Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ЗАСТОСУНОК ДЛЯ ЗБОРУ ФІДБЕКУ СТУДЕНТІВ У НАВЧАЛЬНИХ РЕТРОСПЕКТИВАХ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп’ютерні науки»
Юрій Миколайович Черемха
Науковий керівник:
старший викладач
Тетяна Анатоліївна Кирик

м. Рівне, 2026 рік

ЗМІСТ

ВСТУП

У сучасному освітньому середовищі дедалі більшої ваги набувають інструменти, що забезпечують регулярний та структурований зворотний зв’язок між здобувачами освіти й викладачами. Навчальний процес є динамічним: змінюються підходи до викладання, темп подачі матеріалу, формат занять і очікування студентів. За таких умов важливо не лише оцінювати підсумкові результати навчання, а й оперативно отримувати інформацію про те, як саме сприймається навчальний матеріал, які труднощі виникають і що можна покращити вже в межах поточного курсу. Одним із ефективних форматів для цього є навчальні ретроспективи, у яких зворотний зв’язок виступає основою для осмислення досвіду та планування змін.

Актуальність теми зумовлена потребою закладів вищої освіти у впровадженні зрозумілих і доступних механізмів збору фідбеку, які можуть працювати як після заняття, так і в оперативному режимі. Традиційні підходи до опитування часто не забезпечують належної системності, швидкого отримання результатів та зручного аналізу, а також не завжди стимулюють студентів до відкритого висловлення думок. У зв’язку з цим актуальним є створення вебзастосунку, що дозволяє організувати процес збору відгуків у ретроспективах, забезпечити збереження даних, отримувати узагальнені показники та робити висновки на основі кількісних і якісних відповідей.

Мета кваліфікаційної роботи полягає у проєктуванні та реалізації вебзастосунку для збору й аналізу студентського фідбеку в навчальних ретроспективах, а також у створенні інтерфейсів для двох ролей користувачів (студента та викладача) з підтримкою керування процесом збору відгуків, з метою автоматизації процесу отримання, збереження та візуалізації результатів опитувань.

Об’єктом дослідження є процес отримання та використання зворотного зв’язку в навчальному процесі, зокрема в межах навчальних ретроспектив.

Предметом дослідження є програмні засоби та технології розробки вебзастосунку для збору фідбеку, зокрема підходи до побудови клієнтського

інтерфейсу, організації хмарного зберігання даних і формування візуальної аналітики результатів

Методи дослідження. У кваліфікаційній роботі застосовано комплекс методів дослідження для досягнення поставленої мети та виконання визначених завдань:

Аналіз і порівняння: для опрацювання науково-методичних підходів до організації зворотного зв’язку в освітньому процесі та зіставлення функціональних можливостей існуючих інструментів збору фідбеку з метою визначення їх застосовності у форматі навчальних ретроспектив.

Узагальнення та систематизація: для формування вимог до вебзастосунку, визначення ключових сценаріїв використання та обґрунтування концепції системи збору й первинного аналізу відгуків.

Моделювання: для проєктування архітектурної моделі застосунку, визначення ролей користувачів (студент/викладач), побудови схем взаємодії компонентів та розробки структури даних у NoSQL-середовищі (Cloud Firestore).

Проєктування: для розробки веб-інтерфейсів студента й адміністратора з урахуванням вимог до зручності використання, а також для визначення логіки взаємодії клієнтської частини із хмарною інфраструктурою.

Експериментальне тестування: для перевірки працездатності та валідації основних функцій системи (подання відгуку, керування доступністю збору, отримання та відображення статистики, візуалізація результатів).

Для досягнення поставленої мети визначено такі завдання кваліфікаційної роботи:

– дослідити роль і значення студентського фідбеку в навчальних ретроспективах та сформувати вимоги до системи збору відгуків;

– проаналізувати наявні інструменти збору зворотного зв’язку та визначити їх функціональні можливості й обмеження в освітньому контексті середовищі;

– спроєктувати архітектуру вебзастосунку з використанням хмарної інфраструктури та визначити модель зберігання даних;

– реалізувати інтерфейс студента для подання відгуку та інтерфейс викладача для керування збором і перегляду результатів;

– забезпечити первинну аналітику та візуалізацію даних (агрегація оцінок, графічне представлення розподілу, перегляд та аналіз коментарів).

– забезпечити захист системи та розмежування прав доступу за допомогою механізмів автентифікації та політик безпеки;

– інтегрувати інструменти штучного інтелекту на основі великих мовних моделей для автоматизованого семантичного аналізу та узагальнення студентських коментарів.

Структура роботи. Основна частина кваліфікаційної роботи складається з чотирьох розділів. У першому розділі розглянуто теоретичні аспекти використання фідбеку в навчальному процесі та обґрунтовано потребу в інструментах його системного збору; також наведено огляд поширених рішень і їх застосовність до формату ретроспектив. Другий розділ присвячено опису технологічних засобів і принципів реалізації вебзастосунку, включаючи організацію клієнтської частини та використання хмарної інфраструктури для зберігання даних. У третьому розділі подано архітектурну модель застосунку, описано логіку взаємодії компонентів (студентського та викладацького модулів), а також представлено реалізовані елементи інтерфейсу й засоби аналітики та візуалізації результатів. У четвертому розділі наведено результати комплексного тестування розробленого вебзастосунку, що включає функціональну перевірку клієнтського та адміністративного модулів, верифікацію надійності політик безпеки із застосуванням інструментарію Postman та аналіз стійкості інтеграції з ШІ до мережевих відмов. Також описано процес автоматизованого розгортання вебзастосунку на платформі Firebase Hosting.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ У СИСТЕМІ ЗБОРУ СТУДЕНТСЬКИХ ФІДБЕКІВ

1.1 Роль студентського фідбеку у навчальному процесі

Студентський фідбек є важливим елементом сучасного навчального процесу, оскільки забезпечує прямий зворотний зв’язок між студентами та викладачами. Він допомагає оперативно виявляти проблеми в організації занять і сприйнятті матеріалу, а також підтримує перехід до студентоцентрованої моделі освіти, де враховується досвід і потреби здобувачів.

Зворотний зв’язок дає змогу оцінити якість викладання з позиції студентів: наскільки зрозуміло подано тему, чи відповідає темп заняття можливостям групи, наскільки ефективними є приклади, завдання та комунікація. На відміну від підсумкових оцінок, фідбек показує реальний навчальний досвід, включно з рівнем залученості, мотивації та психологічного комфорту.

Важливою перевагою фідбеку є те, що він формує діалог у системі «викладач — студент». Анонімні форми зворотного зв’язку зменшують бар’єри та страх відкрито висловлювати зауваження, тому студенти частіше надають чесні пропозиції. Для викладача це стає джерелом конкретних сигналів: які теми потребують додаткового пояснення, що варто змінити в структурі заняття, які методи працюють ефективно, а які — ні [14].

Регулярний збір відгуків дозволяє вдосконалювати навчання не «після завершення семестру», а в процесі: коригувати подачу матеріалу, добір завдань, формат взаємодії, впроваджувати дискусії, групову роботу та інші активні методи. Коли студенти бачать, що їхні коментарі враховують, зростає довіра, відповідальність і готовність конструктивно взаємодіяти, а для викладача позитивний фідбек може бути додатковим чинником професійної підтримки.

У систематизованому вигляді студентський фідбек є корисним і для адміністрації закладу освіти: він допомагає оцінювати загальну якість освітніх компонентів, бачити типові труднощі, планувати підвищення кваліфікації та приймати обґрунтовані організаційні рішення. У підсумку практика регулярного

зворотного зв’язку формує культуру відкритості й партнерства, покращує освітній клімат і підсилює якість освіти в довгостроковій перспективі.

1.2 Порівняння з існуючими рішеннями

У практиці навчальних ретроспектив найчастіше застосовують чотири класи готових інструментів: онлайн-форми (для швидкого збору відповідей за посиланням), професійні платформи опитувань (для складніших сценаріїв і умовної логіки), live-сервіси взаємодії (для збору реакцій під час заняття) та LMS-модулі (коли фідбек збирається в межах курсу).

Google Forms і Microsoft Forms — це базові інструменти для швидкого запуску опитування: створення анкети займає мінімум часу, поширення відбувається через посилання, а підсумки доступні одразу після збору відповідей. Перевагою є низький поріг входу та зрозумілий інтерфейс як для автора, так і для респондентів, а також базова аналітика (зведення, графіки, перегляд відповідей) і можливість експорту (зокрема у табличні формати для подальшої обробки). Недоліком цього класу рішень є обмежена кастомізація процесу ретроспективи: якщо потрібні складні сценарії, нестандартні правила доступу, гнучкі ролі або специфічна логіка, доводиться або спрощувати структуру опитування, або переходити до спеціалізованіших платформ [21, 23, 16, 20].

Typeform і SurveyMonkey належать до більш професійних платформ опитувань, де фокус на гнучкості структури анкети та умовній логіці. Їхня перевага в тому, що можна будувати розгалуження, показувати різні питання залежно від відповідей, застосовувати шаблони та точніше керувати сценарієм проходження, що корисно для поглибленого аналізу причин, очікувань і проблемних зон у ретроспективах. Водночас недоліки проявляються у вищому порозі налаштувань і необхідності тестування логіки: складні анкети легше «зламати» некоректними правилами переходів або надмірною кількістю питань. Додатково, частина можливостей, особливо розширена логіка та аналітика, часто залежить від тарифу, що обмежує використання в навчальних умовах [22, 25].

Slido та Mentimeter (і подібні live-інструменти) найкраще підходять для ретроспектив у форматі синхронної сесії, коли важлива миттєва взаємодія групи. Їхні переваги — високий рівень залучення, швидкі формати (голосування, запитання-відповіді, хмари слів), відображення результатів у реальному часі та зручність модерації обговорення. Це дозволяє перетворити ретроспективу на керовану дискусію, та зібрати ключові теми. Недоліком є залежність від організації: потрібен час у розкладі, ведучий/модератор, а також участь більшості студентів одночасно. Для асинхронного збору фідбеку після заняття такі сервіси зазвичай менш зручні або вимагають інших сценаріїв використання [19, 27].

Moodle та Canvas (LMS-підхід) виграють інтеграцією: фідбек збирається всередині курсу, з використанням ролей і доступів навчальної системи, що спрощує адміністрування та дає контекст (групи, дисципліна, модулі). Перевагою є системність: результати зберігаються в одному середовищі з навчальними матеріалами, а доступи та участь студентів керуються правилами курсу. Недоліком LMS-моделі часто є обмежена гнучкість форм/візуалізацій та інколи менш сучасний UX порівняно зі спеціалізованими платформами. Для нестандартних сценаріїв ретроспективи: складні гілки, нетипові формати питань, інтерактивність як у live-сервісах можливостей LMS може бути недостатньо або їх складніше налаштувати без адміністративної підтримки [6, 15].

У підсумку, вибір готового рішення для навчальних ретроспектив визначається тим, коли і як збирається фідбек: для збору відгуку після заняття найчастіше застосовують форми, для ретроспектив наживо – live-інструменти взаємодії, а для системної курсової оцінки й адміністративного контролю – LMS-модулі, що забезпечують контекст курсу та керування доступами.

1.3 Переваги впровадження власної системи збору фідбеку у навчальних ретроспективах

Впровадження власної системи збору зворотного зв’язку в освітньому середовищі доцільно розглядати не лише як технічну альтернативу готовим

сервісам, а як інструмент управління якістю навчання, що може бути адаптований під конкретні організаційні, методичні та аналітичні потреби закладу освіти.

Першою суттєвою перевагою є безкоштовність експлуатації на рівні ліцензування. На відміну від ряду комерційних платформ, де розширена логіка опитувань, аналітика або адміністративні функції прив’язані до тарифних планів, власна система не потребує регулярних ліцензійних платежів. Це дозволяє спрямовувати ресурси не на оплату підписок, а на покращення функціональності, підтримку та розвиток інструменту відповідно до реальних потреб навчального процесу.

Другою перевагою виступає глибока інтеграція в освітню програму та внутрішні процеси. Власна система може бути узгоджена з логікою дисциплін, модульного контролю, форматом ретроспектив (після теми, після практичної, після модуля), а також із внутрішніми стандартами якості викладання. Завдяки цьому збір фідбеку перестає бути епізодичною активністю, а перетворюється на регулярний елемент навчальної методики із прогнозованими правилами, періодичністю та очікуваними результатами.

Важливою функціональною перевагою є підтримка live-сценаріїв, тобто можливість отримувати зворотний зв’язок у режимі реального часу під час ретроспективи або заняття. Такий режим підсилює залученість студентів, дозволяє оперативно фіксувати реакції групи, швидше виявляти проблемні теми й одразу коригувати хід обговорення. На практиці це дає можливітсь не відкладати результати на потім, а змінювати щось безпосередньо в процесі пари.

