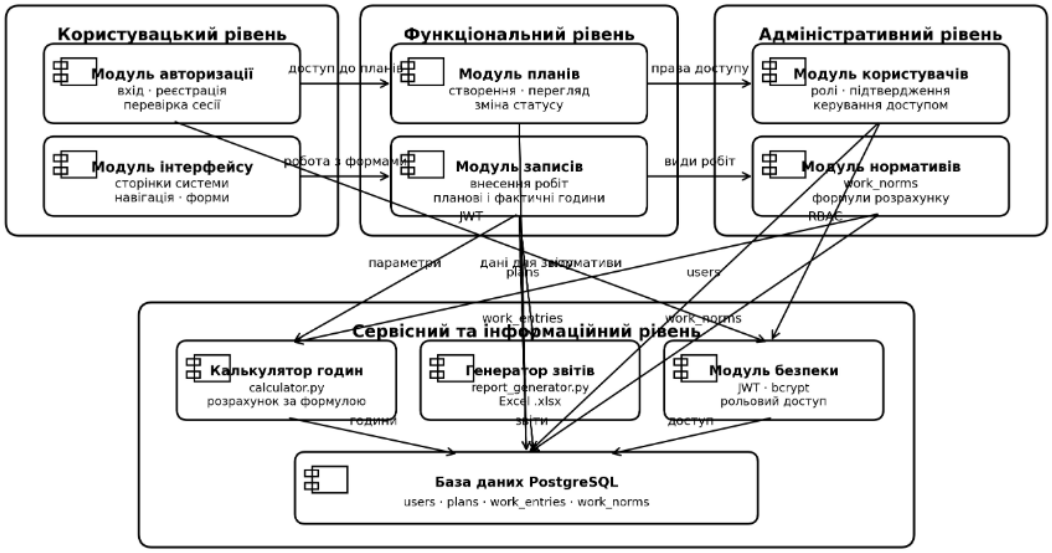


Рисунок 3.4. Діаграма компонентів функціональних модулів електронного журналу

Загальну структуру функціональних модулів електронного журналу подано у вигляді UML-діаграми компонентів або функціональної схеми, наведених на рисунку 3.4.

Таким чином, функціональна структура розробленої інформаційної системи охоплює модулі автентифікації та авторизації, керування користувачами, ведення індивідуальних планів, обліку діяльності, керування нормативами часу, автоматичного розрахунку навантаження, формування звітності та контролю діяльності викладачів. Сукупність зазначених компонентів забезпечує реалізацію всіх основних функцій електронного журналу планування та звітності діяльності науково-педагогічних працівників.

3.5. Проєктування REST API інформаційної системи

Важливою складовою архітектури розробленої інформаційної системи є програмний інтерфейс прикладного програмування REST API, який забезпечує взаємодію між клієнтською та серверною частинами застосунку. Використання REST-підходу дозволяє розділити інтерфейс користувача та бізнес-логіку системи, що спрощує супровід програмного забезпечення, підвищує його масштабованість та забезпечує незалежність окремих компонентів.

У розробленому електронному журналі REST API реалізує набір кінцевих точок (endpoint), через які клієнтська частина виконує операції отримання, створення, оновлення та видалення даних. Передача інформації між компонентами системи здійснюється за протоколом HTTP, а обмін даними реалізується у форматі JSON. Такий підхід забезпечує універсальність взаємодії між сервером і клієнтом та відповідає сучасним вимогам до розроблення веборієнтованих інформаційних систем. Перелік ендпоінтів наведено нижче у

вигляді таблиць.

Авторизація (/auth)

Метод	Шлях	Доступ	Опис
POST	/auth/register	Публічний	Реєстрація (роль new_user)
POST	/auth/login	Публічний	Логін, повертає JWT токен
GET	/auth/me	Авторизований	Дані поточного користувача

Плани (/plans)

Метод	Шлях	Доступ	Опис
GET	/plans/	teacher/head/admin	Список планів
POST	/plans/	teacher	Створити план
GET	/plans/{id}	teacher/head/admin	Деталі плану
PATCH	/plans/{id}	teacher/head/admin	Оновити статус
DELETE	/plans/{id}	teacher	Видалити чернетку
POST	/plans/{id}/entries	teacher/head/admin	Додати запис
PATCH	/plans/{id}/entries/{eid}	teacher/head/admin	Оновити запис
DELETE	/plans/{id}/entries/{eid}	teacher/head/admin	Видалити запис

Норми (/norms)

Метод	Шлях	Доступ	Опис
GET	/norms/	Авторизований	Список норм (фільтр ?section=...)
GET	/norms/{id}	Авторизований	Одна норма
POST	/norms/	admin	Створити норму
PATCH	/norms/{id}	admin	Оновити норму
DELETE	/norms/{id}	admin	Видалити норму

Користувачі (/users)

Метод	Шлях	Доступ	Опис
GET	/users/	admin	Список усіх користувачів
GET	/users/pending	admin	Список new_user
PATCH	/users/{id}/role	admin	Змінити роль

Звіти (/reports)

Метод	Шлях	Доступ	Опис
GET	/reports/plans/{id}/excel	Авторизований	Завантажити Excel (форма Н-4.04)

Під час проєктування REST API було враховано функціональну структуру системи та виділено окремі групи маршрутів відповідно до основних модулів програмного забезпечення. Насамперед передбачено маршрути автентифікації користувачів. Дана група API забезпечує виконання входу до

системи, перевірку облікових даних та формування токенів доступу. Після успішної автентифікації користувач отримує токен, який використовується для виконання подальших запитів до захищених ресурсів системи.

Окрему групу становлять маршрути роботи з користувачами. Вони використовуються для отримання інформації про користувачів системи, створення нових облікових записів, редагування відомостей та керування ролями. Доступ до таких операцій надається лише користувачам із відповідними адміністративними правами.

Для реалізації основного функціоналу електронного журналу передбачено маршрути керування індивідуальними планами роботи. Вони забезпечують створення нових планів, перегляд наявних записів, редагування інформації та отримання детальних відомостей про конкретний план. Усі операції виконуються відповідно до ролі користувача та його прав доступу.

Важливе місце в структурі REST API займають маршрути роботи із записами діяльності викладачів. Через них здійснюється додавання нових видів робіт, редагування існуючих записів, перегляд накопиченої інформації та видалення помилково створених записів. Саме ці дані використовуються системою під час автоматичного розрахунку навантаження та формування звітності.

Для підтримки механізму автоматичних розрахунків у системі реалізовано окрему групу маршрутів роботи з нормативами часу. Вони забезпечують отримання переліку нормативів, їх оновлення та керування параметрами, що використовуються під час визначення трудомісткості різних видів діяльності. Централізація таких операцій дозволяє підтримувати єдині правила розрахунку для всіх користувачів системи.

Окремий набір API-маршрутів призначений для формування звітності. Через відповідні запити користувач може отримати сформований звіт на основі даних індивідуального плану та записів про виконану діяльність. Результатом роботи даного модуля є створення файлів звітної документації, які можуть бути

використані для подальшого аналізу та подання до структурних підрозділів закладу вищої освіти.

Під час проєктування REST API особливу увагу приділено забезпеченню безпеки. Для захисту ресурсів системи використовується механізм автентифікації на основі JSON Web Token. Кожен захищений маршрут виконує перевірку токена доступу та прав користувача перед виконанням операції. Такий підхід дозволяє запобігти несанкціонованому доступу до даних та забезпечує розмежування функціональних можливостей між викладачами, завідувачами кафедр і адміністраторами.

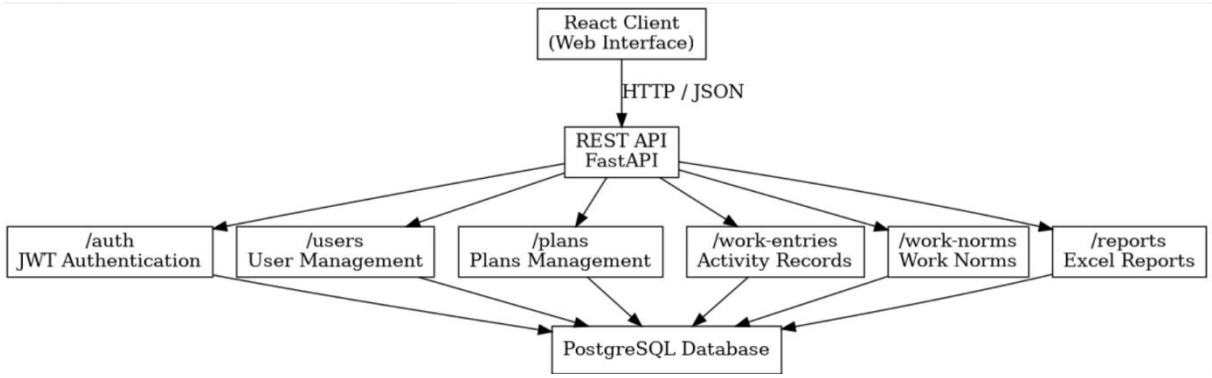


Рисунок 3.5. Структура REST API інформаційної системи Edu Planner

На рисунку 3.5 наведено структуру REST API розробленої інформаційної системи. Центральним елементом взаємодії між клієнтською та серверною частинами є REST API, яке забезпечує роботу модулів автентифікації, керування користувачами, індивідуальними планами, записами діяльності, нормативами часу та формування звітності. Усі функціональні модулі взаємодіють із централізованою базою даних PostgreSQL.

Таким чином, спроєктований REST API забезпечує взаємодію між клієнтською та серверною частинами електронного журналу, підтримує роботу з користувачами, індивідуальними планами, записами діяльності, нормативами часу та звітністю. Використання REST-архітектури дозволяє забезпечити модульність системи, спростити її супровід та створити основу для подальшого

розширення функціональних можливостей програмного забезпечення.

3.6. Проєктування інтерфейсу користувача

Інтерфейс користувача є одним із найважливіших компонентів інформаційної системи, оскільки саме через нього здійснюється взаємодія користувачів із функціональними можливостями електронного журналу. Від якості проєктування інтерфейсу залежить зручність роботи із системою, швидкість виконання операцій та ефективність використання її функціоналу. Під час розроблення інтерфейсу електронного журналу планування та звітності діяльності науково-педагогічних працівників основна увага приділялася забезпеченню простоти навігації, логічності структури сторінок та доступності основних функцій для різних категорій користувачів.

Проєктування інтерфейсу виконувалося з урахуванням ролей користувачів, реалізованих у системі. Після проходження автентифікації користувач отримує доступ до функціональних можливостей відповідно до своєї ролі. Такий підхід дозволяє приховати непотрібні елементи інтерфейсу та спростити роботу з системою. Викладачі отримують доступ до модулів планування діяльності, внесення записів та формування звітності, завідувачі кафедр можуть переглядати інформацію про роботу викладачів, а адміністратори мають доступ до керування користувачами та нормативами часу.

Основою навігації системи є головне меню, через яке здійснюється перехід між основними функціональними модулями. Структура меню побудована відповідно до логіки виконання основних бізнес-процесів та забезпечує швидкий доступ до необхідних сторінок. Усі функціональні можливості згруповані за призначенням, що спрощує пошук потрібних

інструментів та зменшує кількість переходів між сторінками.

Важливим елементом інтерфейсу є сторінка авторизації користувачів. Її основним призначенням є перевірка облікових даних та надання доступу до системи. Під час проєктування даної сторінки особлива увага приділялася простоті введення даних та забезпеченню зрозумілого механізму повідомлення про помилки автентифікації.

Після входу до системи користувач переходить до робочого середовища електронного журналу. Центральне місце займають сторінки роботи з індивідуальними планами. Вони забезпечують перегляд наявних планів, створення нових записів та редагування вже існуючої інформації. Для представлення даних використовуються табличні форми, які забезпечують зручне відображення великого обсягу інформації та підтримують можливість швидкого пошуку необхідних записів.

Для внесення інформації про виконані види діяльності передбачено спеціалізовані форми введення даних. Під час їх проєктування було враховано необхідність мінімізації помилок користувача та спрощення процесу введення інформації. Для цього використовуються випадаючі списки, поля перевірки даних та автоматичне заповнення окремих параметрів на основі нормативів часу, що зберігаються в системі.

Окрему групу сторінок становлять інтерфейси керування нормативами часу та користувачами системи. Вони доступні лише користувачам із відповідними правами доступу та забезпечують виконання адміністративних функцій. При проєктуванні даних сторінок використовувався єдиний стиль оформлення та однакові принципи взаємодії з таблицями й формами, що забезпечує цілісність користувацького інтерфейсу.

Для підвищення зручності роботи користувачів у системі передбачено механізми валідації введених даних та відображення інформаційних повідомлень. У випадку виникнення помилок система інформує користувача про причину їх появи та надає можливість виправити некоректно введені дані.

Це дозволяє підвищити достовірність інформації та зменшити кількість помилкових записів у базі даних.

Важливою вимогою до інтерфейсу стала його адаптивність. Оскільки користувачі можуть працювати із системою на різних типах пристроїв, сторінки повинні коректно відображатися як на персональних комп'ютерах, так і на планшетах або ноутбуках з різними роздільними здатностями екрана. Адаптивна структура інтерфейсу забезпечує збереження функціональності системи незалежно від характеристик пристрою користувача.

Для забезпечення єдиного стилю оформлення всі сторінки електронного журналу реалізовано з використанням однакових принципів побудови інтерфейсу. Це стосується розташування меню, форм введення даних, таблиць, кнопок керування та інформаційних повідомлень. Використання уніфікованого дизайну сприяє швидкому освоєнню системи новими користувачами та підвищує загальну зручність роботи із програмним забезпеченням.

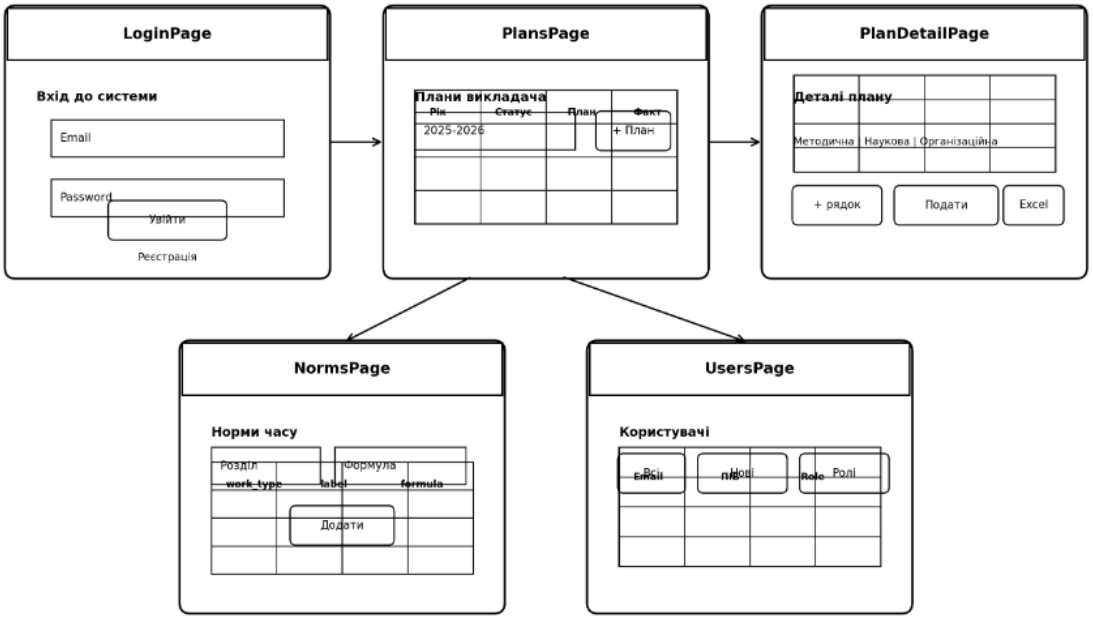


Рисунок 3.6. Прогнозовані макети основних сторінок застосунку

Макети основних сторінок інтерфейсу користувача наведені на рисунку 3.6. На них відображено структуру навігації, розташування функціональних елементів та особливості взаємодії користувача з основними модулями

системи.

Таким чином, спроєктований інтерфейс користувача забезпечує зручний доступ до функціональних можливостей електронного журналу, підтримує рольову модель доступу та створює комфортні умови для роботи викладачів, завідувачів кафедр і адміністраторів. Використання єдиних принципів організації сторінок, механізмів валідації даних та адаптивного дизайну сприяє підвищенню ефективності використання інформаційної системи в освітньому процесі.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Обґрунтування вибору технологій розроблення

Edu Planner являє собою вебзастосунок, призначений для автоматизації процесів планування та обліку діяльності науково-педагогічних працівників. Система реалізує електронний аналог документа «Індивідуальний план роботи науково-педагогічного працівника» (форма № Н-4.04) та забезпечує перехід від використання електронних таблиць Excel до централізованого веборієнтованого середовища. Функціональні можливості системи включають розмежування прав доступу між категоріями користувачів, автоматизований розрахунок обсягу виконаних робіт у годинах, а також формування звітної документації у форматі Excel.

Основною метою функціонування системи є автоматизація обліку навчального навантаження та інших видів діяльності викладачів закладу вищої освіти. Водночас система забезпечує підвищення прозорості процесів планування та контролю виконання індивідуальних планів, надаючи завідувачу кафедри інструменти моніторингу результатів роботи науково-педагогічних працівників. Для адміністратора передбачено механізм централізованого керування нормативами часу, що дозволяє змінювати правила розрахунку показників без внесення змін до програмного коду. Крім того, система забезпечує автоматичне формування звітних документів відповідно до форми Н-4.04 у форматі Excel.

Розроблення програмного забезпечення здійснювалося із використанням ітеративної моделі життєвого циклу програмного продукту (Iterative Development). Реалізація функціональних можливостей виконувалася послідовними етапами, що охоплювали створення базової інфраструктури проєкту, розроблення моделей даних, реалізацію механізмів автентифікації та авторизації, впровадження бізнес-логіки, формування звітності, розроблення

клієнтської частини та подальше розгортання системи. Кожна ітерація завершувалася отриманням працездатного програмного компонента, що дозволяло виконувати тестування окремих функціональних можливостей та поступово розширювати функціонал інформаційної системи.

Ітерація	Зміст
1	Конфігурація, база даних, підключення PostgreSQL
2	Моделі даних: User, Plan, WorkEntry, WorkNorm
3	Авторизація: реєстрація, логін, JWT, ролі
4	CRUD норм часу (адміністратор)
5	Калькулятор годин (eval по формулах з БД)
6	CRUD планів та записів викладача
7	Генерація Excel-звіту (openpyxl)
8	React-фронтенд: сторінки, форми, валідація
9	Docker Compose, деплой, документація

Під час розроблення електронного журналу планування та звітності діяльності науково-педагогічних працівників особливу увагу було приділено вибору технологій, які забезпечують необхідний рівень продуктивності, масштабованості, безпеки та зручності подальшого супроводу програмного забезпечення. Оскільки система передбачає роботу з великою кількістю структурованих даних, підтримку рольової моделі доступу, автоматичне формування звітності та взаємодію між клієнтською і серверною частинами, технологічний стек повинен був забезпечити ефективну реалізацію всіх функціональних вимог.