Окремої уваги заслуговує делікатніше налаштування та розширена аналітика. Власна система може підтримувати ті метрики й зрізи, які найбільш релевантні для закладу: деталізацію за дисциплінами, групами, викладачами, типами занять, часовими періодами; порівняння динаміки, виявлення тенденцій, аналіз розподілів, кореляцій, повторюваних проблем. Важливо, що аналітичні панелі можна проектувати під конкретні управлінські запити, а не пристосовуватися до обмежень універсальних шаблонів.

Суттєвим педагогічним фактором є анонімність, що підвищує відкритість до відгуків. Коли студент не пов’язує відповідь зі своєю особою, зростає ймовірність чеснішого формулювання проблем, сумнівів і критики. Це особливо важливо в ситуаціях, де присутній психологічний бар’єр: страх негативної оцінки, конфлікту або упередженого ставлення. Водночас анонімність слід поєднувати з культурою академічної доброчесності та етичними правилами комунікації, щоб уникати неконструктивних повідомлень.

Технічно прикладною перевагою, яка безпосередньо впливає на якість даних, є можливість обмежити кількість відповідей з одного браузера, один відгук з одного пристрою чи профілю. Це зменшує ризик повторних подань та частково захищає від спотворення результатів через багаторазове голосування. У навчальному середовищі така перевірка працює як простий механізм підвищення достовірності агрегованих показників за мінімальних витрат часу.

Додатково, власна система має низку стратегічних переваг. По-перше, це контроль над даними та їх життєвим циклом: заклад сам визначає, які поля збираються, як довго зберігаються відповіді, хто має доступ до сирих даних і хто — лише до агрегованих звітів. По-друге, це розширюваність і модульність: за потреби можна поступово додавати нові типи питань, сценарії фідбеку після занять або live-під час ретроспективи, автоматизовані звіти, категоризацію коментарів, механізми «плану дій» після ретроспективи. По-третє, це можливість інтеграції з іншими внутрішніми системами як розклад, електронний журнал, LMS, що дозволяє зменшити ручні операції та підвищити точність прив’язки фідбеку до контексту заняття.

У сукупності наведені переваги демонструють, що власна система збору фідбеку може бути не просто інструментом опитування, а елементом цілісної моделі управління якістю навчання: від регулярного збору даних до їх інтерпретації, прийняття рішень і оцінювання ефективності впроваджених змін.

1.4 Основні вимоги до функціоналу системи

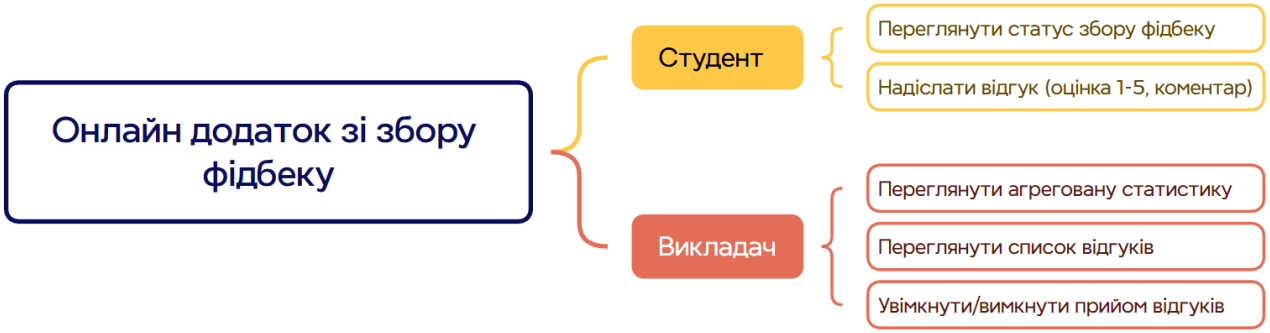

Рисунок 1.1. Mind Map системи збору фідбеку у навчальних ретроспективах

Сучасна структура системи збору фідбеку для навчальних ретроспектив базується на чіткому розподілі ролей між студентом та авторизованим викладачем рис. 1.1, що забезпечується окремими функціональними модулями та вебінтерфейсами.

Система повинна забезпечувати надійну автентифікацію та контроль доступу. Доступ до перегляду результатів, аналітики та керування статусом збору дозволяється лише після успішної перевірки облікових даних викладача (email та пароль). На рівні бази даних доступ до читання та запису повинен регламентуватися правилами безпеки, що запобігають несанкціонованому перегляду або зміні даних неавторизованими користувачами.

Система повинна надавати можливість подання відгуку у деталізованому предметному контексті. Стандартизований формат подання відгуку включає: обов’язковий вибір викладача зі списку доступних у системі, динамічне завантаження та вибір навчальної дисципліни, закріпленої за обраним викладачем, вибір оцінки за фіксованою шкалою 1–5, додаткове необов’язкове текстове поле «коментар». Перед відправленням система має виконувати валідацію обраних параметрів та відображати повідомлення про статус операції.

Система повинна підтримувати структуроване та безпечне зберігання даних у хмарній NoSQL-базі. Дані відгуку мають зберігатися у вигляді документів, що містять реляційні зв’язки, а також серверну часову мітку timestamp для забезпечення хронологічної точності аналітики.

Система повинна забезпечувати керування доступністю збору фідбеку в реальному часі**.** Ця вимога реалізується через централізований параметр стану в базі даних, який викладач може змінювати через адміністративну панель. При вимкненні прийому відгуків студентська форма повинна автоматично блокуватися для всіх користувачів.

Система повинна обмежувати повторне надсилання відгуків для конкретного контексту. Для забезпечення унікальності відповідей використовується механізм фіксації факту відправлення на рівні локального сховища браузера. Блокування повторного подання має спрацьовувати індивідуально для кожної комбінації «викладач – предмет».

Система повинна реалізовувати розширену аналітику та візуалізацію результатів. Адміністративний модуль має забезпечувати: відображення агрегованої статистики (середня оцінка, розподіл кількості оцінок) за допомогою динамічних, можливість фільтрації результатів за конкретною навчальною дисципліною, автоматичне оновлення даних при надходженні нових відгуків без перезавантаження сторінки.

Система повинна враховувати аспекти безпеки та адаптивності. Вимога включає обов'язкове екранування спеціальних символів у текстових коментарях для захисту адміністративної панелі від шкідливих вставок (XSS). Інтерфейс має бути повністю адаптивним для коректної роботи на мобільних пристроях, що є пріоритетним сценарієм для студентської аудиторії.

Напрями подальшого вдосконалення

Напрями подальшого вдосконалення системи включають розвиток гнучкого конструктора запитань або розширення набору шаблонів ретроспектив за методиками «що було добре», «що покращити» та «що завадило». Перспективним є впровадження додаткових шкал оцінювання, таких як «зрозумілість матеріалу»,

«темп заняття» чи «практична користь», а також тегування коментарів для глибшої категоризації.

З точки зору експлуатації та UX доцільно запланувати впровадження QR-кодів для швидкого доступу студентів до форми ретроспективи безпосередньо в аудиторії. Також актуальним залишається вдосконалення модульності коду через централізацію повідомлень про помилки та розширення журналювання подій, що підвищить підтримуваність системи при її масштабуванні на рівень усього навчального закладу.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ЗАСТОСУНКУ

2.1 HTML, CSS та JavaScript як основа клієнтського інтерфейсу

Клієнтський інтерфейс сучасного вебзастосунку ґрунтується на тріаді технологій HTML, CSS та JavaScript, які у сукупності формують логічну структуру сторінки, її візуальне оформлення та інтерактивну складову. Такий підхід є стандартом для розробки вебінтерфейсів, оскільки забезпечує кросплатформеність, доступність через браузер та можливість швидкого розгортання без встановлення додаткового програмного забезпечення. У контексті системи збору фідбеку у навчальних ретроспективах ці технології дозволяють створити інтерфейс, орієнтований на оперативне подання відгуків студентами та на зручний перегляд агрегованих результатів викладачем.

HTML (HyperText Markup Language) виконує роль семантичного каркаса інтерфейсу. За допомогою HTML визначаються основні структурні елементи сторінки: заголовки, форми, поля введення, перемикачі, кнопки, таблиці, інформаційні блоки тощо. Семантична розмітка є принципово важливою з погляду доступності й коректного сприйняття сторінки браузерами та допоміжними засобами (скрінрідерами), а також сприяє логічній організації інтерфейсу. Для системи збору фідбеку HTML-структура забезпечує чітку послідовність дій користувача: ознайомлення зі статусом збору, заповнення форми (оцінка й коментар) та надсилання даних, а також відображення повідомлень про успішність або помилки.

CSS (Cascading Style Sheets) відповідає за презентаційний рівень і забезпечує візуальну цілісність інтерфейсу. Саме CSS визначає типографіку, кольорову схему, розміщення елементів, відступи, адаптивність, а також узгодженість компонентів у межах сторінки. Для вебзастосунків навчального призначення особливого значення набувають читабельність, мінімізація візуального шуму, контрастність та однозначність інтерпретації елементів керування, наприклад, чітке виділення активних кнопок і станів. Крім того, CSS дозволяє реалізовувати адаптивний дизайн, завдяки чому інтерфейс коректно відображається на різних пристроях —

від смартфонів до настільних комп’ютерів. Для системи фідбеку це критично, оскільки студенти часто взаємодіють із формою саме через мобільні пристрої.

JavaScript формує поведінкову частину клієнтського інтерфейсу, забезпечуючи інтерактивність і зв’язок із серверною/хмарною інфраструктурою. У межах вебзастосунку JavaScript виконує обробку подій: клік, введення, надсилання форми, валідацію введених даних, керування станами інтерфейсу (активний/неактивний збір фідбеку, блокування повторного надсилання), а також оновлення відображення без перезавантаження сторінки. З точки зору користувацького досвіду це дозволяє реалізувати швидку й зрозумілу взаємодію: користувач одразу отримує повідомлення про результат дії, а викладач — актуальні дані в адміністративній панелі.

З позиції архітектури клієнтської частини важливою є реалізація принципу розділення відповідальностей: HTML відповідає за структуру і семантику, CSS — за стиль, а JavaScript — за логіку та взаємодію. Такий поділ спрощує супровід коду, полегшує тестування інтерфейсних сценаріїв і дає змогу масштабувати застосунок без порушення його цілісності. У практичному застосуванні це означає, що зміни у дизайні (CSS) не мають впливати на бізнес-логіку (JS), а розширення логіки не повинно вимагати повного переписування структури сторінки (HTML).

Окремо слід підкреслити значення клієнт-серверної взаємодії, яка в сучасних вебрішеннях реалізується через хмарні сервіси. JavaScript виступає основним інструментом реалізації логіки на стороні клієнта, забезпечуючи асинхронний обмін даними між інтерфейсом користувача та хмарною інфраструктурою. У контексті системи збору фідбеку він виконує роль «посередника»: відправляє відповіді до бази даних, отримує поточний статус збору та підтягує результати для побудови динамічних візуалізацій. Завдяки цьому інтерфейс стає не статичним набором сторінок, а гнучкою системою, що реагує на зміни даних у реальному часі.

Таким чином, HTML, CSS та JavaScript утворюють технологічну основу клієнтського інтерфейсу, яка забезпечує структурованість, візуальну зрозумілість та інтерактивність вебзастосунку. Для системи збору фідбеку у навчальних ретроспективах ця тріада є оптимальною, оскільки дозволяє реалізувати доступний

у браузері інтерфейс для студентів і викладачів, підтримати адаптивність, забезпечити швидку обробку введення та організувати надійний обмін даними з хмарною інфраструктурою.

2.2 Firebase як хмарна інфраструктура

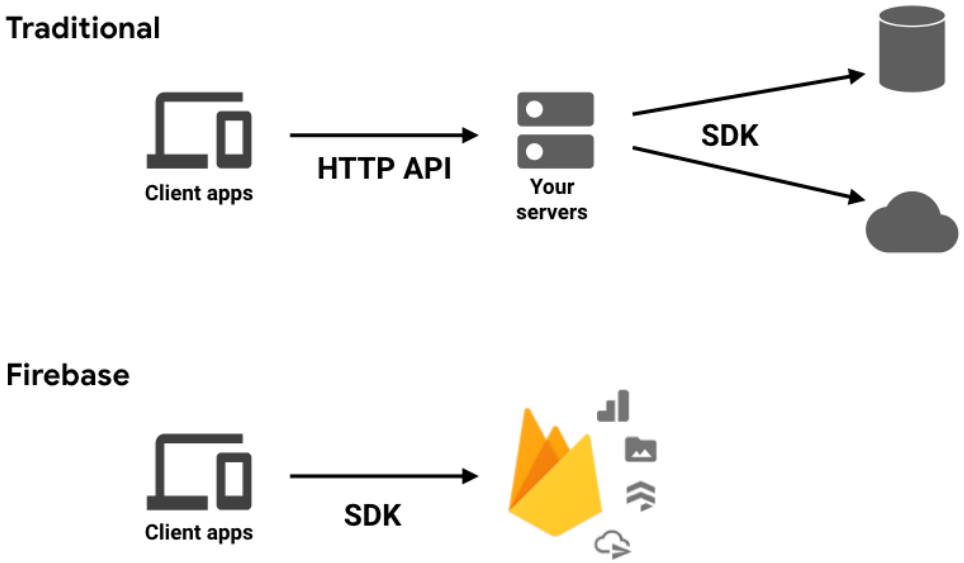

Рисунок 2.1. Порівняння традиційної клієнт-серверної взаємодії та підходу Firebase (BaaS)

Діаграма на рис. 2.1 представляє Firebase як хмарну платформу типу Backend-as-a-Service (BaaS), що надає набір готових сервісів для побудови серверної частини застосунків без необхідності розгортати та адмініструвати власні сервери. У контексті вебзастосунків Firebase виконує роль інфраструктурного шару, який забезпечує зберігання даних, автентифікацію, хостинг, виконання серверної логіки, а також моніторинг і керування якістю роботи системи. Такий підхід спрощує розробку прототипів і продуктивних систем, зменшуючи витрати на підтримку інфраструктури та прискорюючи впровадження [8].