Стек технологій

Backend

Технологія	Версія	Призначення
Python	3.11	Основна мова backend
FastAPI	0.111	REST API фреймворк, автодокументація (Swagger)
SQLAlchemy	2.0	ORM, робота з базою даних
Pydantic v2	2.x	Валідація даних, схеми запитів/відповідей
python-jose	3.3	Генерація та верифікація JWT токенів

passlib + bcrypt	1.7 / 4.0	Хешування паролів
openpyxl	3.1	Генерація Excel-файлів
psycopg2-binary	2.9	Драйвер PostgreSQL
uvicorn	0.29	ASGI сервер

Frontend

Технологія	Версія	Призначення
React	18.3	UI бібліотека
React Router v6	6.24	Клієнтська маршрутизація (SPA)
Axios	1.7	HTTP-клієнт, JWT інтерцептори
Vite	5.3	Збірник, dev-сервер з проксі
nginx	alpine	Веб-сервер для продакшн-збірки

База даних та інфраструктура

Технологія	Версія	Призначення
PostgreSQL	16	Основна реляційна БД (продакшн)
SQLite	—	БД для локальної розробки без Docker
Docker	24+	Контейнеризація сервісів
Docker Compose	2.x	Оркестрація контейнерів
pgAdmin 4	latest	Веб-інтерфейс адміністрування БД
ngrok	—	Тунелювання для публічного доступу

Деплой (продакшн)

Сервіс	Тарифний план	Призначення
Render.com	Free	Хостинг backend (Python) та frontend (Static Site)
Neon.tech	Free	Managed PostgreSQL (serverless)

Для реалізації серверної частини інформаційної системи було обрано мову програмування Python. Вибір даної мови зумовлений її широким поширенням у сфері веброзробки, наявністю великої кількості бібліотек та інструментів, а також високою швидкістю створення програмних рішень. Python характеризується лаконічним синтаксисом, що сприяє підвищенню читабельності програмного коду та спрощує його супровід. Для проєктів, пов’язаних з автоматизацією освітніх процесів, це є особливо важливим, оскільки подальший розвиток системи може здійснюватися різними розробниками.

Серед переваг Python слід відзначити простоту реалізації бізнес-логіки, високу швидкість розроблення та наявність розвиненої екосистеми. Крім того, мова добре інтегрується із сучасними вебфреймворками, системами керування базами даних та бібліотеками для формування звітності. До недоліків можна віднести дещо нижчу продуктивність порівняно з компільованими мовами програмування, зокрема C# або Java. Проте для інформаційної системи планування та звітності, де основне навантаження пов’язане з обробкою запитів та роботою з базою даних, дане обмеження не має критичного впливу на ефективність функціонування програмного забезпечення.

Для побудови серверної частини системи було використано фреймворк FastAPI. Даний інструмент є одним із сучасних засобів розроблення вебсервісів мовою Python та орієнтований на створення високопродуктивних REST API. В архітектурі розробленого електронного журналу FastAPI використовується для реалізації маршрутів автентифікації, керування користувачами, планами роботи, нормативами часу та формування звітності.

Основною перевагою FastAPI є висока швидкодія завдяки використанню асинхронної моделі виконання запитів. Крім того, фреймворк автоматично генерує інтерактивну документацію API, що значно спрощує тестування та супровід системи. Важливою перевагою є також тісна інтеграція з Pydantic та сучасними засобами валідації даних. Недоліком FastAPI можна вважати меншу кількість готових модулів порівняно з Django, однак для даного проєкту використання більш легкого та продуктивного рішення є цілком виправданим.

Для реалізації об’єктно-реляційного відображення було використано бібліотеку SQLAlchemy. Дана технологія забезпечує взаємодію між програмними моделями та реляційною базою даних, дозволяючи працювати з даними на рівні об’єктів Python. У межах проєкту SQLAlchemy використовується для створення моделей користувачів, планів роботи, записів діяльності та нормативів часу.

Використання ORM-підходу дозволяє значно спростити розроблення

програмного забезпечення, оскільки розробнику не потрібно формувати більшість SQL-запитів вручну. Крім того, SQLAlchemy забезпечує підтримку міграцій та підвищує переносимість програмного коду між різними системами керування базами даних. До недоліків можна віднести деяке зниження продуктивності порівняно з використанням чистих SQL-запитів, проте для інформаційних систем даного класу переваги ORM суттєво переважають можливі втрати швидкодії.

Важливою складовою серверної частини стала бібліотека Pydantic, яка використовується для валідації даних та опису схем обміну інформацією між клієнтом і сервером. У системі Pydantic забезпечує перевірку коректності отриманих запитів та формування структурованих відповідей REST API.

Основною перевагою використання Pydantic є автоматична перевірка типів даних, що дозволяє виявляти помилки ще на етапі отримання запиту. Це сприяє підвищенню надійності програмного забезпечення та зменшує ймовірність виникнення помилок під час виконання бізнес-логіки. Недоліком можна вважати додаткові витрати ресурсів на валідацію великих обсягів даних, однак у межах розробленої системи цей фактор практично не впливає на продуктивність.

Для зберігання даних було обрано систему керування базами даних PostgreSQL. Дана СКБД є одним із найбільш поширених рішень для створення корпоративних інформаційних систем та характеризується високою надійністю, продуктивністю та відповідністю стандартам SQL.

Використання PostgreSQL дозволяє ефективно працювати зі структурованими даними, забезпечувати цілісність інформації та підтримувати складні зв’язки між таблицями. Для електронного журналу це є важливим, оскільки система містить взаємопов’язані сутності користувачів, планів роботи, записів діяльності та нормативів часу. Серед переваг PostgreSQL слід відзначити підтримку транзакцій, механізмів резервного копіювання та високий рівень безпеки. Недоліком є дещо складніше адміністрування порівняно з

деякими вбудованими базами даних, проте для багатокористувацької системи такі особливості є виправданими.

Клієнтська частина інформаційної системи реалізована з використанням бібліотеки React. Даний інструмент дозволяє створювати сучасні односторінкові вебзастосунки та забезпечує високий рівень інтерактивності користувацького інтерфейсу.

Використання React у межах проєкту дозволило реалізувати модульну структуру інтерфейсу та забезпечити повторне використання окремих компонентів. Завдяки механізму Virtual DOM суттєво зменшується кількість операцій оновлення сторінки, що позитивно впливає на швидкодію застосунку. Серед недоліків можна виділити необхідність додаткового вивчення екосистеми React та використання сторонніх бібліотек для реалізації окремих функцій. Однак отримані переваги значно перевищують складність початкового налаштування.

Для організації навігації між сторінками застосунку використано React Router. Даний інструмент забезпечує маршрутизацію всередині клієнтської частини системи та дозволяє створювати багатосторінкову логіку роботи без повного перезавантаження вебсторінок.

Використання React Router дозволяє забезпечити зручний перехід між сторінками планів роботи, звітності, користувачів та нормативів часу. Крім того, маршрутизація використовується для реалізації механізмів обмеження доступу до окремих сторінок залежно від ролі користувача. Недоліки даної бібліотеки є незначними та переважно стосуються складності налаштування великих ієрархій маршрутів.

Для взаємодії клієнтської частини з REST API використано бібліотеку Axios. Вона забезпечує виконання HTTP-запитів та обробку відповідей сервера.

Перевагами Axios є простота використання, підтримка перехоплювачів запитів та відповідей, а також можливість централізованої обробки помилок. У розробленій системі це дозволило реалізувати передачу JWT-токенів та

організувати єдиний механізм обробки помилок автентифікації. До недоліків можна віднести необхідність додаткового налаштування окремих сценаріїв роботи із запитами, проте це не створює суттєвих труднощів під час розроблення.

Для забезпечення безпеки системи використано механізм автентифікації на основі JSON Web Token. JWT дозволяє реалізувати безстанну модель автентифікації та забезпечує захищену передачу інформації про користувача між клієнтом і сервером.

Перевагою такого підходу є висока масштабованість та відсутність необхідності зберігати інформацію про сесії на сервері. Це особливо актуально для веборієнтованих інформаційних систем із REST-архітектурою. Для захисту паролів користувачів використовується бібліотека bcrypt, яка забезпечує хешування облікових даних перед їх збереженням у базі даних. Недоліком JWT є необхідність контролю терміну дії токенів та реалізації додаткових механізмів відкликання доступу у випадку компрометації токена.

Однією з функціональних особливостей електронного журналу є автоматичне формування звітності. Для реалізації цієї можливості використовується бібліотека openpyxl, яка забезпечує створення та редагування файлів Microsoft Excel.

Використання openpyxl дозволяє автоматизувати процес формування звітів без необхідності встановлення додаткового програмного забезпечення на сервері. Перевагою даної бібліотеки є підтримка форматування таблиць, роботи з формулами та створення складних структур документів. До недоліків можна віднести певні обмеження продуктивності під час формування дуже великих файлів, однак для обсягів даних, характерних для електронного журналу викладача, ці обмеження не є суттєвими.

Для контейнеризації програмного забезпечення використано Docker та Docker Compose. Дані технології забезпечують ізольоване виконання окремих компонентів системи та спрощують процес розгортання застосунку.

Контейнеризація дозволяє гарантувати однаковість програмного середовища на етапах розроблення, тестування та експлуатації системи. Використання Docker Compose додатково спрощує запуск взаємопов’язаних сервісів, зокрема серверної частини, клієнтського інтерфейсу та бази даних. Недоліком такого підходу є додаткове споживання системних ресурсів, однак переваги у вигляді спрощення розгортання та супроводу повністю компенсують даний фактор.

Для хмарного розгортання системи використано платформу Render та хмарну базу даних Neon. Таке рішення дозволяє забезпечити доступність програмного забезпечення через мережу Інтернет без необхідності підтримки власної серверної інфраструктури.

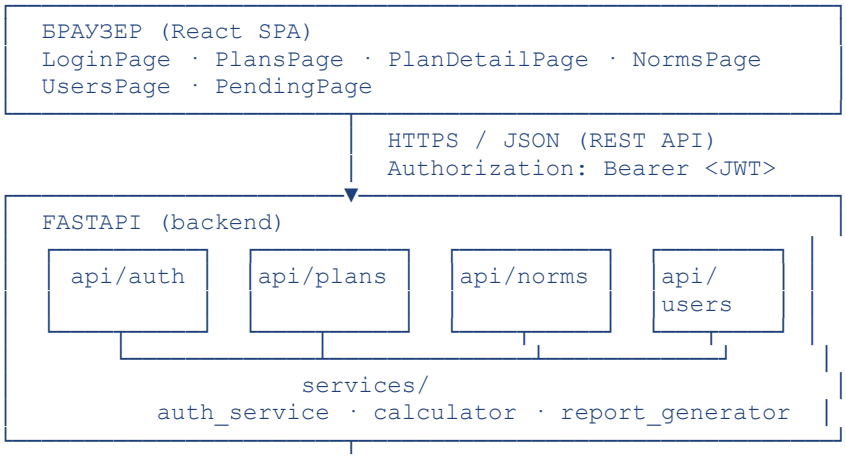
Використання Render забезпечує автоматичне розгортання нових версій програмного забезпечення та спрощує процес адміністрування серверної частини. У свою чергу, Neon надає масштабовану хмарну інфраструктуру для PostgreSQL. Недоліком використання хмарних сервісів є залежність від сторонніх постачальників послуг та необхідність постійного підключення до мережі Інтернет.

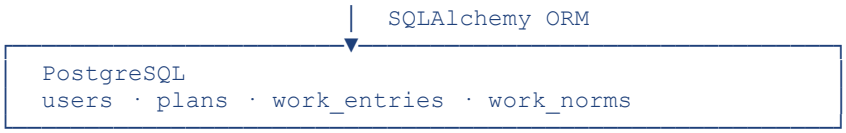
Отже, обраний технологічний стек повністю відповідає вимогам розроблюваної інформаційної системи та забезпечує реалізацію всіх необхідних функціональних можливостей. Поєднання FastAPI, SQLAlchemy, PostgreSQL та React дозволило реалізувати сучасну клієнт-серверну архітектуру з чітким розподілом відповідальності між компонентами. Використання JWT, bcrypt та механізмів рольового доступу забезпечує необхідний рівень безпеки, а застосування openpyxl дозволяє автоматизувати формування звітної документації. Docker, Render та Neon спрощують процес розгортання і супроводу системи. Сукупність обраних технологій створює надійну основу для подальшого розвитку електронного журналу планування та звітності діяльності науково-педагогічних працівників.

4.2. Реалізація серверної частини інформаційної системи

Серверна частина інформаційної системи Edu Planner реалізує основну бізнес-логіку електронного журналу планування та звітності діяльності науково-педагогічних працівників. Її основним призначенням є забезпечення обробки запитів користувачів, взаємодії з базою даних, виконання розрахунків навчального навантаження, контролю доступу до ресурсів системи та формування звітної документації. У межах розробленого програмного забезпечення серверна частина побудована за клієнт-серверною архітектурою та функціонує незалежно від інтерфейсу користувача.

Архітектура реалізованої системи складається з трьох основних рівнів: клієнтського рівня, серверного рівня та рівня зберігання даних. Клієнтська частина реалізована засобами React та відповідає за взаємодію користувача із системою. Серверна частина побудована на основі FastAPI та виконує обробку HTTP-запитів, перевірку прав доступу, реалізацію бізнес-логіки та формування відповідей у форматі JSON. Зберігання інформації здійснюється у базі даних PostgreSQL, яка містить відомості про користувачів, індивідуальні плани, записи діяльності, нормативи часу та сформовані звіти. Схема взаємодії наведена нижче:





Під час реалізації серверної частини було використано принцип модульної організації програмного коду. Такий підхід дозволяє відокремити різні функціональні компоненти системи та забезпечує можливість їх незалежного розвитку й супроводу. Структура серверного застосунку включає модулі конфігурації, маршрутизації, моделей даних, схем обміну інформацією, сервісів та допоміжних компонентів.

Шари backend (деталі)

Шар	Директорія	Відповідальність
API Layer	app/api/	Прийом HTTP-запитів, перевірка JWT, делегування сервісам
Service Layer	app/services/	Бізнес-логіка: розрахунок годин, генерація звітів, хешування
Data Layer	app/models/	SQLAlchemy-моделі, маппінг таблиць БД
Schema Layer	app/schemas/	Pydantic-схеми: валідація вхідних даних та серіалізація відповідей

Ролі та права доступу

Роль	Реєстрація	Плани	Норми	Користувачі	Звіти
new_user	Автоматично	Немає доступу	Немає	Немає	Немає
teacher	Адмін видає	Свої плани (CRUD)	Перегляд	Немає	Свої
head	Адмін видає	Перегляд усіх, підпис	Перегляд	Немає	Будь-чий
admin	Вручну в БД	Перегляд усіх	Повний CRUD	Повний CRUD	Будь-чий

Центральним елементом серверної частини є застосунок FastAPI, який забезпечує приймання та обробку HTTP-запитів. Для кожної функціональної підсистеми реалізовано окремі маршрути API, що відповідають за виконання конкретних операцій. Такий підхід забезпечує логічний поділ програмного коду та спрощує підтримку системи під час її подальшого розвитку.

Для реалізації доступу до даних використано окремий рівень моделей. Кожна сутність предметної області представлена відповідним класом SQLAlchemy, який описує структуру таблиці бази даних та зв'язки між

об'єктами. До основних моделей системи належать користувачі, індивідуальні плани роботи, записи про виконану діяльність та нормативи часу. Використання ORM-підходу дозволило реалізувати взаємодію з базою даних на рівні програмних об'єктів без необхідності створення великої кількості SQL-запитів вручну.

Для передачі даних між клієнтом і сервером використовуються схеми Pydantic. Вони забезпечують валідацію отриманих даних та формують структуру відповідей API. Застосування даного підходу дозволяє контролювати коректність інформації ще до моменту виконання бізнес-логіки та зменшує ймовірність виникнення помилок під час роботи системи.

Важливим компонентом серверної частини є модуль автентифікації та авторизації користувачів. Його реалізація базується на використанні JSON Web Token. Після успішного проходження процедури входу користувач отримує токен доступу, який використовується під час подальшої взаємодії із системою. Кожен захищений маршрут виконує перевірку токена та визначає права користувача перед виконанням відповідної операції.

Для зберігання паролів використовується механізм хешування на основі бібліотеки bcrypt. У базі даних зберігаються не самі паролі користувачів, а їх захищені хеш-значення. Такий підхід дозволяє підвищити рівень безпеки інформаційної системи та мінімізувати ризик компрометації облікових даних.

Функціональні можливості системи реалізовано за допомогою окремих сервісних компонентів. Вони відповідають за виконання бізнес-логіки та дозволяють відокремити алгоритми обробки даних від механізмів взаємодії з API. Одним із таких компонентів є сервіс авторизації, який виконує перевірку облікових даних та формування JWT-токенів. Окремий сервіс використовується для автоматичного розрахунку кількості годин відповідно до нормативів часу та параметрів виконаних робіт.

Механізм авторизації (JWT)

Автентифікація побудована на JSON Web Tokens (RFC 7519). Після успішного логіну сервер повертає access_token. Клієнт зберігає токен у localStorage і передає його в заголовку кожного запиту: Authorization: Bearer <token>.

- Алгоритм підпису: HS256
- Термін дії токена: 60 хвилин (налаштовується через .env)
- Паролі: bcrypt хеш (cost factor 12)
- new_user блокується на рівні залежності get_approved_user — до будь-якого захищеного ресурсу

Особливістю реалізованої серверної частини є підтримка автоматизованого механізму розрахунку навантаження. Після внесення користувачем інформації про виконану діяльність система виконує необхідні обчислення на основі встановлених нормативів та автоматично визначає кількість годин за відповідним видом роботи. Це дозволяє мінімізувати вплив людського фактора та підвищити точність формування індивідуальних планів і звітів.

Калькулятор годин

Норми часу зберігаються в таблиці work_norms як рядки формул. При додаванні або зміні запису плану автоматично викликається recalculate_plan(), яка обчислює години через безпечний eval:

```
def calculate_hours(formula: str, params: dict) -> float:
    allowed = {"__builtins__": {}}
    allowed.update(params)
    return float(eval(formula, allowed))

# Приклад:
# formula = "кількість * 20"
# params  = {"кількість": 3}
# result  = 60.0 год.
```

Однією з ключових складових серверної частини інформаційної системи є моделі даних, реалізовані за допомогою SQLAlchemy. Вони забезпечують відображення об'єктів предметної області на структуру реляційної бази даних та використовуються під час виконання всіх операцій зі збереження, отримання, оновлення та видалення інформації. Застосування ORM-підходу дозволило забезпечити тісну інтеграцію між програмним кодом і базою даних, спростити реалізацію бізнес-логіки та підвищити читабельність програмного

коду.

Центральною сутністю системи є модель користувача (User), яка використовується для зберігання інформації про облікові записи. Вона містить відомості про ім'я користувача, адресу електронної пошти, пароль у вигляді захищеного хешу, роль у системі та службові параметри, необхідні для організації процесів автентифікації та авторизації. Саме через дану модель реалізується механізм рольового доступу, який забезпечує розмежування функціональних можливостей між викладачами, завідувачами кафедр та адміністраторами.

Таблиця users

Поле	Тип	Опис
id	SERIAL PK	Унікальний ідентифікатор
email	VARCHAR(255)	Email (унікальний, використовується для входу)
full_name	VARCHAR(255)	Повне ім'я викладача
hashed_password	VARCHAR(255)	bcrypt хеш пароля
role	VARCHAR(20)	Роль: new_user | teacher | head | admin
is_active	BOOLEAN	Чи активний акаунт
created_at	TIMESTAMP	Дата реєстрації

Для представлення індивідуальних планів роботи використовується модель Plan. Вона містить інформацію про навчальний рік, поточний статус плану, загальну кількість запланованих і фактично виконаних годин, а також посилання на користувача, якому належить відповідний план. Кожен викладач може мати декілька планів для різних навчальних періодів, тому між моделями User та Plan реалізовано зв'язок типу «один до багатьох».

Таблиця plans

Поле	Тип	Опис
id	SERIAL PK	Унікальний ідентифікатор
teacher_id	FK → users	Власник плану
academic_year	VARCHAR(9)	Навчальний рік (напр. 2024-2025)
status	VARCHAR(20)	draft | submitted | approved
total_hours_planned	INTEGER	Сума запланованих годин
total_hours_actual	INTEGER	Сума фактичних годин
created_at	TIMESTAMP	Дата створення
updated_at	TIMESTAMP	Дата останнього оновлення

Відомості про окремі види діяльності зберігаються у моделі WorkEntry. Кожен запис містить інформацію про категорію роботи, її тип, параметри розрахунку, заплановану та фактично виконану кількість годин, а також додаткові коментарі. Дана модель використовується для накопичення інформації про навчальну, наукову, методичну та організаційну діяльність викладача. Зв'язок між моделями Plan та WorkEntry також реалізовано за схемою «один до багатьох», оскільки один індивідуальний план може містити значну кількість окремих записів діяльності.

Таблиця work_entries

Поле	Тип	Опис
id	SERIAL PK	Унікальний ідентифікатор
plan_id	FK → plans	Прив'язка до плану
section	VARCHAR(50)	Розділ: методична | наукова | організаційна
work_type	VARCHAR(100)	Тип роботи (ключ норми)
param_value	FLOAT	Значення параметру для формули
hours_planned	FLOAT	Розрахована кількість годин
hours_actual	FLOAT	Фактично виконано годин
sort_order	INTEGER	Порядок відображення
note	TEXT	Примітка викладача

Таблиця work_norms

Поле	Тип	Опис
id	SERIAL PK	Унікальний ідентифікатор
section	VARCHAR(50)	Розділ плану
work_type	VARCHAR(100)	Унікальний ключ виду роботи
label	VARCHAR(255)	Назва для відображення в UI
param_name	VARCHAR(100)	Назва змінної у формулі
formula	TEXT	Арифметична формула (напр. кількість * 20)
updated_at	TIMESTAMP	Дата останнього оновлення

Зв'язки між таблицями

```
users ──< plans ──< work_entries
                ↑
work_norms.work_type → work_entries.work_type  (логічний
зв'язок, не FK)

Логіка: FK не використовується щоб норми можна було редагувати
        незалежно від вже введених даних викладачами.
```

Окреме місце в структурі бази даних займає модель WorkNorm, призначена для зберігання нормативів часу. Вона містить відомості про

категорію роботи, її назву, параметри розрахунку та відповідні формули визначення трудомісткості. Нормативи використовуються під час автоматичного обчислення кількості годин та можуть редагуватися адміністраторами без внесення змін до програмного коду. Такий підхід забезпечує гнучкість системи та спрощує її адаптацію до змін нормативної документації закладу вищої освіти.

Зв'язки між моделями даних забезпечують цілісність інформації та коректність виконання операцій над даними. Використання механізмів SQLAlchemy дозволяє автоматизувати роботу з реляційними зв'язками, виконувати каскадні операції та формувати складні вибірки без необхідності написання великої кількості SQL-запитів вручну.

Структуру основних модулів серверної частини та зв'язки між ними представлено у вигляді UML-діаграми класів, наведеної на рисунку 4.2. На діаграмі відображено основні сутності системи, їх атрибути та взаємозв'язки, що забезпечують реалізацію функціональних можливостей електронного журналу планування та звітності.

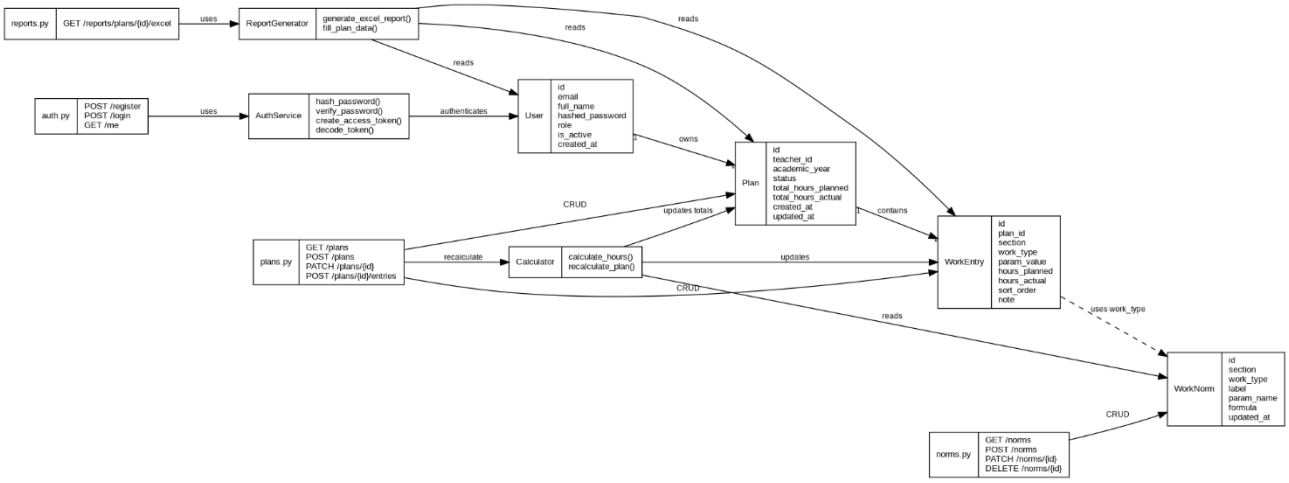


Рисунок 4.2. Діаграма класів серверної частини застосунку

Для генерації звітної документації реалізовано окремий модуль формування звітів. Його робота базується на використанні бібліотеки openpyxl, яка забезпечує створення файлів Microsoft Excel. На основі даних, що

зберігаються у базі даних, система автоматично формує звітні документи відповідно до структури індивідуального плану роботи науково-педагогічного працівника. Згенерований файл може бути завантажений користувачем для подальшого використання або подання до відповідних структурних підрозділів закладу освіти.

Окрему увагу під час реалізації серверної частини приділено забезпеченню рольової моделі доступу. У системі передбачено три категорії користувачів: викладач, завідувач кафедри та адміністратор. Для кожної ролі визначено перелік доступних операцій. Викладач має можливість працювати лише з власними планами та записами діяльності. Завідувач кафедри може переглядати інформацію про підпорядкованих працівників та контролювати виконання планів. Адміністратор отримує повний доступ до керування користувачами, нормативами часу та іншими системними параметрами.

Для взаємодії з базою даних використовується окремий механізм керування підключеннями. Це дозволяє забезпечити ефективне використання ресурсів сервера та підтримувати стабільну роботу системи за наявності декількох одночасних користувачів. Усі операції читання та запису даних виконуються через відповідні сесії SQLAlchemy із використанням механізмів транзакційності PostgreSQL.

З метою спрощення процесу розгортання серверна частина системи контейнеризована за допомогою Docker. Використання контейнерної технології дозволяє забезпечити однакове програмне середовище на етапах розроблення, тестування та експлуатації системи. Крім того, це спрощує перенесення застосунку між різними серверами та забезпечує його стабільну роботу незалежно від конфігурації операційної системи.

Таким чином, реалізована серверна частина інформаційної системи забезпечує виконання всіх основних функціональних можливостей електронного журналу планування та звітності діяльності науково-педагогічних працівників. Використання FastAPI, SQLAlchemy, PostgreSQL та допоміжних

сервісних компонентів дозволило побудувати модульну архітектуру програмного забезпечення, яка забезпечує ефективну обробку даних, підтримку рольового доступу, автоматизацію розрахунків та формування звітної документації. Отримане програмне рішення характеризується достатнім рівнем надійності, масштабованості та придатності до подальшого розвитку.

4.3. Реалізація клієнтської частини електронного журналу

Клієнтська частина інформаційної системи Edu Planner забезпечує взаємодію користувачів із функціональними можливостями електронного журналу та виконує роль програмного інтерфейсу між користувачем і серверною частиною застосунку. Основним завданням клієнтської частини є отримання даних від сервера, їх відображення у зручному для сприйняття вигляді, а також передавання введеної користувачем інформації до серверної частини для подальшої обробки.

Реалізація клієнтської частини виконана із використанням бібліотеки React, яка дозволяє створювати сучасні односторінкові вебзастосунки та забезпечує ефективне оновлення інтерфейсу без необхідності повного перезавантаження сторінок. Використання компонентного підходу дозволило розділити інтерфейс на незалежні функціональні елементи, кожен із яких відповідає за виконання окремої задачі. Така структура спрощує супровід програмного коду та створює можливості для подальшого розширення функціональності системи.

Архітектура клієнтської частини побудована відповідно до принципів Single Page Application (SPA). Усі основні операції виконуються без перезавантаження сторінки, а взаємодія із сервером здійснюється через REST API. Такий підхід забезпечує швидку реакцію інтерфейсу на дії користувача та підвищує зручність роботи із системою.

Для організації навігації між сторінками застосунку використовується бібліотека React Router. Вона забезпечує маршрутизацію всередині клієнтської частини та дозволяє реалізувати окремі сторінки для різних функціональних модулів системи. Після успішної авторизації користувач отримує доступ до сторінок відповідно до своєї ролі, а перехід між розділами здійснюється без повторного завантаження вебзастосунку.

Важливим елементом клієнтської частини є сторінка авторизації користувачів. Вона забезпечує введення облікових даних, передачу їх на сервер та отримання JWT-токена доступу. Після успішного проходження процедури автентифікації отриманий токен зберігається на стороні клієнта та використовується під час подальшого виконання запитів до захищених ресурсів системи.

Основу інтерфейсу користувача становлять сторінки роботи з індивідуальними планами. Вони забезпечують перегляд створених планів, створення нових записів та редагування наявної інформації. Дані відображаються у вигляді таблиць та форм введення, що дозволяє користувачам швидко знаходити необхідну інформацію та виконувати операції над нею. Для кожного плану передбачено можливість перегляду детальної інформації та внесення змін до окремих видів діяльності.

Для реалізації обліку діяльності викладача використано спеціалізовані форми введення даних. Користувач може додавати записи щодо навчальної, наукової, методичної та організаційної роботи, після чого відповідна інформація передається до серверної частини системи. Інтерфейс форм розроблено таким чином, щоб мінімізувати кількість помилок під час введення даних та забезпечити зручність роботи із великим обсягом інформації.

Окремий функціональний модуль клієнтської частини реалізує роботу з нормативами часу. Доступ до нього мають лише користувачі з відповідними правами доступу. Через даний інтерфейс адміністратор може переглядати перелік нормативів, змінювати їх параметри та додавати нові записи.

Застосування окремого інтерфейсу для роботи з нормативами дозволяє змінювати правила розрахунку навантаження без необхідності внесення змін до програмного коду серверної частини.

Для керування обліковими записами реалізовано сторінку адміністрування користувачів. Вона забезпечує перегляд інформації про користувачів системи, створення нових облікових записів та зміну ролей. Використання централізованого механізму керування користувачами спрощує адміністрування системи та підвищує рівень контролю за доступом до інформаційних ресурсів.

Взаємодія клієнтської частини із сервером реалізована за допомогою бібліотеки Axios. Усі запити до REST API виконуються через централізований механізм обробки HTTP-запитів. Використання Axios дозволило реалізувати автоматичну передачу JWT-токенів у заголовках запитів, централізовану обробку помилок та уніфікований механізм отримання даних від сервера.

Особливу увагу під час реалізації клієнтської частини було приділено забезпеченню зручності користувацького інтерфейсу. Для цього використовуються адаптивні елементи відображення даних, логічна структура навігації та єдині принципи побудови сторінок. Усі форми введення містять механізми перевірки даних, які дозволяють своєчасно повідомляти користувача про виявлені помилки та запобігати передаванню некоректної інформації до серверної частини.

Завдяки використанню React та сучасних підходів до розроблення вебінтерфейсів вдалося реалізувати клієнтську частину, яка забезпечує зручну взаємодію користувачів із системою та підтримує всі функціональні можливості електронного журналу. Загальну структуру клієнтської частини та взаємозв'язок основних сторінок доцільно подати у вигляді діаграми навігації.

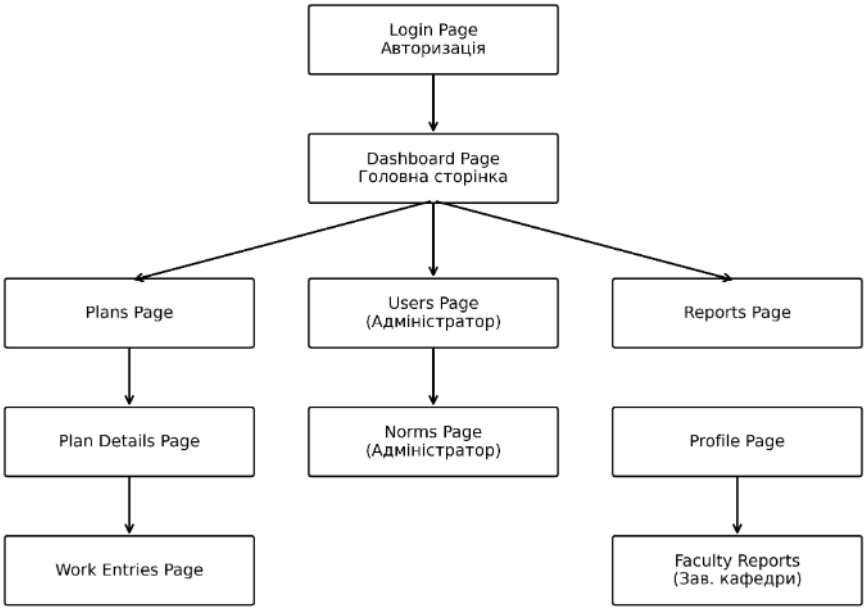


Рисунок 4.3. Діаграма навігації клієнтської частини інформаційної системи Edu Planner

На рисунку 4.3 наведено структуру навігації клієнтської частини системи. Після успішної автентифікації користувач отримує доступ до функціональних модулів відповідно до своєї ролі. Центральним елементом навігації є головна сторінка системи, з якої здійснюється перехід до модулів роботи з індивідуальними планами, звітністю, профілем користувача та адміністративними функціями. Такий підхід забезпечує логічну організацію інтерфейсу та швидкий доступ до основних функціональних можливостей електронного журналу.

Таким чином, реалізована клієнтська частина інформаційної системи забезпечує ефективну взаємодію користувачів із функціональними можливостями електронного журналу, підтримує рольову модель доступу та створює зручне середовище для планування, обліку та контролю діяльності науково-педагогічних працівників.

4.4. Реалізація механізму авторизації та розмежування доступу

Однією з ключових складових інформаційної системи Edu Planner є механізм авторизації та розмежування прав доступу користувачів. Його призначення полягає у забезпеченні захищеного доступу до функціональних можливостей системи відповідно до ролі користувача. У реалізованому програмному рішенні використовується підхід, заснований на технології JSON Web Token (JWT), яка забезпечує автентифікацію користувачів та контроль доступу до ресурсів системи. Особливістю реалізації є поєднання JWT-автентифікації з рольовою моделлю доступу, що дозволяє обмежувати виконання окремих операцій залежно від статусу користувача.

Відповідно до проєктних вимог у системі реалізовано чотири ролі користувачів: new_user**,** teacher**,** head та admin. Після проходження реєстрації користувач автоматично отримує роль new_user, яка не надає доступу до основних функцій електронного журналу. Надалі адміністратор може змінити роль користувача на викладача або завідувача кафедри, після чого стають доступними відповідні функціональні можливості системи. Такий підхід забезпечує додатковий контроль над процесом реєстрації та запобігає несанкціонованому доступу до інформації.

Для зберігання паролів використовується алгоритм хешування bcrypt, реалізований за допомогою бібліотеки Passlib. У сервісі авторизації передбачено окремі функції для створення хешу пароля та його подальшої перевірки під час входу до системи.

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


def hash_password(password: str) -> str:
    return pwd_context.hash(password)


def verify_password(plain: str, hashed: str) -> bool:
    return pwd_context.verify(plain, hashed)
```

Після успішного проходження процедури автентифікації сервер формує JWT-токен, який містить ідентифікатор користувача, його роль та час завершення дії токена. Отриманий токен передається клієнтській частині та надалі використовується під час звернення до захищених ресурсів системи.

```
def create_access_token(user_id: int, role: str) -> str:
    expire = datetime.now(timezone.utc) + timedelta(
        minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES
    )
    payload = {"sub": str(user_id), "role": role, "exp": expire}
    return jwt.encode(payload, settings.SECRET_KEY,
algorithm=settings.ALGORITHM)
```

Для перевірки справжності токена використовується процедура його декодування. Під час обробки кожного запиту сервер отримує інформацію про користувача із JWT та формує об'єкт, який надалі використовується механізмом авторизації.