Ключовим компонентом Firebase для систем збору фідбеку є Cloud Firestore, що реалізує NoSQL модель зберігання. Firestore підтримує документно-колекційний підхід, у межах якого дані організовуються у колекції документів із

полями різних типів. Для задачі зворотного зв’язку це дозволяє природно представляти кожен відгук як окремий документ з оцінкою, коментарем і часовою міткою, а службові налаштування, наприклад, «збір увімкнено/вимкнено» — як окремий документ конфігурації. Важливою характеристикою Firestore є підтримка потокового оновлення даних у реальному часі, що створює підґрунтя для адміністративних панелей із живою статистикою без перезавантаження сторінки [12].

Не менш значущим елементом хмарної інфраструктури є Firebase Hosting, який забезпечує розміщення статичного клієнтського застосунку (HTML/CSS/JavaScript) з доставкою через мережу розповсюдження контенту (CDN). Для освітніх застосунків це означає стабільний доступ до інтерфейсу студентів та викладачів, спрощений процес публікації (деплой) і зрозумілу модель версіонування оновлень [9]. Hosting логічно доповнює Firestore, формуючи типову архітектуру статичного фронтенду та хмарної бази даних, де значна частина бізнес-логіки може виконуватися на клієнті.

З позиції безпеки та керованості важливою складовою Firebase виступають Security Rules – правила доступу, що визначають, хто і які операції може виконувати з даними: читання, запис, оновлення, видалення [10, 11]. У навчальних сценаріях це забезпечує можливість формально закріпити розмежування прав: наприклад, дозволити студентам лише створювати записи відгуків, а модераторам — змінювати налаштування збору та переглядати агреговану інформацію. За необхідності контроль доступу підсилюється Firebase Authentication, який надає механізми входу та ідентифікації користувачів через пошту, зовнішні провайдери тощо, і може бути використаний для реалізації ролей: викладач, студент, адміністратор у поєднанні з правилами безпеки [7].

Для розширення функціональності Firebase пропонує механізм Cloud Functions або інтеграцію з Google Cloud, що дозволяє виконувати серверні функції у відповідь на події, наприклад, появу нового відгуку або HTTP-запиту. У системі ретроспектив це відкриває можливості централізованого підрахунку статистики, формування звітів, автоматичної перевірки обмежень, а також реалізації інтеграцій.

Таким чином, Firebase підтримує як мінімалістичні прототипи без бекенду, так і поступовий перехід до більш «суворих» серверних механізмів контролю.

Важливою перевагою Firebase як інфраструктури є масштабованість і готовність до пікових навантажень, що характерні для навчальних процесів, наприклад, коли вся група одночасно надсилає фідбек після заняття. Хмарна модель зменшує ризики перевантаження власного сервера, а також спрощує супровід, оскільки оновлення й базове адміністрування платформи виконуються провайдером. Додатково Firebase надає засоби моніторингу та контролю якості: журналювання, спостережуваність, інструменти аналізу продуктивності та помилок, що корисно для підтримки стабільної роботи в умовах реальної експлуатації.

Отже, Firebase як хмарна інфраструктура формує технологічно обґрунтовану основу для системи збору фідбеку у навчальних ретроспективах: платформа забезпечує швидке розгортання, централізоване зберігання даних, оновлення в реальному часі, механізми контролю доступу та можливість еволюційного розвитку — від простого прототипу до повноцінного сервісу з розширеною аналітикою й інтеграціями.

2.3 Chart.js як інструмент для візуалізації фідбеків

Chart.js є поширеною JavaScript-бібліотекою для побудови графіків у вебзастосунках, що використовує елемент HTML5 Canvas як основу для відображення візуальних компонентів. У контексті системи збору фідбеку Chart.js доцільно розглядати як інструмент оперативної інтерпретації даних, який переводить числові результати опитування, зокрема оцінки за шкалою 1–5 у наочні графічні форми. Це підвищує читабельність інформації для викладача або модератора ретроспективи, зменшує час на аналіз і дозволяє швидше приймати педагогічні та організаційні рішення.

Ключова цінність Chart.js у системах фідбеку полягає у можливості візуалізувати розподіл відповідей. Для шкал оцінювання типовим рішенням є стовпчикова діаграма (bar chart), яка відображає, скільки студентів обрали кожне

значення (1, 2, 3, 4, 5) [4]. Така форма представлення надає більш інформативну картину, ніж лише середнє значення: наприклад, дозволяє відрізнити ситуацію «переважно високі оцінки» від ситуації «поляризація» (частина — 5, частина — 1), що має різні методичні наслідки. Отже, Chart.js підтримує не лише демонстрацію «рівня задоволеності», а й діагностику неоднорідності групових реакцій.

Додатковою перевагою Chart.js є придатність до оновлення даних у реальному часі, що важливо для адміністративних панелей. У разі використання потокового отримання даних, наприклад, з хмарної бази даних графік може автоматично перераховуватися при появі нових відгуків. Це формує підхід «живої» аналітики, коли викладач спостерігає зміни розподілу оцінок у міру надходження відповідей, що особливо корисно в ретроспективах одразу після заняття або під час live-обговорення.

З погляду реалізації Chart.js має методичну перевагу як бібліотека з низьким порогом інтеграції. Вона легко підключається до сторінки й дозволяє описувати діаграми декларативно: задається тип графіка, набори даних, підписи осей та параметри відображення. Разом із тим бібліотека забезпечує достатній рівень налаштування: легенди, підказки, шкали, що дозволяє адаптувати візуалізацію до академічних вимог звітності й презентабельності.

З точки зору якості аналітики Chart.js підтримує різні способи подання результатів: окрім стовпчикових діаграм, можуть використовуватися лінійні графіки для трендів у часі (зміни середнього балу або частки низьких оцінок між заняттями), кругові діаграми для часток, а також комбіновані рішення. У подальшому розвитку системи це відкриває шлях до формування візуальних панелей, де показники представлені на різних рівнях деталізації: від загального стану групи — до порівняння модулів, тем, типів занять.

Водночас застосування Chart.js має й певні обмеження. Оскільки рендеринг відбувається через Canvas, то при дуже великих обсягах даних або потребі у складних інтерактивних сценаріях може знадобитися оптимізація або перехід до більш спеціалізованих бібліотек. Крім того, коректність висновків напряму залежить від підготовки даних: система має забезпечити правильний підрахунок

частот, узгоджене сортування шкал, коректну обробку пропусків і єдині правила агрегації, інакше навіть якісно побудована діаграма може ввести в оману.

Отже, Chart.js є обґрунтованим вибором для візуалізації фідбеків у навчальних ретроспективах, оскільки поєднує простоту інтеграції, достатню гнучкість налаштувань та можливість відображення статистики у зрозумілому графічному вигляді. Використання цієї бібліотеки підвищує аналітичну цінність зібраних даних, сприяє швидкому прочитанню результатів і підтримує прийняття рішень на основі кількісних показників.

2.4 Google Gemini API як інструмент для аналізу фідбеків

Google Gemini API є потужним інтерфейсом для взаємодії з великими мовними моделями (LLM) від Google, що дозволяє інтегрувати технології обробки природної мови (NLP) у вебзастосунки. У контексті системи збору фідбеку Gemini API доцільно розглядати як інструмент інтелектуальної інтерпретації якісних даних, який автоматизує аналіз неструктурованих текстових коментарів студентів. Це позбавляє викладача необхідності вручну опрацьовувати кожен відгук, суттєво зменшує час на аналіз та дозволяє миттєво отримувати стисле резюме проведеного заняття.

Ключова цінність Gemini API у системах фідбеку полягає у здатності моделі агрегації та узагальнення текстового масиву. Замість лінійного перегляду десятків коментарів, викладач отримує згенерований текст, в якому алгоритм самостійно виділяє головні плюси та мінуси навчального процесу на основі відповідей групи. Така форма представлення дозволяє швидко діагностувати емоційний фон та загальні тенденції, перетворюючи розрізнені суб'єктивні думки на структурований методичний зворотний зв'язок.

Додатковою перевагою використання Gemini API є можливість оперативної генерації аналітики на запит. У разі накопичення достатньої кількості нових відгуків викладач може ініціювати процес аналізу безпосередньо з адміністративної панелі. Інтеграція підтримує асинхронну взаємодію, що формує підхід «інтелектуального асистента», який завжди використовує найактуальніший зріз бази даних для формування висновків.

З погляду реалізації інтеграція Gemini має методичну перевагу як хмарний NLP-сервіс із низьким порогом входу. Вона не вимагає розгортання власних нейромереж чи наявності складних обчислювальних потужностей на стороні сервера. Взаємодія здійснюється через стандартні HTTP-запити (REST API) з передачею даних у форматі JSON, де логіка аналізу повністю визначається декларативним текстовим промптом, наприклад, системною інструкцією виступати в ролі «помічника викладача».

З точки зору якості аналітики LLM-моделі підтримують різні способи обробки результатів: окрім базової суммаризації, вони можуть використовуватися для аналізу тональності, категоризації проблем за темами або генерації рекомендацій щодо покращення курсу. У подальшому розвитку системи це відкриває шлях до автоматичного тегування коментарів та побудови динамічних дашбордів на основі якісних текстових показників.

Водночас застосування Google Gemini API має й певні обмеження. Оскільки обробка відбувається на зовнішніх серверах, система є залежною від стабільності мережі та лімітів провайдера. Наприклад, можливе перевищення квоти запитів (помилка HTTP 429 Too Many Requests), що вимагає імплементації на стороні клієнта надійних механізмів повторних спроб (retry logic) та обробки тайм-аутів для забезпечення стабільного UX. Крім того, якість результату напряму залежить від точності сформульованого промпту (prompt engineering), а згенероване резюме має розглядатися як допоміжний аналітичний інструмент.

Отже, Google Gemini API є обґрунтованим вибором для текстової аналітики фідбеків у навчальних ретроспективах, оскільки поєднує простоту REST-інтеграції, високу швидкість обробки та здатність до глибокого розуміння контексту. Використання цього інструменту виводить систему за межі звичайного збору кількісних оцінок, перетворюючи її на платформу, що підтримує прийняття рішень на основі автоматизованого семантичного аналізу.

РОЗДІЛ 3. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ

3.1 Архітектурна модель застосунку

Архітектурну модель розробленого вебзастосунку доцільно визначати як клієнт-хмарну (client–cloud), побудовану на принципах serverless із використанням платформи Backend-as-a-Service (BaaS). У межах такої архітектури ключові функції прикладної логіки реалізуються на стороні клієнта (у браузері), тоді як зберігання даних, синхронізація та доступність забезпечуються хмарними сервісами. Такий підхід є методично виправданим оскільки знижує складність розгортання, мінімізує потребу в адмініструванні серверів і водночас створює основу для подальшого масштабування та функціонального розвитку.

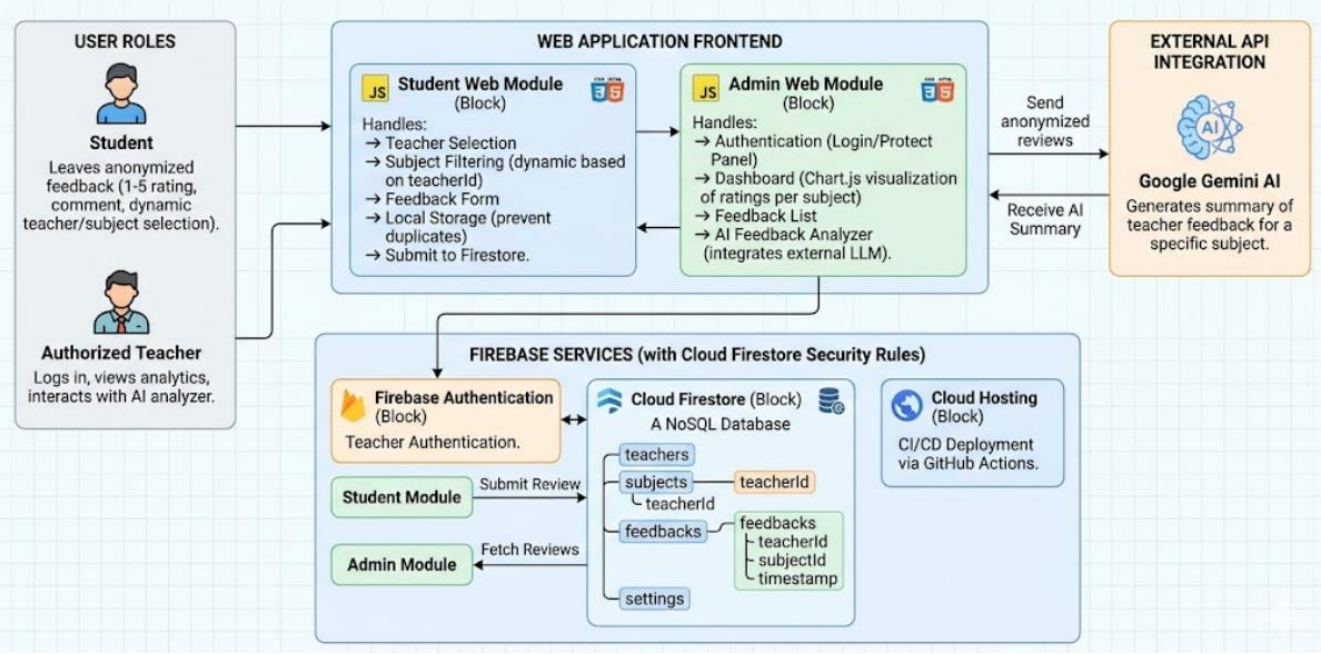

Рисунок 3.1. Схема взаємодії компонентів системи збору та аналізу студентських відгуків

Діаграма на рис. 3.1 відображує структурну архітектуру застосунку, яка охоплює клієнтський рівень, рівень даних у хмарі, інфраструктурний рівень розгортання а також роботу з API Google Gemini AI. Клієнтський рівень представлений вебінтерфейсом, реалізованим засобами HTML, CSS та JavaScript, і

включає два функціонально відокремлені модулі: студентський та викладацький. Студентський модуль забезпечує введення відгуку у форматі (оцінка за шкалою 1–5, коментар, прив’язка до викладача і предмету), виконує первинну перевірку коректності введених даних і відображає користувачеві системні повідомлення щодо статусу збору. Викладацький модуль призначений для керування процесом ретроспективи, зокрема увімкнення та вимкнення прийому відгуків, а також для перегляду результатів у вигляді агрегованих показників і візуалізацій. Для підвищення наочності результатів у структурі клієнтського рівня використовується модуль графічного подання даних, а саме стовпчикова діаграма розподілу оцінок, що сприяє швидкому аналізу та інтерпретації фідбеків. Для загального аналізу усіх відгуків, а також аналізу за критерієм обраного предмету є функція оброблення відгуків з допомогою Google Gemini API.

Рівень даних реалізовано на основі хмарної бази Cloud Firestore, яка забезпечує документно-колекційну модель зберігання. З позиції предметної області доцільно розрізняти дві категорії інформації: операційні дані (власне відгуки) та конфігураційні дані (параметри, що визначають режим роботи системи). Операційні дані зберігаються у вигляді окремих записів із мінімально необхідними атрибутами як оцінка, текстовий коментар, часовий маркер. Конфігураційні дані містять, зокрема, параметр, який визначає, чи дозволено приймати відгуки в поточний момент. Така декомпозиція дозволяє відокремити дані користувацьких взаємодій від налаштувань керування та підтримувати прозору логіку доступності функцій [1].

Інфраструктурний рівень представлений сервісом Firebase Hosting, що забезпечує публікацію клієнтської частини як статичного вебресурсу. У межах архітектурної моделі окремого значення набувають механізми контролю доступу, які реалізуються через правила безпеки бази даних (Security Rules), а в розширеному варіанті — через механізми автентифікації та ролей. З методичної точки зору це є критичним для чіткого розмежування прав: студент має отримувати доступ до подання відгуку у дозволений часовий інтервал, тоді як викладацький модуль повинен бути захищений від несанкціонованого використання.

Логіка взаємодії компонентів описується як послідовність потоків даних. Під час подання відгуку клієнтський модуль спочатку зчитує конфігураційний стан системи, визначаючи доступність функції збору. У разі дозволеного режиму студент заповнює форму, після чого виконується перевірка даних і запис у Firestore із серверною часовою міткою. Для базового контролю повторних подань використовується локальна фіксація факту відправлення на рівні браузера, що забезпечує обмеження одного відгуку з одного середовища перегляду [24, 26]. В адміністративному сценарії панель викладача отримує дані з бази, здійснює агрегацію показників, наприклад, середнє значення та частотний розподіл оцінок і представляє результати у табличній та графічній формах. Окремим керувальним потоком є зміна стану «збір увімкнено/вимкнено», що здійснюється через запис відповідного параметра у конфігураційний документ і відображається на доступності студентського інтерфейсу.

Запропонована архітектурна модель відзначається високими нефункціональними характеристиками, які є критичними для навчального середовища: масштабованістю завдяки хмарній інфраструктурі Firebase, оперативністю доступу до даних та надійністю автоматизованого розгортання (завдяки CI/CD пайплайну на Firebase Hosting). Висока підтримуваність системи досягається чітким рольовим і компонентним розділенням клієнтської частини.

Важливою особливістю реалізованої архітектури є вже впроваджений багаторівневий контроль доступу. Захист адміністративної панелі забезпечується механізмами Firebase Authentication, а безпека операцій із базою даних на серверному рівні суворо регламентується політиками Firestore Security Rules. Більше того, базова модель даних була розширена для підтримки ключових контекстних сутностей навчального процесу: кожен відгук має жорстку реляційну прив'язку до конкретної навчальної дисципліни та викладача. Важливим доповненням архітектури стала інтеграція засобів штучного інтелекту (великих мовних моделей на базі Google Gemini API) для автоматизованого узагальнення текстових коментарів.

У сукупності ці архітектурні та технологічні рішення значно підвищують достовірність зібраних даних, автоматизують складні аналітичні процеси та забезпечують повну відповідність розробленої системи сучасним вимогам реальної експлуатації.

3.2 Логіка взаємодії з Firebase

Логіку взаємодії вебзастосунку з платформою Firebase доцільно описувати як сукупність узгоджених клієнтсько-хмарних операцій, що виконуються через SDK Firebase та забезпечують керованість процесу збору фідбеку, збереження даних і формування аналітики. В основі такої взаємодії лежить принцип: клієнт виконує прикладну логіку, а хмара забезпечує зберігання, синхронізацію та доступність, що є характерним для моделей Backend-as-a-Service і serverless-архітектур.

У системі виділяються два основні типи даних: операційні та конфігураційні. Операційні дані формуються під час подання відгуків і зберігаються у вигляді окремих документів у колекції, де кожен запис містить мінімально необхідні атрибути: оцінка, коментар, час подання. Конфігураційні дані відповідають за керування режимом роботи застосунку і зберігаються в окремому документі налаштувань. Такий поділ забезпечує прозорість моделі даних: керувальні параметри не змішуються з масивом користувацьких відповідей і можуть змінюватися незалежно від них.

Взаємодія студентського інтерфейсу з Firebase реалізує послідовність операцій, орієнтованих на контроль доступності та коректне збереження відгуку. На початковому етапі клієнтський модуль ініціює зчитування конфігураційного документа, щоб визначити стан системи — чи дозволено приймати відгуки в поточний момент. Якщо збір вимкнено, інтерфейс переходить у режим блокування: форма подання деактивується, а користувачеві відображається відповідне повідомлення. Якщо збір активний, користувач отримує доступ до введення даних. Після заповнення форми система виконує первинну валідацію: перевірку діапазону оцінки та коректності полів, після чого здійснює операцію запису нового документа у хмарну базу даних. Час подання фіксується серверною часовою міткою, що

мінімізує вплив локальних налаштувань пристрою та забезпечує уніфікований часовий контекст для подальшої аналітики.

Для базового контролю повторного надсилання відгуку застосовується локальний механізм на рівні браузера, який фіксує факт успішної відправки. У разі повторної спроби подання в межах одного середовища перегляду інтерфейс блокує форму та інформує користувача, що відгук уже був надісланий. Важливо зазначити, що такий підхід є клієнтським і має обмежену надійність, оскільки залежить від стану локального сховища, однак він виконує роль простого бар’єра від випадкових повторних натискань і дублювання відповідей.

Адміністративний інтерфейс взаємодіє з Firebase у двох взаємопов’язаних напрямах: керування режимом збору та аналітичний перегляд результатів. Керування реалізується через операції читання і запису конфігураційного документа: панель відображає поточний стан (увімкнено/вимкнено), а зміна перемикача ініціює оновлення відповідного параметра в базі даних. Таким чином, система підтримує централізоване керування доступністю збору фідбеку, що забезпечує узгодженість поведінки всіх клієнтських сесій.

Аналітичний перегляд ґрунтується на отриманні набору операційних даних із колекції відгуків. За наявності механізмів потокового оновлення адміністративна панель може підтримувати актуальний стан без ручного оновлення сторінки: при надходженні нових документів автоматично перераховуються агреговані показники, зокрема середнє значення оцінки та частотний розподіл за шкалою 1–5. Отримані результати представлено у табличному вигляді та у вигляді графічної візуалізації. Додатково під час відображення текстових коментарів застосовується екранування спеціальних символів, що зменшує ризики некоректного рендерингу або небажаних вставок у вебінтерфейсі.

З погляду програмної організації взаємодія з Firebase реалізується через типові CRUD-операції (Create/Read/Update) над документами Firestore: створення нового відгуку, читання налаштувань і масиву відгуків, оновлення параметра «активність збору». У межах академічного опису важливо підкреслити, що правильність і безпечність такої взаємодії у реальній експлуатації забезпечуються

суворими політиками доступу на стороні хмари (Firestore Security Rules) та обов'язковими механізмами автентифікації викладачів (Firebase Authentication). Такий підхід гарантує надійний захист системи, унеможливлюючи несанкціонований доступ до адміністративних функцій, перегляду масиву відгуків та редагування конфігураційних даних.

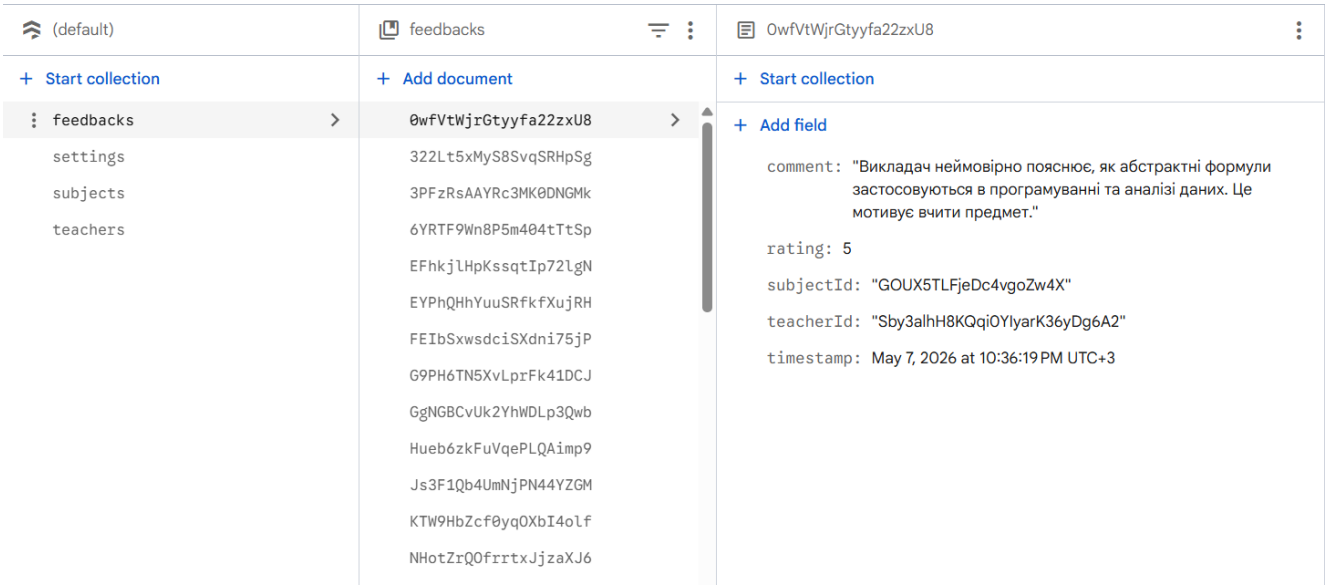

Рисунок 3.2. Структура колекції feedbacks у базі даних Firestore

Отже, логіка взаємодії з Firebase у даному застосунку забезпечує повний цикл роботи системи фідбеку: від керування доступністю збору та подання відгуків — до їх централізованого збереження, як зображено на рис. 3.2 і відображення агрегованих результатів. Така модель є технологічно обґрунтованою для навчальних вебсистем, оскільки поєднує простоту інтеграції, масштабованість хмарної інфраструктури та можливість розвитку в напрямі більш строгих механізмів контролю й аналітики.