```
def decode_token(token: str) -> TokenData:
    payload = jwt.decode(token, settings.SECRET_KEY,
algorithms=[settings.ALGORITHM])
    user_id = int(payload["sub"])
    role = payload["role"]
    return TokenData(user_id=user_id, role=role)
```

Реєстрація нового користувача реалізована через REST API. Під час створення облікового запису виконується перевірка наявності користувача з відповідною адресою електронної пошти, після чого пароль автоматично хешується та зберігається у базі даних.

```
@router.post("/register", response_model=UserOut,
status_code=status.HTTP_201_CREATED)
def register(data: UserRegister, db: Session = Depends(get_db)):
    if get_user_by_email(db, data.email):
        raise HTTPException(status_code=400, detail="Email вже
зареєстровано")

    user = User(
        email=data.email,
        full_name=data.full_name,
        hashed_password=hash_password(data.password),
    )
    db.add(user)
    db.commit()
    db.refresh(user)
```

```
    return user
```

Вхід до системи здійснюється шляхом перевірки введених користувачем облікових даних. У разі успішної перевірки сервер створює JWT-токен та повертає його клієнту.

```
@router.post("/login", response_model=Token)
def login(data: UserLogin, db: Session = Depends(get_db)):
    user = authenticate_user(db, data.email, data.password)
    if not user:
        raise HTTPException(status_code=401, detail="Невірний email або
пароль")
    if not user.is_active:
        raise HTTPException(status_code=403, detail="Акаунт деактивовано")

    token = create_access_token(user.id, user.role)
    return Token(access_token=token)
```

Для отримання інформації про поточного користувача та перевірки дійсності токена використовується залежність FastAPI `get_current_user`. Вона автоматично виконується перед зверненням до захищених ендпоінтів та забезпечує ідентифікацію користувача.

```
def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(bearer),
    db: Session = Depends(get_db),
) -> User:
    try:
        token_data = decode_token(credentials.credentials)
    except JWTError:
        raise HTTPException(status_code=401, detail="Невалідний або
прострочений токен")

    user = db.get(User, token_data.user_id)
    if not user or not user.is_active:
        raise HTTPException(status_code=401, detail="Користувача не
знайдено")
    return user
```

Особливістю реалізованого механізму є блокування нових користувачів до моменту підтвердження адміністратором. Для цього використовується окрема функція перевірки ролі.

```
def get_approved_user(current_user: User = Depends(get_current_user)) ->
User:
    if current_user.role == "new_user":
```

```
            raise HTTPException(
                status_code=403,
                detail="Акаунт очікує підтвердження адміністратором",
            )
        return current_user
```

Розмежування прав доступу реалізовано за допомогою спеціальної фабрики залежностей `require_role`. Вона дозволяє задавати перелік ролей, яким дозволено виконувати певну операцію. Якщо роль поточного користувача не входить до визначеного списку, сервер повертає повідомлення про відсутність необхідних прав доступу.

```
def require_role(*roles: str):
    def dependency(current_user: User = Depends(get_approved_user)) ->
User:
        if current_user.role not in roles:
            raise HTTPException(status_code=403, detail="Недостатньо
прав")
        return current_user
    return dependency
```

Застосування залежностей FastAPI дозволяє централізувати логіку перевірки прав доступу та уникнути дублювання програмного коду в окремих ендпоінтах. У результаті механізм авторизації та рольового доступу реалізований як окремий рівень безпеки серверної частини системи.

Реалізований механізм забезпечує захист облікових даних користувачів, контроль доступу до функціональних модулів системи та підтримку рольової моделі роботи електронного журналу. Використання JWT-токенів дозволяє ефективно організувати взаємодію між клієнтською та серверною частинами застосунку, а застосування ролей забезпечує виконання вимог щодо розмежування доступу між викладачами, завідувачами кафедр та адміністраторами системи. Даний підхід підвищує безпеку програмного забезпечення та забезпечує коректне функціонування електронного журналу в умовах багатокористувацького середовища.

4.5. Реалізація модулів планування, обліку годин і формування звітності

Основне функціональне призначення інформаційної системи Edu Planner полягає в автоматизації процесів планування індивідуальної роботи науково-педагогічних працівників, обліку виконаних видів діяльності та формуванні звітної документації відповідно до вимог закладу вищої освіти. Для реалізації зазначених можливостей у системі розроблено комплекс взаємопов'язаних модулів, які забезпечують створення індивідуальних планів, автоматичний розрахунок годин та генерацію звітів у форматі Excel.

Модуль планування реалізовано на основі сутності Plan, яка використовується для збереження індивідуального плану викладача на відповідний навчальний рік. Кожен план містить інформацію про власника, навчальний рік, статус документа та сумарну кількість запланованих і фактично виконаних годин. Передбачено підтримку кількох станів життєвого циклу плану: draft, submitted та approved, що дозволяє організувати процес погодження та затвердження документів у межах кафедри.

Для створення та керування планами використовується набір REST-ендпоінтів модуля plans, який забезпечує операції створення, перегляду, оновлення та видалення планів. Відповідний модуль дозволяє викладачеві створювати нові плани, вносити зміни до їх структури та подавати їх на перевірку завідувачу кафедри. Усі операції виконуються через серверну частину FastAPI із подальшим збереженням інформації у базі даних PostgreSQL.

Основною структурною одиницею плану є запис про виконання певного виду діяльності, який представлено сутністю WorkEntry. Кожен запис містить інформацію про розділ плану, тип роботи, значення параметра для обчислення нормативу, кількість запланованих і фактично виконаних годин, а також службову інформацію для впорядкування та відображення записів.

Використання окремої сутності дозволяє гнучко формувати структуру індивідуального плану та підтримувати різні види діяльності без зміни програмного коду системи.

Під час реалізації модуля обліку годин використано механізм нормативів часу, які зберігаються у таблиці work_norms. Для кожного виду роботи адміністратор може визначити назву, параметр розрахунку та математичну формулу, що використовується для автоматичного обчислення годин. Такий підхід дозволяє адаптувати систему до змін нормативної документації без внесення змін до програмної реалізації серверної частини.

Ключовим елементом модуля обліку є сервіс calculator, який забезпечує автоматичний розрахунок трудомісткості виконуваних робіт. Після створення або редагування запису плану система автоматично виконує перерахунок відповідних показників. У технічній документації наведено реалізацію функції обчислення кількості годин на основі формули, що зберігається у базі даних.

```
def calculate_hours(formula: str, params: dict) -> float:
    allowed = {"__builtins__": {}}
    allowed.update(params)
    return float(eval(formula, allowed))
```

Застосування такого підходу дозволяє реалізувати універсальний механізм розрахунку нормативів для різних видів діяльності. Наприклад, для наукової роботи формула може враховувати кількість публікацій, для методичної роботи — кількість підготовлених матеріалів, а для організаційної діяльності — обсяг виконаних організаційних заходів. При цьому сам алгоритм розрахунку залишається незмінним, а змінюються лише значення формул, що зберігаються у базі даних.

Важливою складовою системи є механізм автоматичного перерахунку сумарних показників плану. Після внесення змін до окремих записів система викликає процедуру перерахунку, яка оновлює загальну кількість планових та фактичних годин. Завдяки цьому користувач завжди працює з актуальними даними, а керівництво кафедри має можливість оперативно контролювати стан виконання індивідуальних планів.

Окремий функціональний модуль призначений для формування звітної документації. Його реалізація базується на сервісі report_generator, який використовує бібліотеку OpenPyXL для автоматичного створення файлів Microsoft Excel. Формування звіту виконується через спеціальний REST-ендпоінт:

```
GET /reports/plans/{id}/excel
```

Після надходження запиту система отримує дані про план, його записи та інформацію про викладача, після чого автоматично заповнює шаблон звітного документа. Сформований файл передається користувачеві для завантаження у форматі Excel. Такий підхід дозволяє автоматизувати підготовку документів форми Н-4.04 та суттєво зменшити витрати часу на ручне оформлення звітності.

Особливістю реалізованого механізму є підтримка рольової моделі доступу до звітів. Викладач має можливість завантажувати лише власні звіти, тоді як завідувач кафедри та адміністратор можуть формувати звітність для будь-якого викладача, що забезпечує необхідний рівень контролю за виконанням індивідуальних планів.

Розроблені модулі планування, обліку годин та формування звітності функціонують як єдина підсистема інформаційного середовища Edu Planner. Їх спільна робота забезпечує автоматизацію процесів планування діяльності науково-педагогічних працівників, оперативний контроль виконання робіт та швидке формування звітних документів відповідно до вимог освітнього процесу.

4.6. Тестування та розрогртання інформаційної системи

Після завершення реалізації серверної та клієнтської частин інформаційної системи Edu Planner було проведено комплексне тестування програмного забезпечення та виконано його підготовку до розгортання у

робочому середовищі. Основною метою тестування була перевірка коректності функціонування всіх компонентів системи, виявлення можливих помилок у взаємодії між модулями та підтвердження відповідності програмного продукту функціональним вимогам, сформованим на етапі проєктування.

Тестування проводилося поетапно та охоплювало перевірку серверної частини, клієнтського застосунку, бази даних, механізмів авторизації та рольового доступу, модулів планування й обліку діяльності, а також підсистеми формування звітної документації. Окрему увагу було приділено перевірці коректності взаємодії між компонентами системи через REST API та забезпеченню цілісності даних під час виконання основних операцій.

На початковому етапі було проведено тестування механізму реєстрації та авторизації користувачів. Перевірялися процеси створення нового облікового запису, входу до системи, формування JWT-токенів, перевірки дійсності токенів та завершення сеансу роботи. Результати тестування підтвердили коректне функціонування механізму автентифікації та відсутність помилок під час обробки облікових даних користувачів.

Окремо перевірявся механізм рольового доступу, реалізований для ролей new_user, teacher, head та admin. Для кожної ролі виконувалися сценарії доступу до різних функціональних модулів системи. Було підтверджено, що нові користувачі після реєстрації не отримують доступу до функціональних можливостей системи до моменту підтвердження адміністратором. Викладачі мають доступ лише до власних індивідуальних планів, завідувач кафедри може переглядати та затверджувати плани викладачів, а адміністратор отримує повний доступ до управління користувачами та нормативами часу.

Наступним етапом стало тестування модулів планування діяльності науково-педагогічних працівників. Перевірялися операції створення індивідуального плану, додавання записів до розділів методичної, наукової та організаційної роботи, редагування введених даних та зміни статусів плану. Особлива увага приділялася правильності відображення сумарної кількості

годин та коректності збереження інформації у базі даних.

Під час тестування підсистеми обліку годин перевірялася робота механізму автоматичного розрахунку трудомісткості на основі нормативів часу. Для різних видів діяльності виконувалося введення параметрів розрахунку та порівняння отриманих результатів із очікуваними значеннями. Результати перевірки підтвердили правильність роботи алгоритмів розрахунку та коректне оновлення сумарних показників індивідуального плану після внесення змін.

Важливою складовою тестування стало дослідження підсистеми формування звітної документації. Для цього створювалися тестові індивідуальні плани з різними наборами видів діяльності та різними значеннями нормативних показників. Після заповнення планів виконувалося автоматичне формування звітів у форматі Microsoft Excel. Отримані документи перевірялися на відповідність структурі форми Н-4.04, правильність перенесення даних із бази даних, коректність відображення планових та фактичних показників, а також наявність усіх необхідних реквізитів. У результаті проведених перевірок було встановлено, що сформовані файли повністю відповідають структурі індивідуального плану роботи науково-педагогічного працівника та можуть використовуватися в реальному освітньому процесі.

Окрім функціонального тестування, було виконано перевірку коректності роботи клієнтської частини системи. Перевірялися навігація між сторінками, робота форм введення даних, відображення повідомлень про помилки, механізми оновлення інформації без перезавантаження сторінки та коректність взаємодії з REST API. Особливу увагу приділено перевірці сценаріїв роботи для різних категорій користувачів. Результати тестування підтвердили правильність функціонування інтерфейсу та відповідність його структури проєктним рішенням.

Після завершення функціонального тестування було виконано розгортання інформаційної системи. Для забезпечення незалежності від

програмного середовища та спрощення процесу встановлення застосунку використано контейнерну технологію Docker. Такий підхід дозволяє запускати всі компоненти системи у стандартизованому середовищі незалежно від особливостей операційної системи та конфігурації комп'ютера. Далі подано коротку інструкцію для розгортання проекту.

Розгортання проекту

1. Вимоги

Компонент	Мінімальна версія
Docker Desktop	4.x (Mac/Windows) або Docker Engine 24+ (Linux)
Docker Compose	2.x (входить у Docker Desktop)
Git	2.x
ngrok (опційно)	3.x — для публічного доступу

2. Клонування репозиторію

Відкрийте термінал.

Перейдіть до бажаної директорії:

```
cd ~/Projects
```

Клонуйте репозиторій:

```
git clone https://github.com/YOUR_USER/edu-planner.git
cd edu-planner
```

3. Налаштування змінних оточення

Скопіюйте шаблон та відредагуйте:

```
cp .env.example .env   # якщо є шаблон
```

Вміст файлу .env для локального запуску:

```
DATABASE_URL=postgresql+psycopg2://edu:edu@db:5432/edu_planner
SECRET_KEY=замініть-на-довгий-випадковий-рядок
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=60
```

4. Запуск через Docker Compose

Зберіть та запустіть усі контейнери:

```
docker compose up --build
```

Дочекайтеся повідомлення у логах backend:

```
INFO:     Application startup complete.
```

Перевірте доступність сервісів:

Сервіс	URL
Frontend (застосунок)	http://localhost:5173
Backend API (Swagger)	http://localhost:8000/docs
pgAdmin (БД)	http://localhost:5050
PostgreSQL (прямий)	localhost:5432

5. Створення першого адміністратора

Виконайте скрипт у контейнері backend:

```
docker compose exec backend python create_admin.py
```

У консолі з'явиться повідомлення:

```
Створено: admin admin@edu.com / admin123
```

Увійдіть у застосунок з цими даними та змініть пароль.

6. Підключення pgAdmin до БД

Відкрийте http://localhost:5050
Логін: admin@admin.com, пароль: admin
Клікніть Add New Server та введіть:

Поле	Значення
Name	edu-planner
Host	db
Port	5432
Database	edu_planner
Username	edu
Password	edu

7. Зупинка проекту

```
docker compose down          # зупинити (дані збережено)
docker compose down -v       # зупинити + видалити базу даних
```

8. Публічний доступ через ngrok

Встановіть та авторизуйте ngrok (https://ngrok.com)
Запустіть тунель:

```
ngrok http 5173    # весь застосунок через nginx proxy
```

Скопіюйте URL виду https://xxxx.ngrok-free.app — надайте студентам або замовнику.

9. Деплой на Render + Neon (продакшн)

Створіть акаунт на neon.tech, новий проект, скопіюйте Connection String.
Зробіть push репозиторію на GitHub.
Зайдіть на render.com → New → Blueprint → підключіть репозиторій.
Render знайде render.yaml і автоматично створить два сервіси.
У сервісі edu-planner-api вставте змінну DATABASE_URL зі значенням з Neon.
Після деплою API скопіюйте його URL та вставте у VITE_API_URL frontend-сервісу.
Тригерніть повторний деплой frontend.

Для розгортання системи необхідна наявність Docker Desktop версії 4.x або Docker Engine версії 24 і вище, Docker Compose версії 2.x, а також системи контролю версій Git. У разі необхідності надання зовнішнього доступу до локально розгорнутого застосунку додатково може використовуватися сервіс ngrok. Використання зазначених програмних засобів дозволяє забезпечити швидке встановлення та запуск усіх компонентів інформаційної системи.

Після отримання вихідного коду з репозиторію GitHub виконується налаштування змінних середовища. Для цього використовується

конфігураційний файл .env, у якому визначаються параметри підключення до бази даних PostgreSQL, секретний ключ для генерації JWT-токенів, алгоритм шифрування та час життя токенів доступу. Застосування змінних середовища дозволяє відокремити конфігураційні параметри від програмного коду та забезпечує підвищення рівня безпеки системи.

У розробленому проєкті для локального запуску використовується підключення до бази даних PostgreSQL через параметр DATABASE_URL, що містить відомості про адресу сервера, назву бази даних та облікові дані користувача. Додатково задаються параметри SECRET_KEY, ALGORITHM та ACCESS_TOKEN_EXPIRE_MINUTES, які використовуються механізмом JWT-автентифікації. Такий підхід дозволяє легко змінювати параметри середовища без внесення змін до програмного коду застосунку.

Запуск інформаційної системи здійснюється за допомогою Docker Compose. Під час виконання команди запуску автоматично створюються всі необхідні контейнери та встановлюються зв'язки між ними. У результаті формується єдине середовище виконання, яке включає клієнтську частину на React, серверний застосунок FastAPI, систему керування базами даних PostgreSQL та засіб адміністрування pgAdmin.

Після успішного запуску користувачу стають доступними всі основні сервіси системи. Клієнтська частина функціонує через вебінтерфейс, серверна частина надає REST API та документацію Swagger, а база даних PostgreSQL забезпечує зберігання всієї службової та користувацької інформації. Наявність інтерактивної документації Swagger значно спрощує тестування REST API та дозволяє перевіряти роботу окремих ендпоїнтів без використання додаткових інструментів.

Після першого запуску виконується створення адміністративного облікового запису за допомогою спеціального службового сценарію create_admin.py. Автоматичне створення адміністратора дозволяє забезпечити початковий доступ до системи без необхідності безпосереднього втручання у

базу даних. Надалі адміністратор може створювати нових користувачів, змінювати їх ролі та виконувати загальне налаштування системи.

Для адміністрування бази даних використовується вебзастосунок pgAdmin. Після підключення до сервера PostgreSQL адміністратор отримує доступ до структури таблиць, даних користувачів, індивідуальних планів та нормативів часу. Використання pgAdmin значно спрощує супровід системи, дозволяє виконувати резервне копіювання та відновлення даних, а також здійснювати моніторинг роботи бази даних у процесі експлуатації.

Для демонстрації роботи системи поза межами локальної мережі передбачено використання сервісу ngrok. Даний інструмент створює захищений тунель між локальним сервером та мережею Інтернет, що дозволяє надавати доступ до вебзастосунку без його розміщення на окремому сервері. Такий підхід є особливо зручним під час демонстрації програмного забезпечення замовнику, проведення попереднього тестування або організації тимчасового доступу для потенційних користувачів системи. Після запуску тунелю система отримує публічну вебадресу, за якою клієнти можуть працювати із застосунком через звичайний браузер.

Для промислової експлуатації проєкту передбачено можливість розгортання у хмарному середовищі. У межах розробленого рішення використовується платформа Render для розміщення серверної та клієнтської частин застосунку, а також хмарна база даних PostgreSQL, що надається сервісом Neon. Такий підхід дозволяє забезпечити постійну доступність інформаційної системи через мережу Інтернет та значно спрощує процес адміністрування програмного забезпечення.

Процес хмарного розгортання передбачає створення проєкту бази даних у середовищі Neon та отримання рядка підключення до неї. Після цього вихідний код інформаційної системи розміщується у репозиторії GitHub та підключається до сервісу Render. Завдяки використанню конфігураційного файлу розгортання система автоматично створює необхідні сервіси та виконує

налаштування середовища виконання. Після вказання параметрів підключення до бази даних виконується автоматичне розгортання серверної частини застосунку.

Наступним етапом є налаштування клієнтської частини. Для цього у конфігураційних параметрах застосунку вказується адреса розгорнутого REST API, після чого виконується повторне розгортання інтерфейсу користувача. У результаті формується повністю працездатна хмарна інфраструктура, що забезпечує доступ до інформаційної системи незалежно від місця перебування користувача.

Після завершення технічного розгортання було проведено експлуатаційне тестування програмного забезпечення відповідно до ролей користувачів, реалізованих у системі. Для кожної категорії користувачів перевірялися типові сценарії роботи, що дозволило оцінити коректність функціонування програмного забезпечення в умовах, максимально наближених до реальної експлуатації.

Для ролі new_user перевірявся процес реєстрації нового користувача. Було встановлено, що після заповнення форми реєстрації система коректно створює новий обліковий запис та автоматично призначає йому роль нового користувача. Після входу до системи такий користувач отримує повідомлення про необхідність підтвердження облікового запису адміністратором і не має доступу до функціональних модулів електронного журналу. Це дозволяє запобігти несанкціонованому використанню системи особами, які не належать до навчального закладу.

Для ролі teacher перевірялися сценарії створення індивідуального плану, додавання записів діяльності, редагування планових та фактичних показників, подання плану на перевірку та формування звітів. Було підтверджено, що викладач має доступ виключно до власних планів та не може переглядати інформацію інших користувачів. Під час внесення записів система автоматично виконує розрахунок нормативних годин відповідно до встановлених правил та

оновлює підсумкові показники плану. Також успішно протестовано механізм експорту звітів у форматі Microsoft Excel.

Окрему увагу приділено перевірці ролі head, яка відповідає завідувачу кафедри. Після авторизації користувач даної ролі отримує доступ до індивідуальних планів усіх викладачів кафедри. Під час тестування було перевірено перегляд поданих планів, аналіз інформації про виконання робіт, затвердження документів та формування звітів для будь-якого викладача. Результати тестування підтвердили коректну роботу механізму погодження планів та зміну їх статусів відповідно до визначеного життєвого циклу документа.

Для ролі admin виконувалося тестування адміністративних функцій системи. Було перевірено процес зміни ролей користувачів, перегляд переліку зареєстрованих облікових записів, фільтрацію користувачів за ролями, створення нових нормативів часу та редагування існуючих правил розрахунку. Особлива увага приділялася перевірці механізму роботи з нормативами, оскільки саме вони використовуються під час автоматичного обчислення трудомісткості різних видів діяльності. Результати тестування підтвердили можливість гнучкого налаштування системи без необхідності внесення змін до програмного коду.

Під час перевірки модуля нормативів було встановлено, що адміністратор може створювати нові види робіт, задавати для них параметри розрахунку та визначати відповідні формули обчислення годин. Додатково перевірявся механізм попереднього перегляду результатів розрахунку, який дозволяє ще на етапі налаштування переконатися у правильності роботи формули. Такий підхід суттєво підвищує гнучкість інформаційної системи та забезпечує її адаптацію до змін нормативної документації закладу освіти.

Результати проведеного тестування показали стабільну роботу всіх функціональних модулів інформаційної системи. У процесі перевірки не було виявлено критичних помилок, які могли б впливати на працездатність або

безпеку програмного забезпечення. Реалізовані механізми авторизації, планування, обліку діяльності, формування звітності та адміністрування функціонують коректно та забезпечують виконання поставлених вимог.

Таким чином, проведені роботи з тестування та розгортання підтвердили готовність інформаційної системи Edu Planner до практичного використання в освітньому процесі. Застосування контейнерної технології Docker, сучасних засобів хмарного розгортання та рольової моделі доступу дозволило створити надійне програмне рішення, придатне для автоматизації процесів планування та звітності діяльності науково-педагогічних працівників закладів вищої освіти.

Після завершення технічного розгортання було проведено експлуатаційне тестування програмного забезпечення відповідно до ролей користувачів, реалізованих у системі. Для кожної категорії користувачів перевірялися типові сценарії роботи, що дозволило оцінити коректність функціонування програмного забезпечення в умовах, максимально наближених до реальної експлуатації.

Початковим етапом роботи із системою є реєстрація нового користувача. Для створення облікового запису необхідно перейти на сторінку реєстрації, ввести повне ім'я, адресу електронної пошти та пароль. Після надсилання форми система створює новий обліковий запис і автоматично призначає користувачу роль new_user. Інтерфейс сторінки реєстрації наведено на рисунку 4.8.

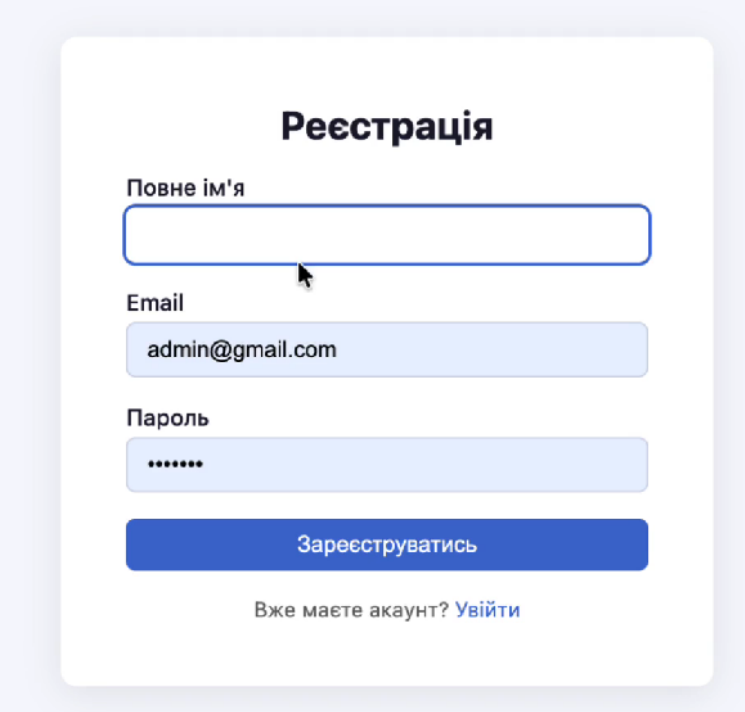


Рисунок 4.8. Сторінка реєстрації користувача

Після завершення процедури реєстрації користувач може виконати вхід до системи. Для цього використовується сторінка авторизації, на якій необхідно ввести адресу електронної пошти та пароль. Під час авторизації серверна частина виконує перевірку облікових даних та формує JWT-токен доступу. Після успішного входу користувач перенаправляється до головного інтерфейсу системи. Сторінку доадвання нового викладача наведено на рисунку 4.9.

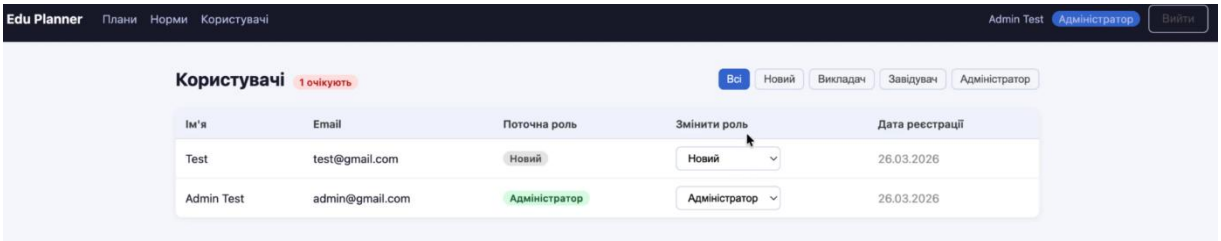


Рисунок 4.9. Сторінка додавання нового користувача

Користувачі, які щойно зареєструвалися, не мають доступу до функціональних можливостей електронного журналу до моменту підтвердження адміністратором. Після входу до системи їм відображається

повідомлення про необхідність очікування зміни ролі. Такий підхід забезпечує додатковий контроль над процесом реєстрації та дозволяє запобігти використанню системи сторонніми особами.

Після призначення ролі teacher користувач отримує доступ до функцій планування та звітності. Головною сторінкою для викладача є перелік індивідуальних планів, у якому відображаються навчальний рік, поточний статус документа, кількість запланованих та фактично виконаних годин. Сторінку перегляду планів наведено на рисунку 4.10.

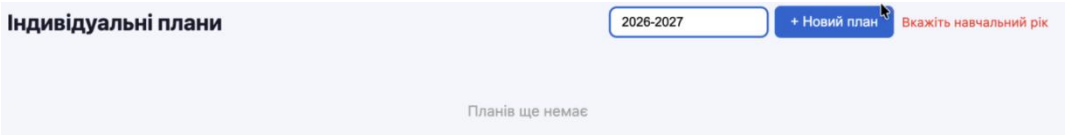


Рисунок 4.10. Список індивідуальних планів викладача

Для створення нового плану користувач вказує навчальний рік та виконує перехід до сторінки редагування документа. Після створення плану система автоматично формує його структуру та надає можливість заповнення розділів відповідно до видів діяльності. Інтерфейс сторінки редагування індивідуального плану наведено на рисунку 4.11.

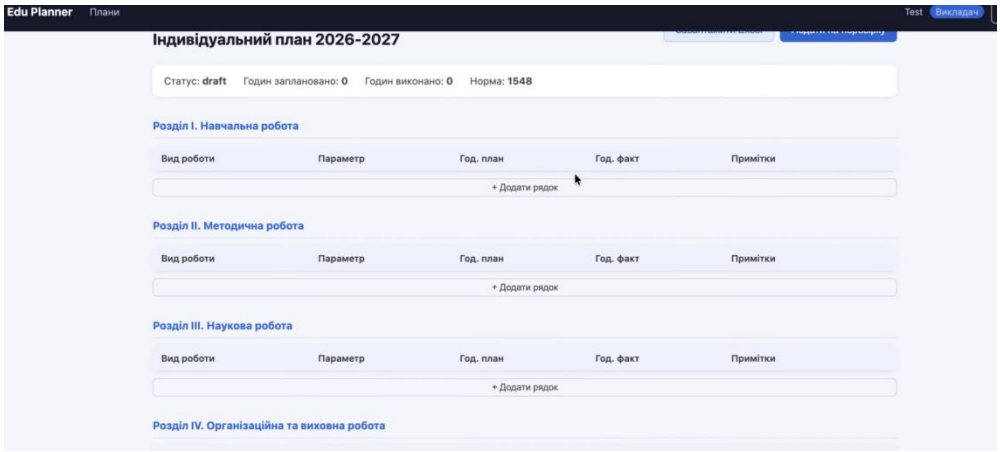


Рисунок 4.11. Сторінка редагування індивідуального плану

У межах індивідуального плану викладач може додавати записи щодо методичної, наукової та організаційної роботи. Для цього необхідно обрати

відповідний розділ, натиснути кнопку додавання нового запису та заповнити форму введення даних. Під час створення запису користувач обирає вид роботи зі списку нормативів, вводить значення параметра розрахунку та за потреби додає примітку. Після збереження система автоматично виконує розрахунок планових годин відповідно до встановлених нормативів. Вікно створення нового запису наведено на рисунку 4.12.

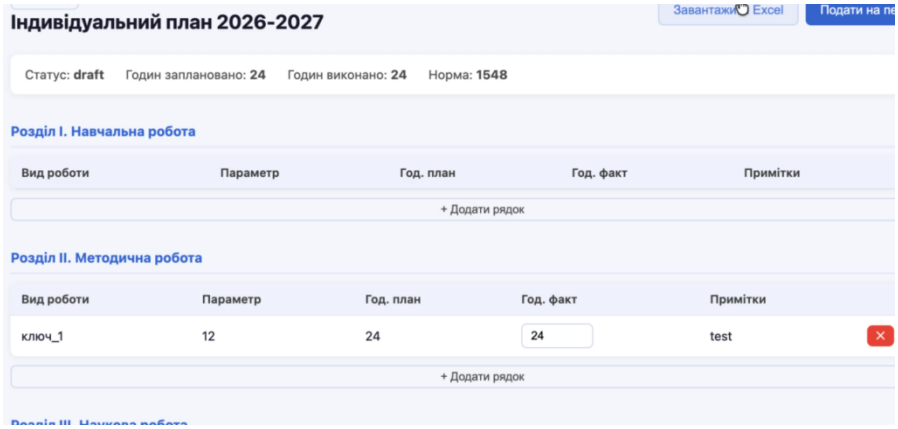


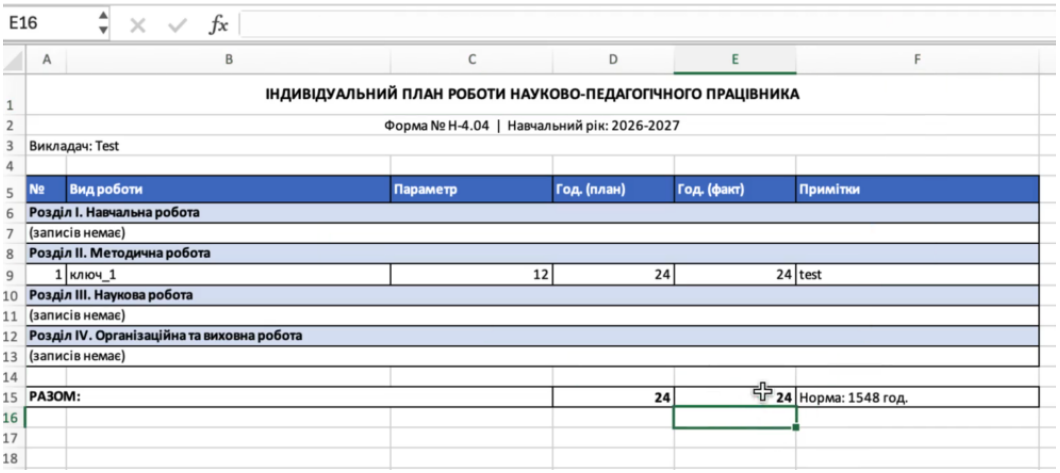
Рисунок 4.12. Додавання запису діяльності до індивідуального плану

Після внесення інформації про заплановану діяльність викладач має можливість фіксувати фактичне виконання робіт шляхом введення відповідних значень у колонку фактичних годин. Під час зміни даних система автоматично виконує перерахунок сумарних показників та оновлює інформацію про виконання плану. Такий механізм дозволяє оперативно контролювати ступінь виконання індивідуального навантаження протягом навчального року.

Після завершення заповнення документа викладач може подати план на перевірку. У результаті статус документа змінюється з «Чернетка» на «Подано». Після цього завідувач кафедри отримує можливість переглянути інформацію та прийняти рішення щодо затвердження документа.

Для формування звітної документації передбачено спеціальну функцію експорту даних у формат Microsoft Excel. Після натискання кнопки завантаження система автоматично генерує звітну форму Н-4.04 та зберігає її на комп'ютері користувача. Приклад сторінки формування звіту наведено на

рисунку 4.13.



ІНДИВІДУАЛЬНИЙ ПЛАН РОБОТИ НАУКОВО-ПЕДАГОГІЧНОГО ПРАЦІВНИКА

Форма № Н-4.04 | Навчальний рік: 2026-2027

Викладач: Test

№	Вид роботи	Параметр	Год. (план)	Год. (факт)	Примітки
Розділ I. Навчальна робота					
(записів немає)					
Розділ II. Методична робота					
1	ключ_1	12	24	24	test
Розділ III. Наукова робота					
(записів немає)					
Розділ IV. Організаційна та виховна робота					
(записів немає)					
РАЗОМ:			24	24	Норма: 1548 год.

Рисунок 4.13. Формування та завантаження Excel звіту

Користувач із роллю head виконує функції контролю та погодження індивідуальних планів. Після авторизації завідувач кафедри отримує доступ до переліку планів усіх викладачів кафедри. У таблиці відображаються відомості про навчальний рік, статус документа та підсумкові показники навантаження.

Після відкриття плану завідувач кафедри може переглянути внесені дані, проаналізувати структуру навантаження та оцінити відповідність документа встановленим вимогам. Якщо план заповнений коректно, користувач може виконати його затвердження. Після цього документ переводиться у стан «Затверджено» та стає доступним лише для перегляду. Сторінку погодження індивідуального плану наведено на рисунку 4.14.

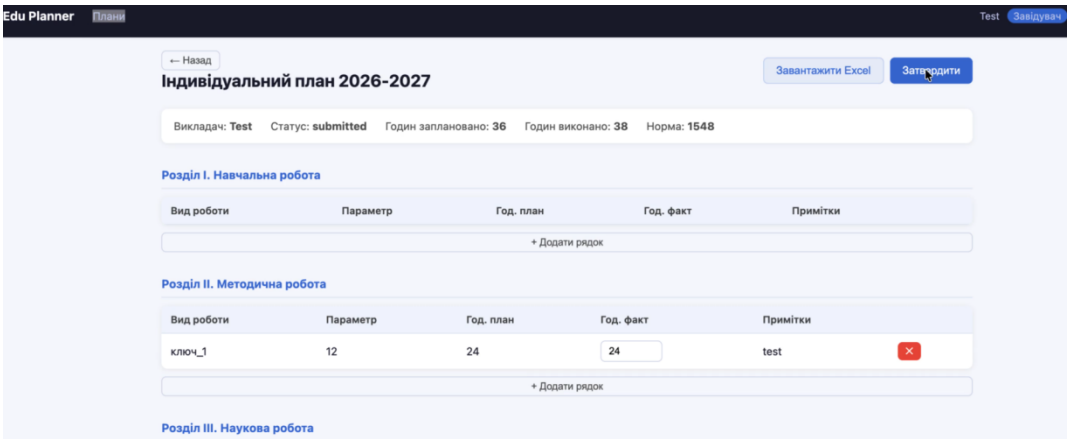


Рисунок 4.14. Затвердження індивідуального плану завідувачем кафедри

Користувач із роллю admin має доступ до функцій адміністрування інформаційної системи. Одним із основних модулів є підсистема керування користувачами, яка дозволяє переглядати список зареєстрованих осіб, змінювати їх ролі та контролювати нові реєстрації. Інтерфейс модуля керування індивідуальними планами користувачів наведено на рисунку 4.15.

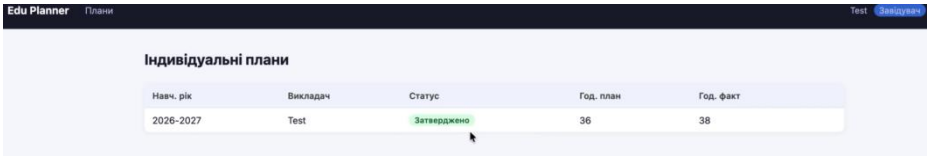


Рисунок 4.15. Модуль управління навчальними планами

Для спрощення адміністрування передбачено можливість фільтрації користувачів за ролями та оперативної зміни їх статусу без необхідності додаткового підтвердження. Це дозволяє швидко надавати доступ новим викладачам та керівникам структурних підрозділів.

Іншим важливим елементом адміністративної частини є модуль керування нормативами часу. Через даний інтерфейс адміністратор може створювати нові нормативи, змінювати існуючі формули розрахунку та налаштовувати параметри різних видів діяльності. Завдяки цьому система може адаптуватися до змін нормативної документації без внесення змін до програмного коду. Інтерфейс сторінки нормативів наведено на рисунку 4.16.

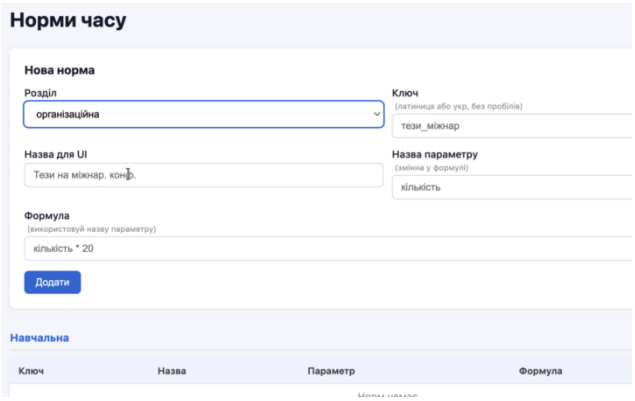


Рисунок 4.16. Модуль керування нормативами часу

Під час тестування адміністративного модуля було підтверджено коректність роботи механізмів створення нових нормативів, редагування формул розрахунку та автоматичного оновлення правил обчислення годин для всіх користувачів системи. Таким чином, реалізований інтерфейс забезпечує повний цикл роботи із нормативними даними та дозволяє підтримувати актуальність інформації без втручання розробника.

Проведене експлуатаційне тестування підтвердило коректність роботи всіх функціональних можливостей системи для кожної категорії користувачів. Реалізовані інтерфейси забезпечують зручну взаємодію із системою, а рольова модель доступу дозволяє ефективно розмежувати функціональні можливості між викладачами, завідувачами кафедр та адміністраторами.

ВИСНОВКИ

У результаті виконання дипломної роботи досягнуто поставленої мети, яка полягала у розробці інформаційної системи для планування та звітності в освітньому процесі. Розроблений програмний продукт являє собою електронний журнал, призначений для автоматизації процесів планування індивідуальної роботи науково-педагогічних працівників, обліку виконаних видів діяльності та формування звітної документації відповідно до вимог закладу вищої освіти.

У ході виконання роботи було проведено аналіз предметної області та досліджено існуючі інформаційні системи управління освітнім процесом. Розглянуто сучасні програмні рішення, що використовуються для організації навчального процесу, обліку діяльності викладачів та формування звітності. Особливу увагу приділено системі IZETA, яка використовується у Полтавському університеті економіки і торгівлі, а також іншим інформаційним системам, що застосовуються у закладах вищої освіти. Проведений аналіз дозволив визначити основні функціональні можливості, переваги та недоліки існуючих рішень, а також сформувати вимоги до розроблюваної системи.

На основі результатів аналізу було визначено функціональні вимоги до інформаційної системи. Передбачено підтримку ролей викладача, завідувача кафедри та адміністратора, можливість створення й редагування індивідуальних планів, автоматичного обліку годин, контролю виконання навантаження, формування звітів та адміністрування нормативів часу. Сформовані вимоги стали основою для подальшого проєктування програмного забезпечення.

У роботі виконано проєктування архітектури інформаційної системи, структури бази даних, ролей користувачів та функціональних модулів. Розроблено модель взаємодії компонентів програмного забезпечення, визначено основні сутності предметної області та зв'язки між ними. Також

спроєктовано REST API для забезпечення взаємодії між клієнтською та серверною частинами застосунку та розроблено структуру користувацького інтерфейсу.

Обґрунтовано вибір технологій розроблення програмного забезпечення. Для реалізації серверної частини використано мову програмування Python та фреймворк FastAPI, які забезпечують створення продуктивних вебсервісів і REST API. Для роботи з базою даних застосовано SQLAlchemy та PostgreSQL. Клієнтська частина реалізована з використанням React, що дозволило створити сучасний динамічний вебінтерфейс. Для контейнеризації та розгортання системи використано Docker і Docker Compose. Обраний стек технологій забезпечив необхідну функціональність, масштабованість та зручність подальшого супроводу програмного забезпечення.

У процесі реалізації програмного продукту розроблено серверну частину інформаційної системи, яка забезпечує обробку запитів користувачів, взаємодію з базою даних, управління індивідуальними планами та формування звітності. Реалізовано механізм авторизації та автентифікації на основі JWT-токенів, а також рольову модель доступу, що забезпечує розмежування функціональних можливостей між різними категоріями користувачів.

Створено клієнтську частину системи, яка забезпечує зручну взаємодію користувача з електронним журналом через веббраузер. Реалізовано інтерфейси для створення та редагування планів, обліку виконаних робіт, контролю нормативів часу, управління користувачами та формування звітних документів. Використання сучасних засобів веброзроблення дозволило створити зрозумілий та зручний інтерфейс для всіх категорій користувачів системи.

Особливу увагу приділено реалізації модулів планування діяльності, автоматичного розрахунку нормативних годин та формування звітності. У системі реалізовано механізм збереження нормативів часу в базі даних із можливістю їх подальшого редагування адміністратором без внесення змін до програмного коду. Також забезпечено автоматичне формування звітів у

форматі Microsoft Excel відповідно до форми Н-4.04, що значно спрощує процес підготовки звітної документації.

Проведено тестування розробленої інформаційної системи, у ході якого перевірено коректність роботи механізмів авторизації, планування, обліку діяльності, формування звітів та адміністрування системи. Результати тестування підтвердили працездатність усіх функціональних модулів та відповідність реалізованого програмного забезпечення поставленим вимогам. Додатково виконано розгортання системи з використанням контейнерної технології Docker та підготовлено програмний продукт до використання в локальному та хмарному середовищах.

За результатами виконання дипломної роботи опубліковано тези на XLIX Міжнародній науковій конференції студентів та аспірантів «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті» (м. Полтава, 23 квітня 2026 р.) [15].

Отже, у результаті виконання дипломної роботи створено повноцінну інформаційну систему для планування та звітності в освітньому процесі, яка забезпечує автоматизацію основних процесів обліку діяльності науково-педагогічних працівників, підвищує ефективність контролю виконання індивідуальних планів та спрощує формування звітної документації. Розроблене програмне забезпечення може бути використане як практичний інструмент для організації роботи кафедр та інших структурних підрозділів закладів вищої освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ateş, H. (2019). USING INFORMATION SYSTEMS AND TECHNOLOGIES IN HIGHER EDUCATION INSTITUTIONS. *International Journal of Research -GRANTHAALAYAH*, *7*(10), 222–232. https://doi.org/10.29121/granthaalayah.v7.i10.2019.390 – Назва з екрану.

2. Docker Compose Documentation. Режим доступу URL: https://docs.docker.com/compose/ – Назва з екрану.

3. Docker Documentation. Режим доступу URL: https://docs.docker.com/ – Назва з екрану.

4. FastAPI Documentation. Режим доступу URL: https://fastapi.tiangolo.com/ – Назва з екрану.

5. Jan-Peter Mund, Evelyn Wallor, Victoriia Khrutba, Angelina Chugai, Iryna Patseva4, and Yuliia Zakharova. Formation of a Digitalization Culture in Higher Education Institutions of Ukraine. Analysis of the current state. Режим доступу URL: https://ceur-ws.org/Vol-3966/W1Paper1.pdf – Назва з екрану.

6. Kravchenko, K., Kravchenko, T., Maiatina, N., Huda, O., & Lysetskyi, B. (2024). Digital Resources and Technologies for Improvement of Educational Process in Ukraine. *Futurity Education*, *4*(2), 4–28. https://doi.org/10.57125/FED.2024.06.25.01 – Назва з екрану.

7. Kryshtanovych, S., Bekh, Y., Stadnichenko, O., Shevchenko, Z., Maikher, V., & Liubinska, O. (2025). Education 4.0: Development of the Ukrainian Education System in the Context of Artificial Intelligence and Information Technologies. *Journal of Management World*, *2025*(2), 315-319. Режим доступу URL: https://doi.org/10.53935/jomw.v2024i4.927 Назва з екрану.

8. O. B. Morgulets and T. M. Derkach, “INFORMATION AND COMMUNICATION TECHNOLOGIES MANAGING THE QUALITY OF EDUCATIONAL ACTIVITIES OF A UNIVERSITY”, *ITLT*, vol. 71, no. 3, pp. 295–304, Jun. 2019, doi: 10.33407/itlt.v71i3.2831. – Назва з екрану.

9. PostgreSQL Documentation. Режим доступу URL: https://www.postgresql.org/docs/ – Назва з екрану.

10. Protsenko, V. (2025). THE MECHANISM FOR INTEGRATING EU DIGITAL STANDARDS INTO PUBLIC EDUCATION ADMINISTRATION IN UKRAINE. *Philosophy and Governance*, (12(16). Режим доступу URL: https://doi.org/10.70651/3041-248X/2025.12.01 – Назва з екрану.

11. React Documentation. Режим доступу URL: https://react.dev/ – Назва з екрану.

12. SQLAlchemy Documentation. Режим доступу URL: https://docs.sqlalchemy.org/ – Назва з екрану.

13. Двірна О. А., Вергал К. Ю., Іванов Ю. В. Автоматизована система управління закладом вищої освіти: управління розкладом та логістикою освітнього процесу // Системи управління, навігації та зв'язку. 2023. № 3(73). С. 86–92. Режим доступу URL: https://journals.nupp.edu.ua/sunz/uk/article/view/3055 – Назва з екрану.

14. Електронний журнал. Індивідуальний план роботи викладача КНЕУ ім. В.Гетьмана Режим доступу URL: https://kneu.edu.ua/ua/University_en/subdivisions/cioms/scc/cioms_vais/e_journal/ Назва з екрану

15. Електронний журнал успішності, ХУУП імені Леоніда Юзькова. Режим доступу URL: https://univer.km.ua/dystantsiyne-seredovyshche/studentu/elektronni-zhurnaly-obliku-uspishnosti – Назва з екрану

16. Інформаційна система «Електронний університет» Режим доступу URL: https://isu1.khmnu.edu.ua/isu/?utm_source=chatgpt.com – Назва з екрану

17. Кобель М.О. Розробка інформаційної системи для планування та звітності в освітньому процесі // Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті : тези доповідей XLIX Міжнародної наукової конференції студентів та аспірантів (м. Полтава, 23 квітня 2026 р.). –

Полтава : ПУЕТ, 2026. – С. 414-417. Режим доступу: https://puet.edu.ua/wp-content/uploads/2026/06/zb_tez-2026-qual-osv-xxi.pdf – Назва з екрану.

18. Кошова О. П. РОЗРОБКА ВЕБ-ДОДАТКІВ ТА СЕРВІСІВ НА ПЛАТФОРМІ NODE.JS / О. П. Кошова, О.В. Ольховська О.В., Д. С. Тацій, Ю.Ф. Олексійчук, О.О. Черненко // Таврійський науковий вісник. Серія: Технічні науки, 2023. Вип. 2. С. 78-89. Режим доступу: https://journals.ksauniv.ks.ua/index.php/tech/issue/view/26 <https://journals.ksauniv.ks.ua/index.php/tech/article/view/358/331> – Назва з екрану.

19. М. М. Косіюк, К. Е. Більовський, and В. М. Лисак, “АВТОМАТИЗОВАНА ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ЗАКЛАДОМ ВИЩОЇ ОСВІТИ «ЕЛЕКТРОННИЙ УНІВЕРСИТЕТ»”, *ITLT*, vol. 93, no. 1, pp. 96–116, Feb. 2023, doi: 10.33407/itlt.v93i1.5107. – Назва з екрану.

20. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава: ПУЕТ, 2024. – 67 с. Режим доступу: URL: http://dspace.puet.edu.ua/bitstream/123456789/14768/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0%20%D0%91%D0%A0%20%D0%B4%D0%BB%D1%8F%20%D0%9A%D0%9D%202024_%D0%B1%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80.pdf - Назва з екрану.

21. Положення про електронний журнал ХАІ. .. Режим доступу URL: https://khai.edu/polozenna-pro-elektronnij-zurnal-obliku-navcalnoi-roboti-studentiv-akademicnoi-grupi? – Назва з екрану.

22. Розвиток інформаційних систем управління освітою як інструмент реалізації державної освітньої політики : монографія / за ред. С. Л. Лондара ; ДНУ «Інститут освітньої аналітики». Київ, 2020. 258 с.. Режим доступу URL: https://iea.gov.ua/wp-content/uploads/2020/07/Rozvitok-IS-2020-monografiyaFINAL_sajt.pdf – Назва з екрану.

ДОДАТОК А
Код програми

EDU PLANNER - ВІДІБРАНІ ФРАГМЕНТИ КОДУ ДЛЯ ДИПЛОМНОЇ РОБОТИ
Код не змінювався. Включено лише ключові файли, що демонструють архітектуру та логіку системи.