3.3 Створення веб-інтерфейсу для студента та адміністратора

Проєктування та реалізація веб-інтерфейсів у системі збору фідбеку для навчальних ретроспектив доцільно здійснювати на основі принципу розмежування ролей і відповідних сценаріїв використання. У межах застосунку передбачено два

основні типи користувачів — студент та викладач, що зумовлює потребу у двох логічно відокремлених інтерфейсах. Такий підхід підвищує керованість системи, забезпечує зрозумілість взаємодії та створює передумови для коректного контролю доступу до функцій.

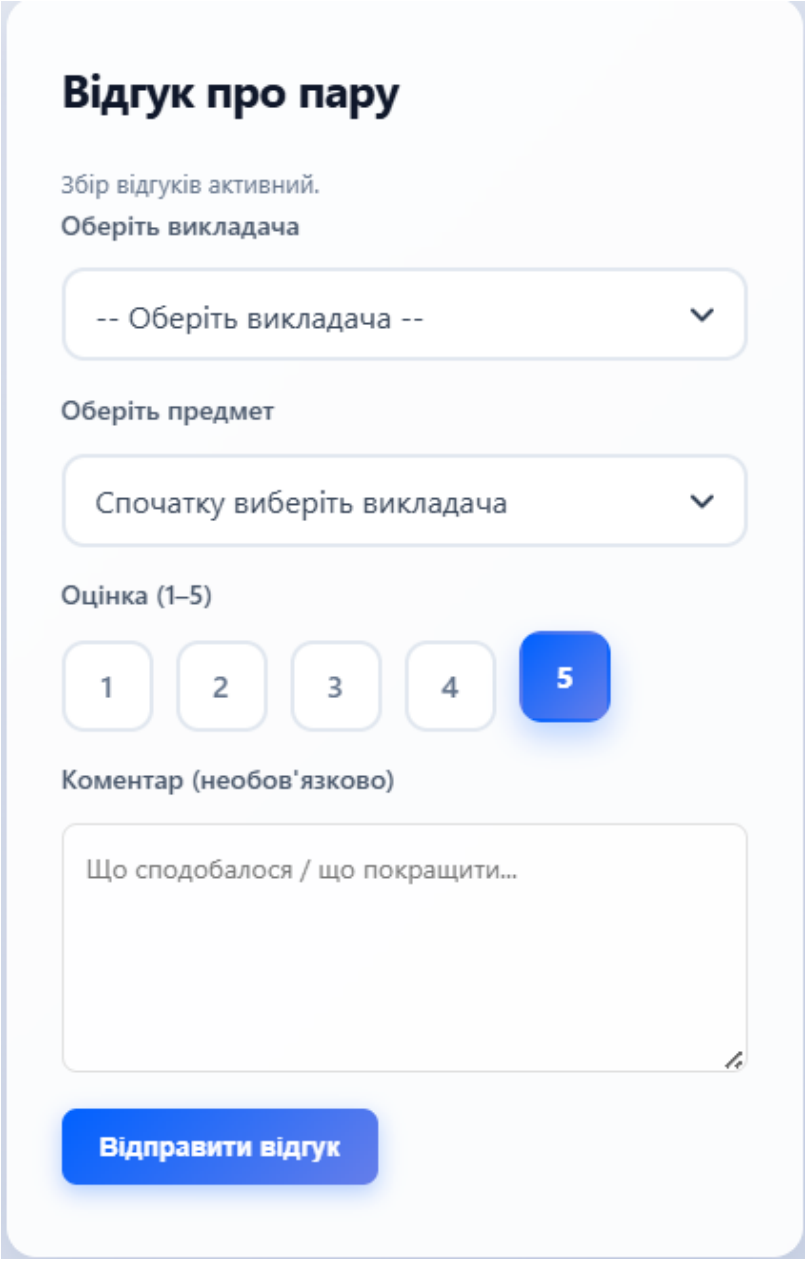

Рисунок 3.3. Сторінка студентського веб-інтерфейсу

На рис. 3.3 зображено студентський веб-інтерфейс, який поєднує сучасний мінімалістичний дизайн із розширеною логікою взаємодії. Головною особливістю

цієї форми є забезпечення чіткого предметного контексту: перед тим як залишити відгук, користувач повинен послідовно обрати викладача та навчальну дисципліну за допомогою випадаючих списків. Важливою UX-деталлю є динамічність інтерфейсу — список предметів залишається неактивним і формується лише після ідентифікації конкретного викладача, що мінімізує ризик помилкового вибору.

Безпосередньо для збору зворотного зв'язку передбачено два елементи: інтерактивну шкалу оцінювання у вигляді кнопок від 1 до 5 (що зручно для використання на мобільних пристроях) та необов’язкове текстове поле для розгорнутого коментаря. Крім того, у верхній частині модуля розміщено динамічний індикатор статусу системи (наприклад, «Збір відгуків активний»). Завдяки зв'язку з хмарною базою даних, якщо викладач вимикає прийом відповідей, форма автоматично блокується, запобігаючи надсиланню даних поза визначеним часовим вікном ретроспективи.

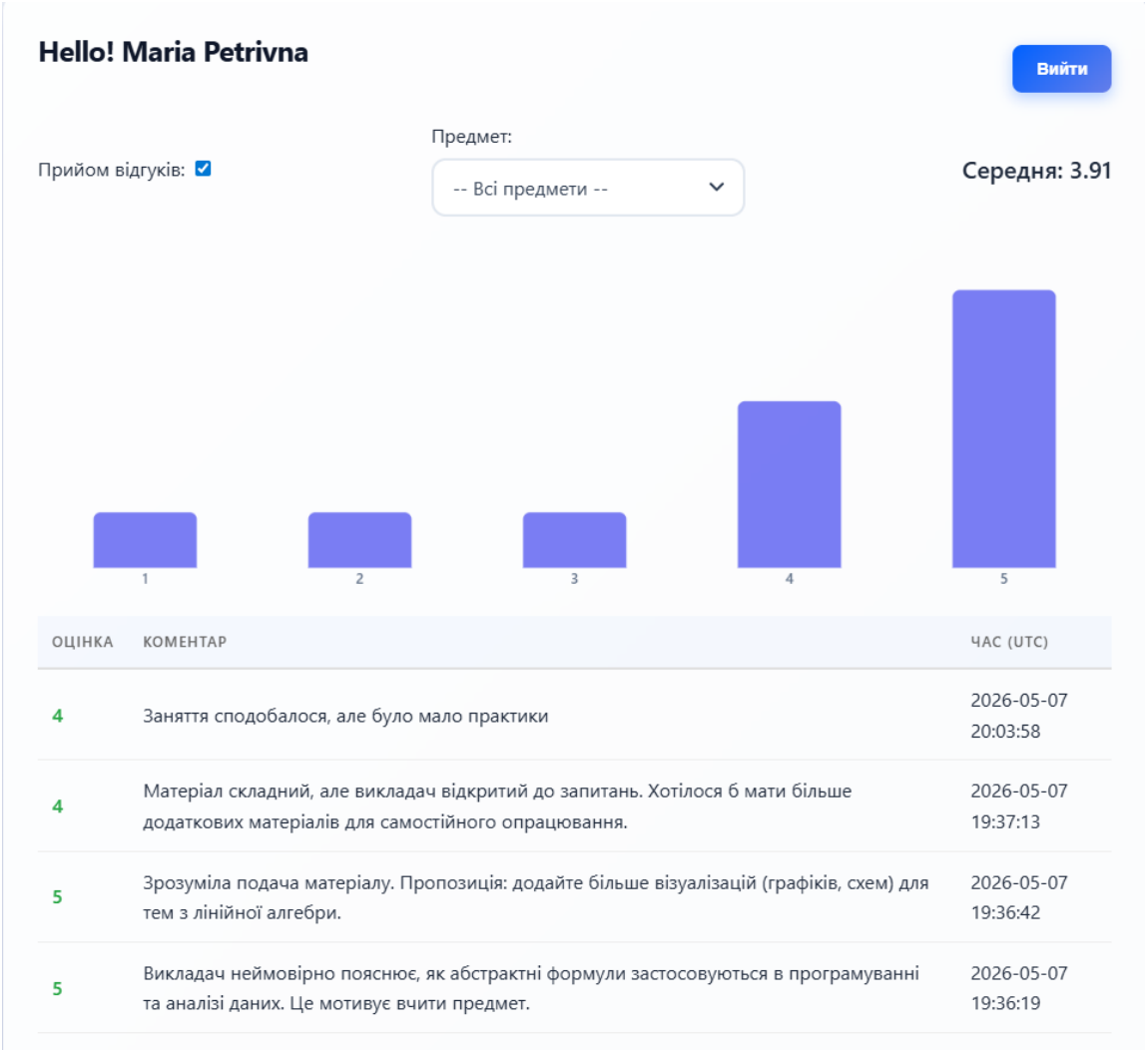

Рисунок 3.4. Сторінка веб-інтерфейсу викладача

Адміністративний веб-інтерфейс, фрагмент якого зображено на рис. 3.4, проєктується як комплексна панель керування та аналізу, що концентрує функції модерації ретроспективи та інтерпретації результатів. Змістовно він включає базові інструменти керування, зокрема механізм увімкнення/вимкнення збору відгуків, що дозволяє організувати ретроспективу в межах суворо визначеного часово-організаційного вікна.

Інтерфейс надає розширене багаторівневе аналітичне подання зібраних даних. Кількісні агреговані показники (середнє значення оцінки та частотний розподіл за шкалою) візуалізуються за допомогою динамічних діаграм, що підвищує наочність результатів і спрощує швидке виявлення загальних тенденцій у групі. Рівень деталізації забезпечується табличним представленням, де кожен відгук виводиться в контексті виставленої оцінки, тексту коментаря та точного часу подання.

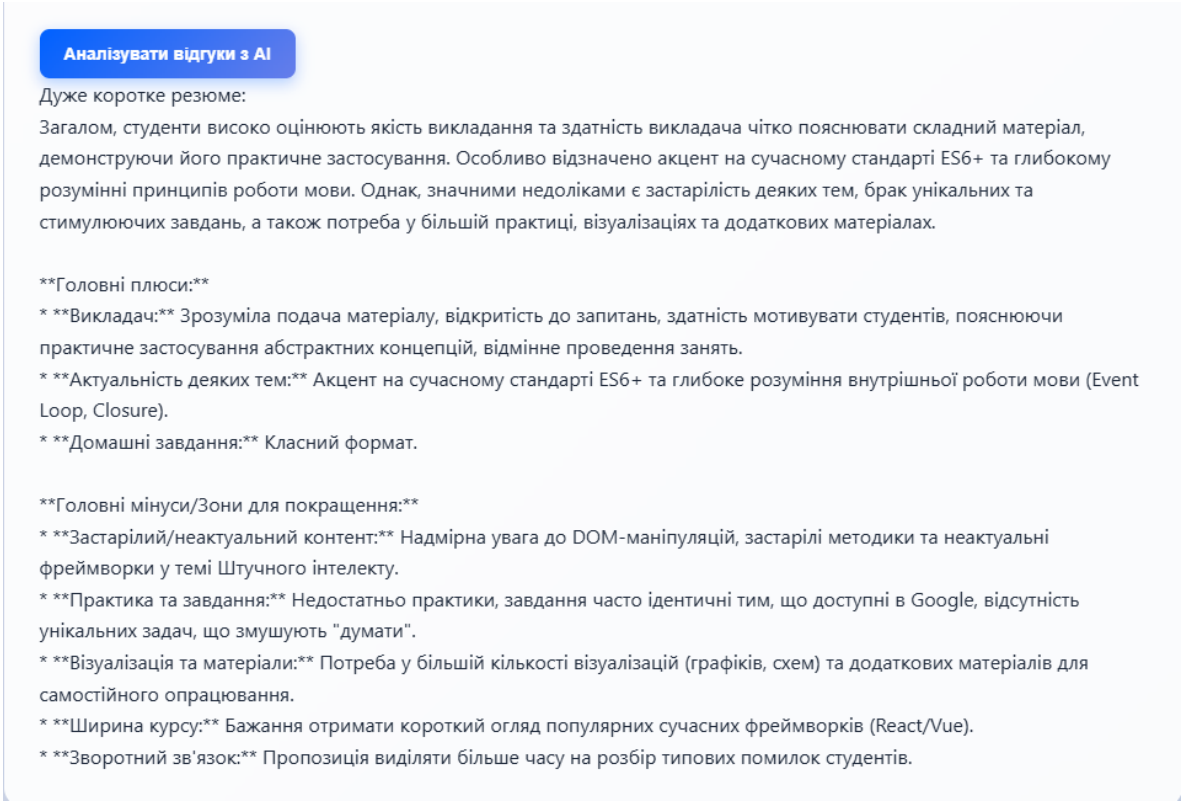

Рисунок 3.5. Сторінка веб-інтерфейсу викладача з елементом «Аналізувати відгуки з AI»

Ключовим компонентом інтерфейсу є інтегрований модуль автоматизованого аналізу тексту, доступний за допомогою кнопки «Аналізувати відгуки з AI» зображений на рис. 3.5. Цей інструмент, побудований на базі великих мовних

моделей, вирішує проблему ручного опрацювання великих масивів неструктурованого тексту. Як видно з рисунка, після ініціалізації запиту алгоритм самостійно аналізує всі зібрані коментарі та генерує структуроване текстове резюме.