```
================================================================================================
FILE: backend/main.py
================================================================================================

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.api.auth import router as auth_router
from app.api.norms import router as norms_router
from app.api.plans import router as plans_router
from app.api.reports import router as reports_router
from app.api.users import router as users_router
from database import Base, engine

# Імпорт моделей — потрібен щоб SQLAlchemy їх зареєстрував до create_all
import app.models.user  # noqa: F401
import app.models.norm  # noqa: F401
import app.models.plan  # noqa: F401
import app.models.work_entry  # noqa: F401

# Створюємо всі таблиці при старті (поки немає Alembic)
Base.metadata.create_all(bind=engine)


def _run_migrations():
    """
    Ручні міграції для змін що create_all не виконує (зміна типу колонки).
    Безпечно — кожен ALTER виконується тільки якщо потрібен.
    """
    with engine.begin() as conn:
        # Міграція: user_role ENUM → VARCHAR(20)
        result = conn.execute(
            text(
                "SELECT data_type FROM information_schema.columns "
                "WHERE table_name='users' AND column_name='role'"
            )
        ).fetchone()
        if result and result[0] != "character varying":
            conn.execute(text("ALTER TABLE users ALTER COLUMN role TYPE VARCHAR(20)"))
            conn.execute(text("ALTER TABLE users ALTER COLUMN role SET DEFAULT 'new_user'"))
            conn.execute(text("DROP TYPE IF EXISTS user_role"))

        # Міграція: додати teacher_name до plans якщо відсутній
        col = conn.execute(
            text(
                "SELECT column_name FROM information_schema.columns "
                "WHERE table_name='plans' AND column_name='teacher_name'"
            )
        ).fetchone()
        if not col:
            conn.execute(
                text("ALTER TABLE plans ADD COLUMN teacher_name VARCHAR(200) NOT NULL DEFAULT ''")
            )

        # Міграція: додати teacher_email до plans якщо відсутній
        col_email = conn.execute(
            text(
                "SELECT column_name FROM information_schema.columns "
                "WHERE table_name='plans' AND column_name='teacher_email'"
            )
        ).fetchone()
        if not col_email:
            conn.execute(
                text("ALTER TABLE plans ADD COLUMN teacher_email VARCHAR(200) NOT NULL DEFAULT ''")
            )


from sqlalchemy import text  # noqa: E402
_run_migrations()
```

```
app = FastAPI(
    title="Edu Planner API",
    description="Інформаційна система для планування та звітності в освітньому процесі",
    version="0.1.0",
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5173"],  # React dev server
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(auth_router, prefix="/api/v1")
app.include_router(norms_router, prefix="/api/v1")
app.include_router(plans_router, prefix="/api/v1")
app.include_router(reports_router, prefix="/api/v1")
app.include_router(users_router, prefix="/api/v1")


@app.get("/")
def root():
    return {"status": "ok", "message": "Edu Planner API працює"}


@app.get("/health")
def health():
    return {"status": "ok"}



================================================================================================
FILE: backend/database.py
================================================================================================

from sqlalchemy import create_engine
from sqlalchemy.orm import DeclarativeBase, sessionmaker

from config import settings

# Neon і стандартний PostgreSQL повертають URL з префіксом "postgres://",
# але SQLAlchemy 2.0 вимагає "postgresql+psycopg2://"
def _fix_url(url: str) -> str:
    if url.startswith("postgres://"):
        return url.replace("postgres://", "postgresql+psycopg2://", 1)
    return url


_url = _fix_url(settings.DATABASE_URL)
_is_sqlite = _url.startswith("sqlite")

engine = create_engine(
    _url,
    connect_args={"check_same_thread": False} if _is_sqlite else {},
)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)


class Base(DeclarativeBase):
    pass


def get_db():
    """Залежність FastAPI — повертає сесію БД і закриває після запиту."""
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()



================================================================================================
FILE: backend/app/models/user.py
```

```
================================================================================================

from datetime import datetime
from typing import TYPE_CHECKING

from sqlalchemy import DateTime, String, func
from sqlalchemy.orm import Mapped, mapped_column, relationship

from database import Base

if TYPE_CHECKING:
    from app.models.plan import Plan


class User(Base):
    __tablename__ = "users"

    id: Mapped[int] = mapped_column(primary_key=True, index=True)
    email: Mapped[str] = mapped_column(String(255), unique=True, index=True, nullable=False)
    full_name: Mapped[str] = mapped_column(String(255), nullable=False)
    hashed_password: Mapped[str] = mapped_column(String(255), nullable=False)
    role: Mapped[str] = mapped_column(
        String(20), default="new_user", nullable=False
    )  # new_user | teacher | head | admin
    is_active: Mapped[bool] = mapped_column(default=True)
    created_at: Mapped[datetime] = mapped_column(DateTime, server_default=func.now())

    plans: Mapped[list["Plan"]] = relationship("Plan", back_populates="teacher")



================================================================================================
FILE: backend/app/models/plan.py
================================================================================================

from datetime import datetime
from typing import TYPE_CHECKING

from sqlalchemy import DateTime, ForeignKey, Integer, String, func
from sqlalchemy.orm import Mapped, mapped_column, relationship

from database import Base

if TYPE_CHECKING:
    from app.models.user import User
    from app.models.work_entry import WorkEntry


class Plan(Base):
    __tablename__ = "plans"

    id: Mapped[int] = mapped_column(primary_key=True, index=True)
    teacher_id: Mapped[int] = mapped_column(ForeignKey("users.id"), nullable=False, index=True)
    teacher_name: Mapped[str] = mapped_column(String(200), default="", nullable=False)
    teacher_email: Mapped[str] = mapped_column(String(200), default="", nullable=False)
    academic_year: Mapped[str] = mapped_column(
        String(9), nullable=False
    )  # напр. '2024-2025'
    status: Mapped[str] = mapped_column(
        String(20), default="draft", nullable=False
    )  # 'draft', 'submitted', 'approved'
    total_hours_planned: Mapped[int] = mapped_column(Integer, default=0)
    total_hours_actual: Mapped[int] = mapped_column(Integer, default=0)
    created_at: Mapped[datetime] = mapped_column(DateTime, server_default=func.now())
    updated_at: Mapped[datetime] = mapped_column(
        DateTime, server_default=func.now(), onupdate=func.now()
    )

    teacher: Mapped["User"] = relationship("User", back_populates="plans")
    entries: Mapped[list["WorkEntry"]] = relationship(
        "WorkEntry", back_populates="plan", cascade="all, delete-orphan"
    )



================================================================================================
FILE: backend/app/models/work_entry.py
```

```
================================================================================================

from typing import TYPE_CHECKING

from sqlalchemy import Float, ForeignKey, Integer, String, Text
from sqlalchemy.orm import Mapped, mapped_column, relationship

from database import Base

if TYPE_CHECKING:
    from app.models.plan import Plan


class WorkEntry(Base):
    __tablename__ = "work_entries"

    id: Mapped[int] = mapped_column(primary_key=True, index=True)
    plan_id: Mapped[int] = mapped_column(ForeignKey("plans.id"), nullable=False, index=True)

    # Розділ відповідає формі Н-4.04
    section: Mapped[str] = mapped_column(
        String(50), nullable=False
    )  # 'навчальна', 'методична', 'наукова', 'організаційна'
    work_type: Mapped[str] = mapped_column(
        String(100), nullable=False
    )  # посилання на work_norms.work_type

    # Параметр для формули (кількість, годин_лекцій тощо)
    param_value: Mapped[float] = mapped_column(Float, default=0)

    # Розраховані години (заповнює calculator.py)
    hours_planned: Mapped[float] = mapped_column(Float, default=0)
    hours_actual: Mapped[float] = mapped_column(Float, default=0)

    # Порядковий номер рядка у розділі
    sort_order: Mapped[int] = mapped_column(Integer, default=0)

    # Примітка викладача (колонка «Примітки» у формі)
    note: Mapped[str | None] = mapped_column(Text, nullable=True)

    plan: Mapped["Plan"] = relationship("Plan", back_populates="entries")



================================================================================================
FILE: backend/app/models/norm.py
================================================================================================

from datetime import datetime

from sqlalchemy import DateTime, String, Text, func
from sqlalchemy.orm import Mapped, mapped_column

from database import Base


class WorkNorm(Base):
    __tablename__ = "work_norms"

    id: Mapped[int] = mapped_column(primary_key=True, index=True)
    section: Mapped[str] = mapped_column(
        String(50), nullable=False
    )  # 'навчальна', 'методична', 'наукова', 'організаційна'
    work_type: Mapped[str] = mapped_column(
        String(100), unique=True, index=True, nullable=False
    )  # унікальний ключ: 'тези_міжнар'
    label: Mapped[str] = mapped_column(String(255), nullable=False)  # назва для UI
    param_name: Mapped[str] = mapped_column(String(100), nullable=False)  # 'кількість'
    formula: Mapped[str] = mapped_column(Text, nullable=False)  # 'кількість * 20'
    updated_at: Mapped[datetime] = mapped_column(
        DateTime, server_default=func.now(), onupdate=func.now()
    )



================================================================================================
FILE: backend/app/services/auth_service.py
```

```
================================================================================================

from datetime import datetime, timedelta, timezone

from jose import JWTError, jwt
from passlib.context import CryptContext
from sqlalchemy.orm import Session

from app.models.user import User
from app.schemas.user import TokenData
from config import settings

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


def hash_password(password: str) -> str:
    return pwd_context.hash(password)


def verify_password(plain: str, hashed: str) -> bool:
    return pwd_context.verify(plain, hashed)


def create_access_token(user_id: int, role: str) -> str:
    expire = datetime.now(timezone.utc) + timedelta(
        minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES
    )
    payload = {"sub": str(user_id), "role": role, "exp": expire}
    return jwt.encode(payload, settings.SECRET_KEY, algorithm=settings.ALGORITHM)


def decode_token(token: str) -> TokenData:
    payload = jwt.decode(token, settings.SECRET_KEY, algorithms=[settings.ALGORITHM])
    user_id = int(payload["sub"])
    role = payload["role"]
    return TokenData(user_id=user_id, role=role)


def get_user_by_email(db: Session, email: str) -> User | None:
    return db.query(User).filter(User.email == email).first()


def authenticate_user(db: Session, email: str, password: str) -> User | None:
    user = get_user_by_email(db, email)
    if not user or not verify_password(password, user.hashed_password):
        return None
    return user



================================================================================================
FILE: backend/app/services/calculator.py
================================================================================================

from sqlalchemy.orm import Session

from app.models.norm import WorkNorm
from app.models.plan import Plan


def calculate_hours(formula: str, params: dict) -> float:
    """
    Безпечно виконує формулу з бази даних.
    formula: "кількість * 20"
    params:  {"кількість": 2}
    """
    allowed = {"__builtins__": {}}
    allowed.update(params)
    result = eval(formula, allowed)  # noqa: S307
    return float(result)


def recalculate_plan(plan: Plan, db: Session) -> None:
    """
    Перераховує години для всіх записів плану і оновлює підсумки.
    Викликається після будь-якої зміни в WorkEntry.
    """
```

```
    norm_map: dict[str, WorkNorm] = {
        n.work_type: n for n in db.query(WorkNorm).all()
    }

    total_planned = 0.0
    total_actual = 0.0

    for entry in plan.entries:
        norm = norm_map.get(entry.work_type)
        if norm:
            params = {norm.param_name: entry.param_value}
            hours = calculate_hours(norm.formula, params)
            entry.hours_planned = hours
        total_planned += entry.hours_planned
        total_actual += entry.hours_actual

    plan.total_hours_planned = int(total_planned)
    plan.total_hours_actual = int(total_actual)
    db.commit()



================================================================================================
FILE: backend/app/api/auth.py
================================================================================================

from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import HTTPAuthorizationCredentials, HTTPBearer
from jose import JWTError
from sqlalchemy.orm import Session

from app.models.user import User
from app.schemas.user import Token, UserOut, UserRegister, UserLogin
from app.services.auth_service import (
    authenticate_user,
    create_access_token,
    decode_token,
    get_user_by_email,
    hash_password,
)
from database import get_db

router = APIRouter(prefix="/auth", tags=["auth"])
bearer = HTTPBearer()


@router.post("/register", response_model=UserOut, status_code=status.HTTP_201_CREATED)
def register(data: UserRegister, db: Session = Depends(get_db)):
    if get_user_by_email(db, data.email):
        raise HTTPException(status_code=400, detail="Email вже зареєстровано")

    user = User(
        email=data.email,
        full_name=data.full_name,
        hashed_password=hash_password(data.password),
    )
    db.add(user)
    db.commit()
    db.refresh(user)
    return user


@router.post("/login", response_model=Token)
def login(data: UserLogin, db: Session = Depends(get_db)):
    user = authenticate_user(db, data.email, data.password)
    if not user:
        raise HTTPException(status_code=401, detail="Невірний email або пароль")
    if not user.is_active:
        raise HTTPException(status_code=403, detail="Акаунт деактивовано")

    token = create_access_token(user.id, user.role)
    return Token(access_token=token)


def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(bearer),
    db: Session = Depends(get_db),
```

```
) -> User:
    """Залежність FastAPI — декодує JWT і повертає поточного користувача."""
    try:
        token_data = decode_token(credentials.credentials)
    except JWTError:
        raise HTTPException(status_code=401, detail="Невалідний або прострочений токен")

    user = db.get(User, token_data.user_id)
    if not user or not user.is_active:
        raise HTTPException(status_code=401, detail="Користувача не знайдено")
    return user


def get_approved_user(current_user: User = Depends(get_current_user)) -> User:
    """Блокує new_user від доступу до захищених ресурсів."""
    if current_user.role == "new_user":
        raise HTTPException(
            status_code=403,
            detail="Акаунт очікує підтвердження адміністратором",
        )
    return current_user


def require_role(*roles: str):
    """Фабрика залежностей для перевірки ролі."""
    def dependency(current_user: User = Depends(get_approved_user)) -> User:
        if current_user.role not in roles:
            raise HTTPException(status_code=403, detail="Недостатньо прав")
        return current_user
    return dependency


@router.get("/me", response_model=UserOut)
def me(current_user: User = Depends(get_current_user)):
    return current_user



================================================================================================
FILE: backend/app/api/plans.py
================================================================================================

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session

from app.api.auth import get_approved_user as get_current_user, require_role
from app.models.plan import Plan
from app.models.user import User
from app.models.work_entry import WorkEntry
from app.schemas.plan import (
    PlanCreate,
    PlanOut,
    PlanSummary,
    PlanUpdate,
    WorkEntryCreate,
    WorkEntryOut,
    WorkEntryUpdate,
)
from app.services.calculator import recalculate_plan
from database import get_db

router = APIRouter(prefix="/plans", tags=["plans"])


# ---------------------------------------------------------------------------
# Плани
# ---------------------------------------------------------------------------

@router.get("/", response_model=list[PlanSummary])
def list_plans(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    """
    teacher    — бачить тільки свої плани
    head/admin — бачать плани всіх
    """
```

```
    if current_user.role == "teacher":
        return db.query(Plan).filter(Plan.teacher_id == current_user.id).all()
    return db.query(Plan).all()


@router.post("/", response_model=PlanOut, status_code=status.HTTP_201_CREATED)
def create_plan(
    data: PlanCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    exists = (
        db.query(Plan)
        .filter(Plan.teacher_id == current_user.id, Plan.academic_year == data.academic_year)
        .first()
    )
    if exists:
        raise HTTPException(status_code=400, detail="План на цей рік вже існує")

    plan = Plan(
        teacher_id=current_user.id,
        academic_year=data.academic_year,
        teacher_name=current_user.full_name,
        teacher_email=current_user.email,
    )
    db.add(plan)
    db.commit()
    db.refresh(plan)
    return plan


@router.get("/{plan_id}", response_model=PlanOut)
def get_plan(
    plan_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)
    return plan


@router.patch("/{plan_id}", response_model=PlanOut)
def update_plan(
    plan_id: int,
    data: PlanUpdate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)

    # Тільки head/admin можуть змінювати статус на 'approved'
    if data.status == "approved" and current_user.role == "teacher":
        raise HTTPException(status_code=403, detail="Недостатньо прав для підтвердження")

    for field, value in data.model_dump(exclude_none=True).items():
        setattr(plan, field, value)

    db.commit()
    db.refresh(plan)
    return plan


@router.delete("/{plan_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_plan(
    plan_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)
    if plan.status == "approved":
        raise HTTPException(status_code=400, detail="Не можна видалити затверджений план")
    db.delete(plan)
    db.commit()


# ---------------------------------------------------------------------------
# Записи плану (WorkEntry)
```

```
# ---------------------------------------------------------------------------

@router.post("/{plan_id}/entries", response_model=WorkEntryOut, status_code=status.HTTP_201_CREATED)
def add_entry(
    plan_id: int,
    data: WorkEntryCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)
    _check_editable(plan)

    entry = WorkEntry(plan_id=plan.id, **data.model_dump())
    db.add(entry)
    db.flush()  # отримуємо id до recalculate
    recalculate_plan(plan, db)
    db.refresh(entry)
    return entry


@router.patch("/{plan_id}/entries/{entry_id}", response_model=WorkEntryOut)
def update_entry(
    plan_id: int,
    entry_id: int,
    data: WorkEntryUpdate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)
    _check_editable(plan)

    entry = db.get(WorkEntry, entry_id)
    if not entry or entry.plan_id != plan.id:
        raise HTTPException(status_code=404, detail="Запис не знайдено")

    for field, value in data.model_dump(exclude_none=True).items():
        setattr(entry, field, value)

    recalculate_plan(plan, db)
    db.refresh(entry)
    return entry


@router.delete("/{plan_id}/entries/{entry_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_entry(
    plan_id: int,
    entry_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user),
):
    plan = _get_accessible_plan(plan_id, current_user, db)
    _check_editable(plan)

    entry = db.get(WorkEntry, entry_id)
    if not entry or entry.plan_id != plan.id:
        raise HTTPException(status_code=404, detail="Запис не знайдено")

    db.delete(entry)
    recalculate_plan(plan, db)


# ---------------------------------------------------------------------------
# Хелпери
# ---------------------------------------------------------------------------

def _get_accessible_plan(plan_id: int, user: User, db: Session) -> Plan:
    plan = db.get(Plan, plan_id)
    if not plan:
        raise HTTPException(status_code=404, detail="План не знайдено")
    if user.role == "teacher" and plan.teacher_id != user.id:
        raise HTTPException(status_code=403, detail="Доступ заборонено")
    return plan


def _check_editable(plan: Plan) -> None:
    if plan.status == "approved":
        raise HTTPException(status_code=400, detail="Затверджений план не можна редагувати")
```

```
================================================================================================
FILE: backend/app/api/norms.py
================================================================================================

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session

from app.api.auth import get_approved_user as get_current_user, require_role
from app.models.norm import WorkNorm
from app.schemas.norm import NormCreate, NormOut, NormUpdate
from database import get_db

router = APIRouter(prefix="/norms", tags=["norms"])


@router.get("/", response_model=list[NormOut])
def list_norms(
    section: str | None = None,
    db: Session = Depends(get_db),
    _=Depends(get_current_user),  # будь-який авторизований
):
    """Повертає всі норми. Можна фільтрувати за розділом: ?section=наукова"""
    q = db.query(WorkNorm)
    if section:
        q = q.filter(WorkNorm.section == section)
    return q.order_by(WorkNorm.section, WorkNorm.id).all()


@router.get("/{norm_id}", response_model=NormOut)
def get_norm(
    norm_id: int,
    db: Session = Depends(get_db),
    _=Depends(get_current_user),
):
    norm = db.get(WorkNorm, norm_id)
    if not norm:
        raise HTTPException(status_code=404, detail="Норму не знайдено")
    return norm


@router.post("/", response_model=NormOut, status_code=status.HTTP_201_CREATED)
def create_norm(
    data: NormCreate,
    db: Session = Depends(get_db),
    _=Depends(require_role("admin")),
):
    if db.query(WorkNorm).filter(WorkNorm.work_type == data.work_type).first():
        raise HTTPException(status_code=400, detail="work_type вже існує")

    norm = WorkNorm(**data.model_dump())
    db.add(norm)
    db.commit()
    db.refresh(norm)
    return norm


@router.patch("/{norm_id}", response_model=NormOut)
def update_norm(
    norm_id: int,
    data: NormUpdate,
    db: Session = Depends(get_db),
    _=Depends(require_role("admin")),
):
    norm = db.get(WorkNorm, norm_id)
    if not norm:
        raise HTTPException(status_code=404, detail="Норму не знайдено")

    for field, value in data.model_dump(exclude_none=True).items():
        setattr(norm, field, value)

    db.commit()
    db.refresh(norm)
    return norm
```

```
@router.delete("/{norm_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_norm(
    norm_id: int,
    db: Session = Depends(get_db),
    _=Depends(require_role("admin")),
):
    norm = db.get(WorkNorm, norm_id)
    if not norm:
        raise HTTPException(status_code=404, detail="Норму не знайдено")

    db.delete(norm)
    db.commit()



================================================================================================
FILE: backend/app/api/users.py
================================================================================================

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.api.auth import get_current_user, require_role
from app.models.user import User
from app.schemas.user import UserOut, UserRoleUpdate
from database import get_db

router = APIRouter(prefix="/users", tags=["users"])


@router.get("/", response_model=list[UserOut])
def list_users(
    db: Session = Depends(get_db),
    _=Depends(require_role("admin")),
):
    return db.query(User).order_by(User.created_at.desc()).all()


@router.get("/pending", response_model=list[UserOut])
def pending_users(
    db: Session = Depends(get_db),
    _=Depends(require_role("admin")),
):
    """Список користувачів, що очікують підтвердження."""
    return db.query(User).filter(User.role == "new_user").order_by(User.created_at).all()


@router.patch("/{user_id}/role", response_model=UserOut)
def update_role(
    user_id: int,
    data: UserRoleUpdate,
    db: Session = Depends(get_db),
    current_admin: User = Depends(require_role("admin")),
):
    user = db.get(User, user_id)
    if not user:
        raise HTTPException(status_code=404, detail="Користувача не знайдено")
    if user.id == current_admin.id:
        raise HTTPException(status_code=400, detail="Не можна змінити власну роль")

    user.role = data.role
    db.commit()
    db.refresh(user)
    return user



================================================================================================
FILE: backend/app/services/report_generator.py
================================================================================================

import io
from typing import IO

from openpyxl import Workbook
from openpyxl.styles import Alignment, Border, Font, PatternFill, Side
```

```
from openpyxl.utils import get_column_letter

from app.models.plan import Plan

SECTIONS = ("навчальна", "методична", "наукова", "організаційна")
SECTION_LABELS = {
    "навчальна": "Розділ I. Навчальна робота",
    "методична": "Розділ II. Методична робота",
    "наукова": "Розділ III. Наукова робота",
    "організаційна": "Розділ IV. Організаційна та виховна робота",
}

_HEADER_FILL = PatternFill("solid", fgColor="4472C4")
_SECTION_FILL = PatternFill("solid", fgColor="D9E1F2")
_THIN = Side(style="thin")
_BORDER = Border(left=_THIN, right=_THIN, top=_THIN, bottom=_THIN)


def _cell(ws, row: int, col: int, value, bold=False, fill=None, wrap=False):
    c = ws.cell(row=row, column=col, value=value)
    c.border = _BORDER
    c.alignment = Alignment(wrap_text=wrap, vertical="center")
    if bold:
        c.font = Font(bold=True)
    if fill:
        c.fill = fill
        c.font = Font(bold=True, color="FFFFFF" if fill == _HEADER_FILL else "000000")
    return c


def generate_excel(plan: Plan, teacher_name: str) -> IO[bytes]:
    wb = Workbook()
    ws = wb.active
    ws.title = "Індивідуальний план"