Згенерований звіт зручно класифікує зворотний зв'язок на дві основні категорії: «Головні плюси» (наприклад, оцінка якості викладання, актуальність тем, формат домашніх завдань) та «Головні мінуси / Зони для покращення» (ідентифікація застарілого контенту, брак практичних завдань чи візуалізацій). Така автоматична семантична кластеризація дозволяє викладачу швидко переходити від читання окремих суб'єктивних думок до прийняття обґрунтованих педагогічних та організаційних рішень на основі узагальнених результатів.

На рівні реалізації веб-інтерфейсів доцільно застосовувати стандартні вебтехнології HTML, CSS та JavaScript як базові засоби формування структури сторінок, їх стилізації та реалізації клієнтської логіки. HTML забезпечує семантичне подання елементів: форма, кнопки, таблиці, блоки повідомлень, CSS — єдину візуальну стилістику та адаптивність інтерфейсу, а JavaScript — поведінкову частину: обробку подій, валідацію, динамічне оновлення інтерфейсу та взаємодію з хмарною інфраструктурою (див. Додаток А) [2, 13]. Така технологічна основа забезпечує прозорість коду, відносну простоту супроводу та можливість поступового нарощування функціональності без необхідності складних інструментів збірки.

Важливим аспектом створення обох інтерфейсів є забезпечення коректного UX і зрозумілої логіки взаємодії. Для студентського модуля пріоритетом виступають простота й швидкість: мінімальна кількість полів, відсутність зайвих дій, чіткі підказки та статуси. Для адміністративного модуля пріоритетом є інформативність і контроль: структурована аналітика, стабільні оновлення даних, зрозуміле керування режимом збору, а також безпечне відображення текстових коментарів. У перспективі розвитку системи необхідним є посилення механізмів доступу (автентифікація, ролі) для гарантування того, що адміністративний інтерфейс доступний лише уповноваженим користувачам.

Таким чином, створення веб-інтерфейсу для студента та адміністратора в межах системи збору фідбеку реалізує принцип рольового поділу функцій: студент отримує простий інструмент подання відгуку, а адміністратор — засоби керування ретроспективою та аналізу результатів. Це забезпечує логічну цілісність процесу збору, підвищує зручність користування та створює основу для подальшого розширення системи, зокрема в напрямі більш глибокої аналітики, інтеграцій та посилення безпеки.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ВЕБЗАСТОСУНКУ

4.1 Тестування клієнтського модуля подання відгуків та логіки взаємодії зі студентом

Першим етапом перевірки якості розробленого програмного продукту стало комплексне функціональне тестування клієнтського модуля подання відгуків, що є ключовою точкою взаємодії зі студентською аудиторією. Головною метою цього етапу було підтвердження безперебійності процесу збору даних, перевірка механізмів валідації та оцінка стійкості інтерфейсу до некоректних дій користувача. Враховуючи, що система передбачає глибоку контекстуалізацію фідбеку, окрема увага приділялася тестуванню алгоритмів динамічного завантаження списків. Під час виконання тестових сценаріїв було перевірено ініціалізацію сторінки, яка автоматично звертається до колекції викладачів у базі даних Cloud Firestore та формує первинний випадаючий список. Тестування підтвердило, що до моменту вибору конкретного викладача поле вибору навчальної дисципліни залишається заблокованим і відображає відповідне інформаційне повідомлення, що унеможливлює порушення реляційної цілісності даних на етапі введення. Після ідентифікації викладача система коректно ініціює функцію фільтрації та підвантажує виключно ті предмети, які закріплені за обраним педагогом.

Наступним кроком стала перевірка механізмів валідації форми та запобігання дублюванню відповідей. Було проведено серію тестів із навмисним пропуском обов'язкових полів. Результати показали, що при спробі відправити форму без вказаної оцінки за шкалою від 1 до 5, або без обраного викладача чи предмета, система миттєво перериває виконання HTTP-запиту та виводить користувачеві локалізоване попередження. Для перевірки алгоритму обмеження повторних відповідей застосовувався метод імітації сесій у різних вкладках браузера. Система після успішного відправлення першого відгуку генерує унікальний складений ключ, що включає ідентифікатори обраного викладача та предмета, і зберігає його у локальному сховищі браузера [24, 26]. На рис 4.1 тестування підтвердило надійність цього підходу: при повторному відкритті сторінки або спробі надіслати

ще один відгук для тієї самої пари «викладач-предмет», інтерфейс автоматично блокує всі елементи форми, роблячи кнопку відправки неактивною.

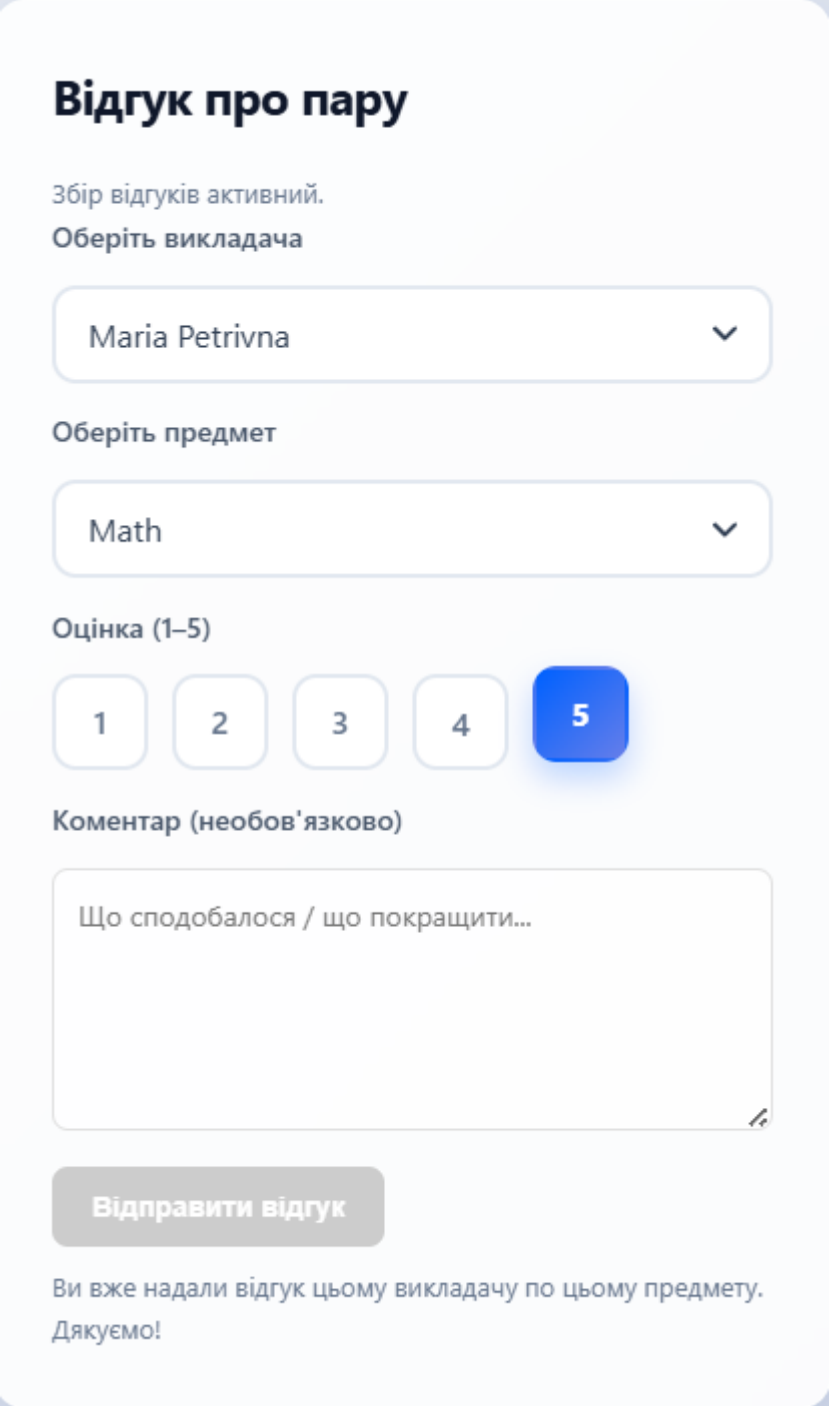

Рисунок 4.1. Автоматичне блокування форми для запобігання повторних надсилань

Окремим аспектом перевірки клієнтської частини стало тестування реакції системи на зміну глобальних конфігураційних параметрів та перевірка

адаптивності дизайну. Було імітовано сценарій, за якого викладач в адміністративній панелі деактивує збір відгуків. Студентський модуль, що використовує асинхронні запити до документа налаштувань під час завантаження, коректно розпізнає стан системи: форма автоматично блокується, а індикатор статусу змінюється на повідомлення про тимчасове припинення прийому відповідей. Для забезпечення безбар'єрного доступу з мобільних пристроїв також було проведено UI-тестування за допомогою інструментів розробника у браузері. Перевірка підтвердила, що завдяки використанню гнучких сіток та сучасних CSS-селекторів, елементи введення, зокрема інтерактивна шкала оцінок, коректно масштабуються та залишаються зручними для взаємодії на екранах різної роздільної здатності.

4.2 Тестування адміністративного модуля, політик безпеки та інструментів інтелектуальної аналітики

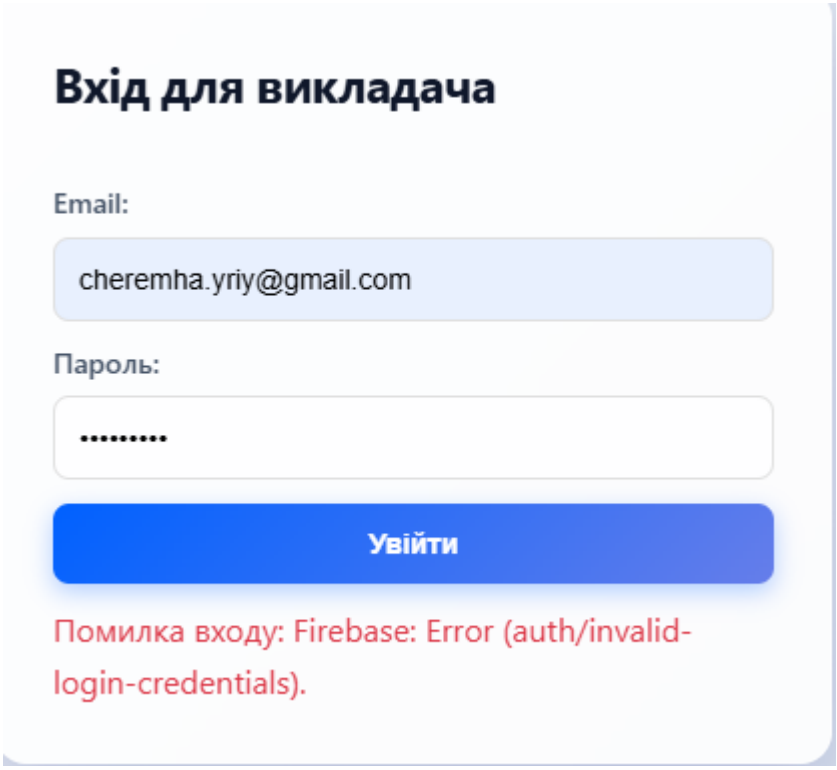

Рисунок 4.2. Форма авторизації на сторінку викладача

Другим напрямом перевірки стало тестування адміністративного модуля, який об’єднує функції керування ретроспективою, візуалізації даних та їх

інтелектуальної обробки. Пріоритетним завданням на цьому етапі була валідація політик безпеки та механізмів авторизації, оскільки система оперує конфіденційними даними оцінювання. Тестування доступу зображено на рис 4.2 підтвердило, що перегляд та аналіз відгуків стають доступними лише після успішної автентифікації користувача через сервіс Firebase Authentication.

Для перевірки надійності захисту використовувалося програмне середовище Postman, за допомогою якого симулювалися прямі HTTP-запити до REST API бази даних від імені неавторизованих користувачів. Спроби такого прямого доступу до колекції відгуків успішно блокуються на серверному рівні, сервер повертає помилку доступу Missing or insufficient permissions рис. 4.3 завдяки налаштованим правилам Cloud Firestore Security Rules. Застосовані декларативні політики гарантують, що читання масиву відгуків та оновлення статусу ретроспективи можливі виключно за умови наявності валідного токена доступу конкретного викладача.

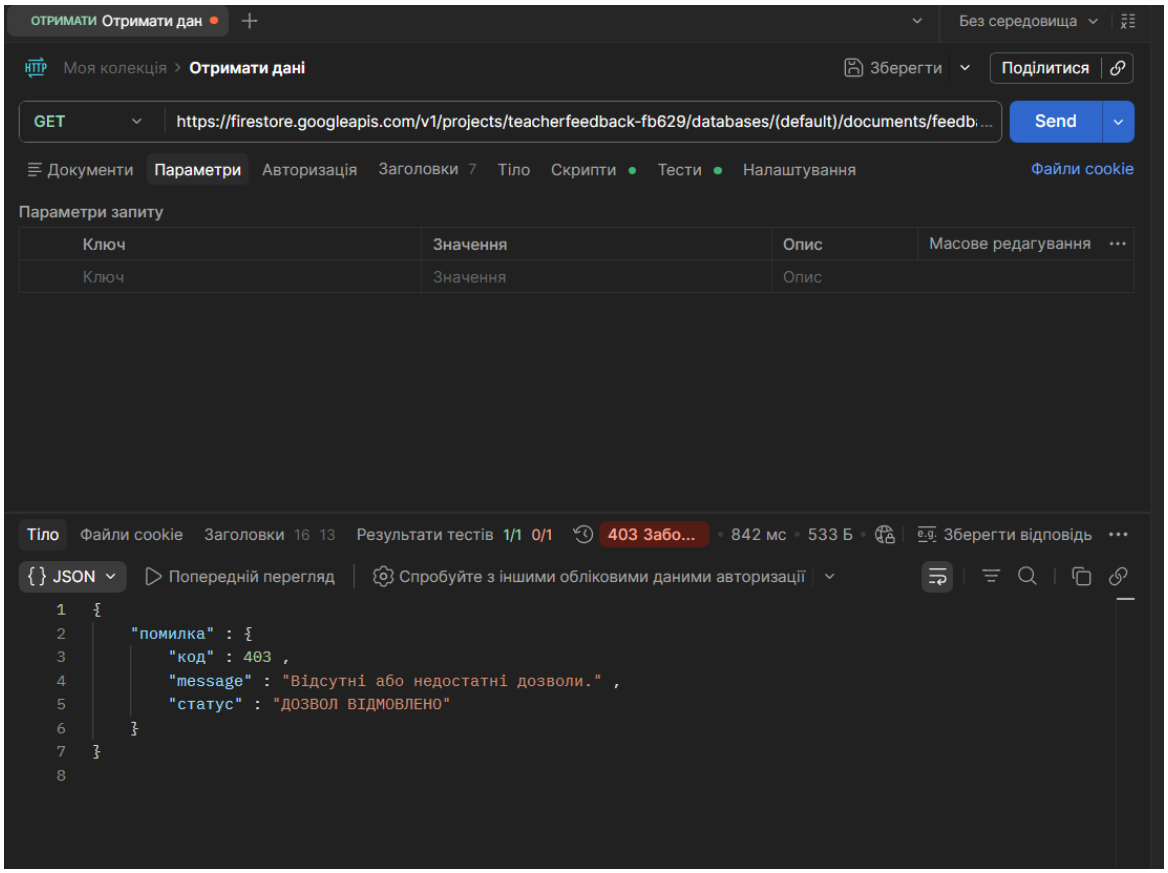

Рисунок 4.3. Перевірка політик безпеки бази даних Firestore за допомогою інструменту Postman

Важливим компонентом тестування адміністративної панелі стала перевірка інструментів візуалізації та динамічного фільтрування. Було створено масив із тестових даних, що симулювали високу активність студентів. Завдяки впровадженню методу потокового прослуховування бази даних, адміністративний інтерфейс без перезавантаження сторінки оновлює показники при надходженні нових записів. Також було перевірено логіку фільтрації: при зміні навчальної дисципліни у випадаючому списку система безпомилково перераховує середній бал, перерозподіляє графік та оновлює таблицю коментарів, відображаючи дані виключно для обраного контексту.

На завершальному етапі було проведено комплексне тестування інтеграції з Google Gemini API, що забезпечує автоматизовану сумаризацію студентських коментарів. Перевірка передбачала надсилання до великої мовної моделі масивів тексту різної складності та тональності. Тести продемонстрували, що система успішно агрегує зібрані коментарі, формує структурований промпт та повертає зв'язне аналітичне резюме, виділяючи основні переваги та недоліки проведеного заняття. Особливу увагу було приділено тестуванню стійкості до відмов. Для цього було штучно імітовано ситуацію перевищення ліміту дозволених запитів до API (помилка HTTP 429 Too Many Requests). Замість критичної зупинки виконання скрипта або виведення незрозумілих системних помилок, застосунок коректно перехоплює це виключення та інформує викладача про тимчасове перевантаження сервера з рекомендацією повторити спробу пізніше. Такий підхід гарантує безперервність роботи інтерфейсу та забезпечує позитивний користувацький досвід навіть в умовах нестабільної роботи зовнішніх хмарних сервісів.

ВИСНОВКИ

Завершення розробки та аналіз результатів кваліфікаційної роботи дозволяють стверджувати, що поставлена мета була повністю досягнута через проєктування та реалізацію комплексного вебзастосунку для збору й інтелектуального аналізу студентського фідбеку. Дослідження ролі зворотного зв’язку в освітньому процесі підтвердило, що впровадження спеціалізованих цифрових інструментів для проведення навчальних ретроспектив є критично важливим чинником підвищення якості викладання та формування студентоцентрованого середовища. У ході роботи було обґрунтовано, що перехід від епізодичних опитувань до системного збору даних дозволяє викладачам оперативно адаптувати методичні підходи відповідно до реальних потреб здобувачів освіти. Порівняльний аналіз існуючих рішень продемонстрував необхідність створення власного інструменту, який забезпечує глибшу контекстуалізацію даних та повну власність закладу над накопиченою інформацією.

Практична реалізація застосунку базується на сучасній архітектурній моделі, що поєднує гнучкість клієнтських технологій HTML, CSS та JavaScript із потужністю хмарної інфраструктури Firebase. Створена система забезпечує розмежування ролей через два функціонально відокремлені модулі. Студентський модуль дозволяє здійснювати подання відгуку з високою точністю прив’язки до освітнього контексту, що реалізується через механізм вибору викладача та динамічне підвантаження відповідних навчальних дисциплін. Важливим технічним рішенням стало впровадження засобів контролю унікальності відповідей на основі локального сховища браузера, що в поєднанні з серверними часовими мітками Firestore забезпечує достовірність та впорядкованість статистичних даних.

Окрему увагу в роботі приділено безпеці та аналітичному потенціалу системи. Адміністративна панель викладача захищена багаторівневою системою контролю доступу, що включає автентифікацію користувачів та декларативні

правила безпеки Cloud Firestore, які унеможливлюють несанкціоноване втручання в дані. Аналітичний блок системи виходить за межі простої агрегації кількісних показників завдяки інтеграції великих мовних моделей через Google Gemini API. Це дозволило реалізувати функцію автоматизованого узагальнення текстових коментарів, яка перетворює масив неструктурованих відгуків у змістовне резюме з виділенням ключових переваг та зон для покращення навчального процесу. Візуалізація результатів засобами бібліотеки Chart.js забезпечує наочність розподілу оцінок, дозволяючи викладачу миттєво діагностувати групові реакції на освітній контент.

Узагальнюючи результати дослідження, можна констатувати, що розроблений вебзастосунок є завершеним програмним продуктом і готовий до експлуатації в реальних умовах. Виконана робота має практичне значення для викладачів, які прагнуть автоматизувати процеси збору фідбеку та отримати глибше розуміння навчального досвіду студентів. Подальший розвиток системи вбачається у розширенні аналітичних інструментів для відстеження динаміки змін у часі, впровадженні функцій експорту звітів у стандартизовані формати та інтеграції механізмів швидкого доступу через QR-коди, що сприятиме ще більшій залученості студентської аудиторії до покращення освітніх практик.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Add data to Cloud Firestore. *Firebase Documentation*. URL: https://firebase.google.com/docs/firestore/manage-data/add-data (дата звернення: 05.10.2025).

2. Add Firebase to your JavaScript project. *Firebase*. URL: https://firebase.google.com/docs/web/setup (дата звернення: 03.12.2025).

3. Bar Chart. *Chart.js Documentation*. URL: https://www.chartjs.org/docs/latest/charts/bar.html (дата звернення: 12.10.2025).

4. CSS Reference. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference (дата звернення: 01.11.2025).

5. Feedback activity. *MoodleDocs*. URL: https://docs.moodle.org/en/Feedback_activity (дата звернення: 06.12.2025).

6. Firebase Authentication. *Firebase Documentation*. URL: https://firebase.google.com/docs/auth (дата звернення: 24.10.2025).

7. Firebase developer documentation. *Firebase*. URL: https://firebase.google.com/docs (дата звернення: 30.11.2025).

8. Firebase Hosting. *Firebase Documentation*. URL: https://firebase.google.com/docs/hosting (дата звернення: 13.12.2025).

9. Firebase Security Rules. *Firebase Documentation*. URL: https://firebase.google.com/docs/rules (дата звернення: 08.10.2025).

10. Get started with Cloud Firestore Security Rules. *Firebase Documentation*. URL: https://firebase.google.com/docs/firestore/security/get-started (дата звернення: 13.12.2025).

11. Get started with Cloud Firestore. *Firebase Documentation*. URL: https://firebase.google.com/docs/firestore/quickstart (дата звернення: 10.11.2025).

12. Get Started with Firebase Authentication on Websites. *Firebase Documentation*. URL: https://firebase.google.com/docs/auth/web/start (дата звернення: 02.12.2025).

13. Hattie J., Timperley H. The Power of Feedback. *Review of Educational Research, 2007*. URL: https://doi.org/10.3102/003465430298487 (дата звернення: 21.10.2025).

14. How do I create a survey in my course? *Instructure Community (Canvas Instructor Guide)*. URL: https://community.canvaslms.com/t5/Instructor-Guide/How-do-I-create-a-survey-in-my-course/ta-p/782 (дата звернення: 16.11.2025).

15. How do I export my form responses? *Microsoft Support*. URL: https://support.microsoft.com/en-us/office/how-do-i-export-my-form-responses-fb0aee53-0fd9-43bf-9c48-af081b6895d5 (дата звернення: 04.11.2025).

16. The HTML Input element. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/input (дата звернення: 13.12.2025).

17. JavaScript Guide. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 30.10.2025).

18. Live Q&A. *Slido*. URL: https://www.slido.com/features-live-qa (дата звернення: 13.12.2025).

19. Microsoft Forms and Excel workbooks. *Microsoft Support*. URL: https://support.microsoft.com/en-us/office/microsoft-forms-and-excel-workbooks-c61b5751-8128-4434-abb0-246f5e2a3b01 (дата звернення: 25.11.2025).

20. Publish & share your form with responders. *Google Docs Editors Help*. URL: https://support.google.com/docs/answer/2839588?hl=en (дата звернення: 13.12.2025).

21. Skip Logic. *SurveyMonkey Help*. URL: https://help.surveymonkey.com/en/surveymonkey/create/skip-logic/ (дата звернення: 15.10.2025).

22. View & manage form responses. *Google Docs Editors Help*. URL: https://support.google.com/docs/answer/139706?hl=en (дата звернення: 27.10.2025).

23. Web Storage API. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата звернення: 13.12.2025).

24. What is branching logic? *Typeform Help Center*. URL: https://help.typeform.com/hc/en-us/articles/360029116392-What-is-branching-logic (дата звернення: 18.10.2025).