    # Ширина колонок
    widths = [5, 40, 20, 15, 15, 30]
    for i, w in enumerate(widths, 1):
        ws.column_dimensions[get_column_letter(i)].width = w

    row = 1

    # Заголовок
    ws.merge_cells(f"A{row}:F{row}")
    c = ws.cell(row=row, column=1, value="ІНДИВІДУАЛЬНИЙ ПЛАН РОБОТИ НАУКОВО-ПЕДАГОГІЧНОГО ПРАЦІВНИКА")
    c.font = Font(bold=True, size=12)
    c.alignment = Alignment(horizontal="center", vertical="center")
    ws.row_dimensions[row].height = 30
    row += 1

    ws.merge_cells(f"A{row}:F{row}")
    ws.cell(row=row, column=1, value=f"Форма № Н-4.04  |  Навчальний рік: {plan.academic_year}")
    ws.cell(row=row, column=1).alignment = Alignment(horizontal="center")
    row += 1

    ws.merge_cells(f"A{row}:F{row}")
    ws.cell(row=row, column=1, value=f"Викладач: {teacher_name}")
    row += 2

    # Шапка таблиці
    headers = ["№", "Вид роботи", "Параметр", "Год. (план)", "Год. (факт)", "Примітки"]
    for col, h in enumerate(headers, 1):
        _cell(ws, row, col, h, fill=_HEADER_FILL)
    ws.row_dimensions[row].height = 20
    row += 1

    entries_by_section: dict[str, list] = {s: [] for s in SECTIONS}
    for e in plan.entries:
        if e.section in entries_by_section:
            entries_by_section[e.section].append(e)

    for section in SECTIONS:
        # Рядок розділу
        ws.merge_cells(f"A{row}:F{row}")
        _cell(ws, row, 1, SECTION_LABELS[section], bold=True, fill=_SECTION_FILL)
        for col in range(2, 7):
            c = ws.cell(row=row, column=col)
```

```
            c.fill = _SECTION_FILL
            c.border = _BORDER
        row += 1

        entries = sorted(entries_by_section[section], key=lambda e: e.sort_order)
        for idx, entry in enumerate(entries, 1):
            _cell(ws, row, 1, idx)
            _cell(ws, row, 2, entry.work_type, wrap=True)
            _cell(ws, row, 3, entry.param_value)
            _cell(ws, row, 4, entry.hours_planned)
            _cell(ws, row, 5, entry.hours_actual)
            _cell(ws, row, 6, entry.note or "", wrap=True)
            row += 1

        if not entries:
            ws.merge_cells(f"A{row}:F{row}")
            _cell(ws, row, 1, "(записів немає)")
            for col in range(2, 7):
                ws.cell(row=row, column=col).border = _BORDER
            row += 1

    # Підсумковий рядок
    row += 1
    ws.merge_cells(f"A{row}:C{row}")
    _cell(ws, row, 1, "РАЗОМ:", bold=True)
    for col in range(2, 4):
        ws.cell(row=row, column=col).border = _BORDER
    _cell(ws, row, 4, plan.total_hours_planned, bold=True)
    _cell(ws, row, 5, plan.total_hours_actual, bold=True)
    _cell(ws, row, 6, f"Норма: 1548 год.")

    buf = io.BytesIO()
    wb.save(buf)
    buf.seek(0)
    return buf



================================================================================================
FILE: frontend/src/context/AuthContext.jsx
================================================================================================

import { createContext, useContext, useEffect, useState } from 'react'
import { getMe } from '../api/auth'

const AuthContext = createContext(null)

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null)
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const token = localStorage.getItem('token')
    if (!token) {
      setLoading(false)
      return
    }
    getMe()
      .then((res) => setUser(res.data))
      .catch(() => localStorage.removeItem('token'))
      .finally(() => setLoading(false))
  }, [])

  const login = (token) => {
    localStorage.setItem('token', token)
    return getMe().then((res) => setUser(res.data))
  }

  const logout = () => {
    localStorage.removeItem('token')
    setUser(null)
  }

  return (
    <AuthContext.Provider value={{ user, loading, login, logout }}>
      {children}
    </AuthContext.Provider>
```

```
  )
}

export const useAuth = () => useContext(AuthContext)



================================================================================================
FILE: frontend/src/components/PrivateRoute.jsx
================================================================================================

import { Navigate } from 'react-router-dom'
import { useAuth } from '../context/AuthContext'

export default function PrivateRoute({ children, roles }) {
  const { user, loading } = useAuth()

  if (loading) return <div className="spinner">Завантаження...</div>
  if (!user) return <Navigate to="/login" replace />
  if (user.role === 'new_user') return <Navigate to="/pending" replace />
  if (roles && !roles.includes(user.role)) return <Navigate to="/plans" replace />

  return children
}



================================================================================================
FILE: frontend/src/pages/LoginPage.jsx
================================================================================================

import { useState } from 'react'
import { Link, useNavigate } from 'react-router-dom'
import { login as apiLogin } from '../api/auth'
import { useAuth } from '../context/AuthContext'

export default function LoginPage() {
  const { login } = useAuth()
  const navigate = useNavigate()
  const [form, setForm] = useState({ email: '', password: '' })
  const [error, setError] = useState('')
  const [loading, setLoading] = useState(false)

  const handleSubmit = async (e) => {
    e.preventDefault()
    setError('')
    setLoading(true)
    try {
      const res = await apiLogin(form)
      await login(res.data.access_token)
      navigate('/plans')
    } catch (err) {
      setError(err.response?.data?.detail || 'Помилка входу')
    } finally {
      setLoading(false)
    }
  }

  return (
    <div className="auth-page">
      <form className="auth-form" onSubmit={handleSubmit}>
        <h1>Вхід</h1>
        {error && <div className="alert alert-error">{error}</div>}
        <label>
          Email
          <input
            type="email"
            value={form.email}
            onChange={(e) => setForm({ ...form, email: e.target.value })}
            required
            autoFocus
          />
        </label>
        <label>
          Пароль
          <input
            type="password"
```

```
          value={form.password}
          onChange={(e) => setForm({ ...form, password: e.target.value })}
          required
        />
      </label>
      <button className="btn btn-primary" disabled={loading}>
        {loading ? 'Вхід...' : 'Увійти'}
      </button>
      <p className="auth-link">
        Немає акаунту? <Link to="/register">Зареєструватись</Link>
      </p>
    </form>
  </div>
 )
}
```

```
================================================================================================
FILE: frontend/src/pages/PlansPage.jsx
================================================================================================

import { useEffect, useState } from 'react'
import { useNavigate } from 'react-router-dom'
import { createPlan, deletePlan, getPlans } from '../api/plans'
import Layout from '../components/Layout'
import { useAuth } from '../context/AuthContext'
import { validateAcademicYear } from '../utils/validate'

const STATUS_LABELS = { draft: 'Чернетка', submitted: 'Подано', approved: 'Затверджено' }
const STATUS_CLASS = { draft: 'badge-gray', submitted: 'badge-blue', approved: 'badge-green' }

export default function PlansPage() {
 const { user } = useAuth()
 const navigate = useNavigate()
 const [plans, setPlans] = useState([])
 const [loading, setLoading] = useState(true)
 const [newYear, setNewYear] = useState('')
 const [creating, setCreating] = useState(false)
 const [error, setError] = useState('')

 const load = () =>
  getPlans()
   .then((r) => setPlans(r.data))
   .finally(() => setLoading(false))

 useEffect(() => { load() }, [])

 const handleCreate = async (e) => {
  e.preventDefault()
  const err = validateAcademicYear(newYear)
  if (err) { setError(err); return }
  setCreating(true)
  setError('')
  try {
   const res = await createPlan({ academic_year: newYear })
   navigate(`/plans/${res.data.id}`)
  } catch (err) {
   setError(err.response?.data?.detail || 'Помилка')
  } finally {
   setCreating(false)
  }
 }

 const handleDelete = async (id) => {
  if (!confirm('Видалити план?')) return
  await deletePlan(id)
  load()
 }

 return (
  <Layout>
   <div className="page">
    <div className="page-header">
     <h1>Індивідуальні плани</h1>
     {user?.role === 'teacher' && (
      <form className="inline-form" onSubmit={handleCreate}>
```

```
            <input
              placeholder="2024-2025"
              value={newYear}
              onChange={(e) => setNewYear(e.target.value)}
              className="input-sm"
            />
            <button className="btn btn-primary" disabled={creating}>
              + Новий план
            </button>
            {error && <span className="text-error">{error}</span>}
          </form>
        )}
      </div>

      {loading ? (
        <div className="spinner">Завантаження...</div>
      ) : plans.length === 0 ? (
        <div className="empty">Планів ще немає</div>
      ) : (
        <table className="table">
          <thead>
            <tr>
              <th>Навч. рік</th>
              {user?.role !== 'teacher' && <th>Викладач</th>}
              <th>Статус</th>
              <th>Год. план</th>
              <th>Год. факт</th>
              <th></th>
            </tr>
          </thead>
          <tbody>
            {plans.map((p) => (
              <tr key={p.id} className="table-row-link" onClick={() => navigate(`/plans/${p.id}`)}>
                <td>{p.academic_year}</td>
                {user?.role !== 'teacher' && <td>{p.teacher_name || p.teacher_email || `#${p.teacher_id}`}</td>}
                <td>
                  <span className={`badge ${STATUS_CLASS[p.status]}`}>
                    {STATUS_LABELS[p.status]}
                  </span>
                </td>
                <td>{p.total_hours_planned}</td>
                <td>{p.total_hours_actual}</td>
                <td onClick={(e) => e.stopPropagation()}>
                  {p.status !== 'approved' && user?.role === 'teacher' && (
                    <button
                      className="btn btn-danger btn-sm"
                      onClick={() => handleDelete(p.id)}
                    >
                      Видалити
                    </button>
                  )}
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      )}
    </div>
  </Layout>
 )
}
```

```
================================================================================================
FILE: frontend/src/pages/PlanDetailPage.jsx
================================================================================================

import { useEffect, useState } from 'react'
import { useNavigate, useParams } from 'react-router-dom'
import { addEntry, deleteEntry, getPlan, updateEntry, updatePlan } from '../api/plans'
import { downloadExcel } from '../api/reports'
import { getNorms } from '../api/norms'
import Layout from '../components/Layout'
import { useAuth } from '../context/AuthContext'
import { validateWorkEntry, hasErrors } from '../utils/validate'
```

```
const SECTIONS = ['навчальна', 'методична', 'наукова', 'організаційна']
const SECTION_LABELS = {
  навчальна: 'Розділ I. Навчальна робота',
  методична: 'Розділ II. Методична робота',
  наукова: 'Розділ III. Наукова робота',
  організаційна: 'Розділ IV. Організаційна та виховна робота',
}
const STATUS_BTN = { draft: 'Подати на перевірку', submitted: 'Затвердити' }

export default function PlanDetailPage() {
  const { id } = useParams()
  const { user } = useAuth()
  const navigate = useNavigate()

  const [plan, setPlan] = useState(null)
  const [norms, setNorms] = useState([])
  const [loading, setLoading] = useState(true)
  const [addingSection, setAddingSection] = useState(null)
  const [newEntry, setNewEntry] = useState({ work_type: '', param_value: 0, note: '' })
  const [entryErrors, setEntryErrors] = useState({})

  const load = () =>
    Promise.all([getPlan(id), getNorms()])
      .then(([planRes, normsRes]) => {
        setPlan(planRes.data)
        setNorms(normsRes.data)
      })
      .finally(() => setLoading(false))

  useEffect(() => { load() }, [id])

  const canEdit =
    plan?.status !== 'approved' &&
    (user?.role !== 'teacher' || plan?.teacher_id === user?.id)

  const canApprove =
    (user?.role === 'head' || user?.role === 'admin') && plan?.status === 'submitted'

  const handleStatusChange = async (newStatus) => {
    await updatePlan(id, { status: newStatus })
    load()
  }

  const handleAddEntry = async (section) => {
    const errors = validateWorkEntry(newEntry)
    if (hasErrors(errors)) { setEntryErrors(errors); return }
    setEntryErrors({})
    await addEntry(id, { ...newEntry, section })
    setAddingSection(null)
    setNewEntry({ work_type: '', param_value: 0, note: '' })
    load()
  }

  const handleDeleteEntry = async (entryId) => {
    if (!confirm('Видалити рядок?')) return
    await deleteEntry(id, entryId)
    load()
  }

  const handleActualChange = async (entryId, value) => {
    const num = parseFloat(value)
    if (isNaN(num) || num < 0) return
    await updateEntry(id, entryId, { hours_actual: num })
    load()
  }

  const openAddForm = (section) => {
    setAddingSection(section)
    setNewEntry({ work_type: '', param_value: 0, note: '' })
    setEntryErrors({})
  }

  if (loading) return <Layout><div className="spinner">Завантаження...</div></Layout>
  if (!plan) return <Layout><div className="empty">План не знайдено</div></Layout>

  const normsBySection = (section) => norms.filter((n) => n.section === section)
  const entriesBySection = (section) => plan.entries.filter((e) => e.section === section)
```

```
return (
  <Layout>
    <div className="page">
      <div className="page-header">
        <div>
          <button className="btn btn-ghost btn-sm" onClick={() => navigate('/plans')}>
            ← Назад
          </button>
          <h1>Індивідуальний план {plan.academic_year}</h1>
        </div>
        <div className="action-row">
          <button className="btn btn-secondary" onClick={() => downloadExcel(plan.id)}>
            Завантажити Excel
          </button>
          {canEdit && plan.status === 'draft' && user?.role === 'teacher' && (
            <button className="btn btn-primary" onClick={() => handleStatusChange('submitted')}>
              {STATUS_BTN.draft}
            </button>
          )}
          {canApprove && (
            <button className="btn btn-primary" onClick={() => handleStatusChange('approved')}>
              {STATUS_BTN.submitted}
            </button>
          )}
        </div>
      </div>

      <div className="plan-summary">
        {user?.role !== 'teacher' && (
          <span>Викладач: <strong>{plan.teacher_name || plan.teacher_email || `#${plan.teacher_id}`}</strong></span>
        )}
        <span>Статус: <strong>{plan.status}</strong></span>
        <span>Годин заплановано: <strong>{plan.total_hours_planned}</strong></span>
        <span>Годин виконано: <strong>{plan.total_hours_actual}</strong></span>
        <span>Норма: <strong>1548</strong></span>
      </div>

      {SECTIONS.map((section) => (
        <div key={section} className="section-block">
          <h2>{SECTION_LABELS[section]}</h2>
          <table className="table">
            <thead>
              <tr>
                <th>Вид роботи</th>
                <th>Параметр</th>
                <th>Год. план</th>
                <th>Год. факт</th>
                <th>Примітки</th>
                {canEdit && <th></th>}
              </tr>
            </thead>
            <tbody>
              {entriesBySection(section).map((entry) => (
                <tr key={entry.id}>
                  <td>{entry.work_type}</td>
                  <td>{entry.param_value}</td>
                  <td>{entry.hours_planned}</td>
                  <td>
                    {canEdit ? (
                      <input
                        type="number"
                        min="0"
                        className="input-sm input-narrow"
                        defaultValue={entry.hours_actual}
                        onBlur={(e) => handleActualChange(entry.id, e.target.value)}
                      />
                    ) : entry.hours_actual}
                  </td>
                  <td>{entry.note || '—'}</td>
                  {canEdit && (
                    <td>
                      <button
                        className="btn btn-danger btn-sm"
                        onClick={() => handleDeleteEntry(entry.id)}
                      >
                        ×
```

```
                  </button>
                </td>
              )}
            </tr>
          ))}
        </tbody>
      </table>

      {canEdit && addingSection === section ? (
        <div className="add-entry-form">
          <div className="entry-field">
            <select
              value={newEntry.work_type}
              onChange={(e) => setNewEntry({ ...newEntry, work_type: e.target.value })}
              className={`input-sm ${entryErrors.work_type ? 'input-error' : ''}`}
            >
              <option value="">— Оберіть вид роботи —</option>
              {normsBySection(section).map((n) => (
                <option key={n.work_type} value={n.work_type}>{n.label}</option>
              ))}
            </select>
            {entryErrors.work_type && <span className="field-error">{entryErrors.work_type}</span>}
          </div>
          <div className="entry-field">
            <input
              type="number"
              min="0"
              placeholder={norms.find((n) => n.work_type === newEntry.work_type)?.param_name || 'Значення'}
              value={newEntry.param_value}
              onChange={(e) => setNewEntry({ ...newEntry, param_value: e.target.value })}
              className={`input-sm input-narrow ${entryErrors.param_value ? 'input-error' : ''}`}
            />
            {entryErrors.param_value && <span className="field-error">{entryErrors.param_value}</span>}
          </div>
          <input
            type="text"
            placeholder="Примітка"
            value={newEntry.note}
            onChange={(e) => setNewEntry({ ...newEntry, note: e.target.value })}
            className="input-sm"
          />
          <button className="btn btn-primary btn-sm" onClick={() => handleAddEntry(section)}>
            Додати
          </button>
          <button className="btn btn-ghost btn-sm" onClick={() => setAddingSection(null)}>
            Скасувати
          </button>
        </div>
      ) : canEdit ? (
        <button className="btn btn-ghost btn-sm" onClick={() => openAddForm(section)}>
          + Додати рядок
        </button>
      ) : null}
    </div>
  ))}
</div>
</Layout>
)
}
```

```
================================================================================================
FILE: frontend/src/App.jsx
================================================================================================

import { Navigate, Route, Routes } from 'react-router-dom'
import PrivateRoute from './components/PrivateRoute'
import LoginPage from './pages/LoginPage'
import RegisterPage from './pages/RegisterPage'
import PendingPage from './pages/PendingPage'
import PlansPage from './pages/PlansPage'
import PlanDetailPage from './pages/PlanDetailPage'
import NormsPage from './pages/NormsPage'
import UsersPage from './pages/UsersPage'

export default function App() {
```

```
  return (
    <Routes>
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/pending" element={<PendingPage />} />
      <Route
        path="/plans"
        element={<PrivateRoute><PlansPage /></PrivateRoute>}
      />
      <Route
        path="/plans/:id"
        element={<PrivateRoute><PlanDetailPage /></PrivateRoute>}
      />
      <Route
        path="/norms"
        element={<PrivateRoute roles={['admin']}><NormsPage /></PrivateRoute>}
      />
      <Route
        path="/users"
        element={<PrivateRoute roles={['admin']}><UsersPage /></PrivateRoute>}
      />
      <Route path="*" element={<Navigate to="/plans" replace />} />
    </Routes>
  )
}
```