25. Window: localStorage property. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення: 13.12.2025).

26. Word Cloud. *Mentimeter*. URL: https://www.mentimeter.com/features/word-cloud (дата звернення: 01.12.2025).

ДОДАТКИ

ДОДАТОК А

Код student.js

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/9.22.2/firebase-app.js";
import {
  getFirestore,
  collection,
  addDoc,
  serverTimestamp,
  doc,
  getDoc,
  getDocs,
  query,
  where
} from "https://www.gstatic.com/firebasejs/9.22.2/firebase-firestore.js";

const firebaseConfig = {
  apiKey: "AIzaSyBj1xLyjUL-1r5JYLKMegwO9UWdKnmK9G8",
  authDomain: "teacherfeedback-fb629.firebaseapp.com",
  projectId: "teacherfeedback-fb629",
  storageBucket: "teacherfeedback-fb629.firebasestorage.app",
  messagingSenderId: "122928960162",
  appId: "1:122928960162:web:fae05648716078a4b5fdfb",
  measurementId: "G-NP8FX9095T"
};
```

```
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);

const statusEl = document.getElementById('status');
const form = document.getElementById('feedbackForm');
const msg = document.getElementById('msg');
const sendBtn = document.getElementById('sendBtn');
const teacherSelect = document.getElementById('teacher-select');
const subjectSelect = document.getElementById('subject-select');

let selectedTeacherId = null;
let selectedSubjectId = null;

async function loadTeachers() {
  try {
    const teachersCollection = collection(db, 'teachers');
    const snapshot = await getDocs(teachersCollection);

    teacherSelect.innerHTML =
      '<option value="">-- Оберіть викладача --</option>';

    snapshot.forEach(doc => {
      const teacher = doc.data();

      const option = document.createElement('option');
      option.value = doc.id;
      option.textContent = teacher.name || 'Без імені';
```

```
      teacherSelect.appendChild(option);
    });

  } catch (err) {
    console.error('Помилка завантаження викладачів:', err);
    statusEl.textContent =
      'Помилка: не вдалося завантажити список викладачів';
  }
}

async function loadSubjectsForTeacher(teacherId) {
  try {
    if (!teacherId) {
      subjectSelect.innerHTML =
        '<option value="">-- Спочатку виберіть викладача --</option>';

      selectedSubjectId = null;
      return;
    }

    const subjectsRef = collection(db, 'subjects');

    const q = query(
      subjectsRef,
      where('teacherId', '==', teacherId)
    );
```

```
    const snapshot = await getDocs(q);

    subjectSelect.innerHTML =
      '<option value="">-- Оберіть предмет --</option>';

    snapshot.forEach(doc => {
      const subject = doc.data();

      const option = document.createElement('option');
      option.value = doc.id;
      option.textContent = subject.name || 'Без назви';

      subjectSelect.appendChild(option);
    });

  } catch (err) {
    console.error('Помилка завантаження предметів:', err);

    subjectSelect.innerHTML =
      '<option value="">Помилка завантаження</option>';
  }
}

teacherSelect.addEventListener('change', (e) => {
  selectedTeacherId = e.target.value;
```

```
  loadSubjectsForTeacher(selectedTeacherId);

  selectedSubjectId = null;
  subjectSelect.value = '';

  checkFeedbackGiven();
});

subjectSelect.addEventListener('change', (e) => {
  selectedSubjectId = e.target.value;

  checkFeedbackGiven();
});

async function checkEnabled() {
  try {
    const settingsRef = doc(db, 'settings', 'meta');

    const snap = await getDoc(settingsRef);

    const enabled = snap.exists()
      ? snap.data().feedbackEnabled
      : true;

    checkFeedbackGiven();

    if (!enabled) {
```

```
      statusEl.textContent =
        'Збір відгуків тимчасово вимкнено.';

      form.querySelectorAll('input,textarea,button')
        .forEach(el => el.disabled = true);

      sendBtn.disabled = true;

    } else {
      statusEl.textContent =
        'Збір відгуків активний.';
    }

  } catch (e) {
    console.error(e);

    statusEl.textContent =
      'Не вдалося перевірити статус (перевірте конфіг).';
  }
}

function checkFeedbackGiven() {
  if (!selectedTeacherId || !selectedSubjectId) {
    return;
  }

  const feedbackKey =
```

```
    `feedbackGiven_${selectedTeacherId}_${selectedSubjectId}`;

  const given = localStorage.getItem(feedbackKey);

  if (given) {
    form.querySelectorAll('input,textarea,button')
      .forEach(el => el.disabled = true);

    sendBtn.disabled = true;

    msg.textContent =
      'Ви вже надали відгук цьому викладачу по цьому предмету. Дякуємо!';

  } else {
    form.querySelectorAll('input,textarea,button')
      .forEach(el => el.disabled = false);

    sendBtn.disabled = false;

    msg.textContent = '';
  }
}

form.addEventListener('submit', async (e) => {
  e.preventDefault();

  if (!selectedTeacherId || !selectedSubjectId) {
```

```
  msg.textContent =
    'Виберіть викладача та предмет';

  return;
}

sendBtn.disabled = true;

const rating = Number(form.rating.value || 5);

const comment =
  document.getElementById('comment').value.trim();

if (!rating || rating < 1 || rating > 5) {
  msg.textContent =
    'Вкажіть оцінку від 1 до 5';

  sendBtn.disabled = false;

  return;
}

try {
  await addDoc(collection(db, 'feedbacks'), {
    rating,
    comment,
    teacherId: selectedTeacherId,
```

```
      subjectId: selectedSubjectId,
      timestamp: serverTimestamp()
    });

    msg.textContent =
      'Дякуємо! Ваш відгук відправлено.';

    form.reset();

    const feedbackKey =
      `feedbackGiven_${selectedTeacherId}_${selectedSubjectId}`;

    localStorage.setItem(feedbackKey, 'true');

    checkFeedbackGiven();

  } catch (err) {
    console.error(err);

    msg.textContent =
      'Помилка надсилання. Спробуйте пізніше.';
  }
});

checkEnabled();
loadTeachers();
```

Код admin.js

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/9.22.2/firebase-app.js";
import {
  getAuth, signInWithEmailAndPassword, signOut, onAuthStateChanged
} from "https://www.gstatic.com/firebasejs/9.22.2/firebase-auth.js";
import {
  getFirestore, collection, query, orderBy, onSnapshot, doc, getDoc, setDoc, where, getDocs
} from "https://www.gstatic.com/firebasejs/9.22.2/firebase-firestore.js";

const firebaseConfig = {
  apiKey: "AIzaSyBj1xLyjUL-1r5JYLKMegwO9UWdKnmK9G8",
  authDomain: "teacherfeedback-fb629.firebaseapp.com",
  projectId: "teacherfeedback-fb629",
  storageBucket: "teacherfeedback-fb629.firebasestorage.app",
  messagingSenderId: "122928960162",
  appId: "1:122928960162:web:fae05648716078a4b5fdfb",
  measurementId: "G-NP8FX9095T"
};

const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const db = getFirestore(app);

const loginContainer = document.getElementById('loginContainer');
const adminContent = document.getElementById('adminContent');
const loginForm = document.getElementById('loginForm');
const emailInput = document.getElementById('email');
```

```
const passwordInput = document.getElementById('password');
const loginError = document.getElementById('loginError');
const logoutBtn = document.getElementById('logoutBtn');
const adminTitle = document.getElementById('adminTitle');

const toggle = document.getElementById('toggle');
const avgEl = document.getElementById('avg');
const tbody = document.querySelector('#table tbody');
const chartCtx = document.getElementById('chart').getContext('2d');
const subjectFilter = document.getElementById('subject-filter');

let chart;
let currentComments = [];
let currentUser = null;
let allFeedbacks = [];
let teacherSubjects = [];

onAuthStateChanged(auth, async (user) => {
  if (user) {
    currentUser = user;

    try {
      const docSnap = await getDoc(doc(db, 'teachers', user.uid));
      if (docSnap.exists() && docSnap.data().name) {
        adminTitle.textContent = "Hello! " + docSnap.data().name;
      } else {
        adminTitle.textContent = "Панель викладача";
```

```
      }
    } catch (error) {
      console.error('Помилка завантаження імені викладача:', error);
      adminTitle.textContent = "Панель викладача";
    }

    loginContainer.style.display = 'none';
    adminContent.style.display = 'block';
    loadStatus();
    loadTeacherSubjects();
    startListening();
  } else {
    currentUser = null;
    adminTitle.textContent = "Відгуки";
    loginContainer.style.display = 'block';
    adminContent.style.display = 'none';
    stopListening();
  }
});

loginForm.addEventListener('submit', async (e) => {
  e.preventDefault();
  const email = emailInput.value;
  const password = passwordInput.value;

  try {
    await signInWithEmailAndPassword(auth, email, password);
```

```
    loginError.textContent = '';
  } catch (error) {
    loginError.textContent = 'Помилка входу: ' + error.message;
  }
});

logoutBtn.addEventListener('click', async () => {
  try {
    await signOut(auth);
  } catch (error) {
    console.error('Помилка виходу:', error);
  }
});

async function loadStatus() {
  const ref = doc(db, 'settings', 'meta');
  const snap = await getDoc(ref);
  const enabled = snap.exists() ? snap.data().feedbackEnabled : true;
  toggle.checked = !!enabled;
}

toggle.addEventListener('change', async () => {
  const ref = doc(db, 'settings', 'meta');

  try {
    await setDoc(ref, { feedbackEnabled: toggle.checked }, { merge: true });
  } catch (e) {
```

```
    console.error(e);
    alert('Не вдалося змінити статус');
  }
});

subjectFilter.addEventListener('change', () => {
  updateAnalytics();
});

async function loadTeacherSubjects() {
  if (!currentUser) return;

  try {
    const subjectsRef = collection(db, 'subjects');
    const q = query(subjectsRef, where('teacherId', '==', currentUser.uid));
    const snapshot = await getDocs(q);

    teacherSubjects = [];
    subjectFilter.innerHTML = '<option value="">-- Всі предмети --</option>';

    snapshot.forEach(doc => {
      teacherSubjects.push({ id: doc.id, name: doc.data().name });

      const option = document.createElement('option');
      option.value = doc.id;
      option.textContent = doc.data().name || 'Без назви';
      subjectFilter.appendChild(option);
```

```
    });

  } catch (err) {
    console.error('Помилка завантаження предметів:', err);
  }
}

let unsubscribe;

function startListening() {
  if (!currentUser) return;

  const q = query(
    collection(db, 'feedbacks'),
    where('teacherId', '==', currentUser.uid),
    orderBy('timestamp', 'desc')
  );

  unsubscribe = onSnapshot(q, snapshot => {
    allFeedbacks = [];

    snapshot.forEach(doc => {
      allFeedbacks.push({
        id: doc.id,
        ...doc.data()
      });
    });
```

```
    updateAnalytics();
  });
}

function updateAnalytics() {
  const selectedSubjectId = subjectFilter.value;

  let feedbacksToDisplay = allFeedbacks;

  if (selectedSubjectId) {
    feedbacksToDisplay = allFeedbacks.filter(f => f.subjectId === selectedSubjectId);
  }

  const rows = [];
  const counts = [0, 0, 0, 0, 0];

  let sum = 0;
  let n = 0;

  tbody.innerHTML = '';
  currentComments = [];

  feedbacksToDisplay.forEach(d => {
    const rating = d.rating || 0;
    const comment = d.comment || '';
    const ts = d.timestamp
```

```
    ? d.timestamp.toDate().toISOString().replace('T', ' ').slice(0, 19)
    : '';

  rows.push({ rating, comment, ts });

  if (rating >= 1 && rating <= 5) {
    counts[rating - 1] += 1;
    sum += rating;
    n++;
  }

  if (d.comment && d.comment.trim() !== '') {
    currentComments.push(`Оцінка: ${d.rating}, Коментар: "${d.comment}"`);
  }
});

for (const r of rows) {
  const tr = document.createElement('tr');

  tr.innerHTML = `
    <td class="${r.rating < 3 ? 'text-red' : r.rating > 3 ? 'text-green' : 'text-yellow'}">
      ${r.rating}
    </td>
    <td>${escapeHtml(r.comment)}</td>
    <td>${r.ts}</td>
  `;
```

```
  tbody.appendChild(tr);
}

const avg = n ? (sum / n).toFixed(2) : '—';
avgEl.textContent = `Середня: ${avg}`;

const labels = ['1', '2', '3', '4', '5'];
const data = counts;

if (!chart) {
  const barGradient = chartCtx.createLinearGradient(0, 0, 0, 300);

  chart = new Chart(chartCtx, {
    type: 'bar',
    data: {
      labels,
      datasets: [{
        label: 'Кількість оцінок',
        data,
        backgroundColor: 'rgba(99, 102, 241, 0.85)',
        hoverBackgroundColor: '#4f46e5',
        borderRadius: 6,
        borderSkipped: 'bottom',
        barPercentage: 0.6,
        categoryPercentage: 0.8
      }]
    },
```

```
options: {
  responsive: true,
  maintainAspectRatio: false,
  animation: {
    duration: 800,
    easing: 'easeOutQuart'
  },
  font: {
    family: 'system-ui, -apple-system, sans-serif'
  },
  plugins: {
    legend: {
      display: false
    },
    tooltip: {
      enabled: true,
      backgroundColor: 'rgba(15, 23, 42, 0.9)',
      titleFont: {
        size: 14,
        family: 'system-ui'
      },
      bodyFont: {
        size: 14,
        family: 'system-ui'
      },
      padding: 12,
      cornerRadius: 8,
```

```
    displayColors: false
  }
},
scales: {
  y: {
    display: false,
    beginAtZero: true,
    suggestedMax: 5,
    grace: '20%',
    ticks: {
      stepSize: 1
    },
    grid: {
      display: false,
      drawBorder: false
    },
    border: {
      display: false
    }
  },
  x: {
    grid: {
      display: false,
      drawBorder: false,
      drawTicks: false
    },
    border: {
```

```
            display: false
          },
          ticks: {
            color: '#64748b',
            font: {
              weight: '600'
            }
          }
        }
      }
    });

  } else {
    chart.data.datasets[0].data = data;
    chart.update();
  }
}

function stopListening() {
  if (unsubscribe) {
    unsubscribe();
    unsubscribe = null;
  }
}

function escapeHtml(text) {
```

```
  return text.replace(/[&<>"']/g, (m) => ({
    '&':'&amp;',
    '<':'&lt;',
    '>':'&gt;',
    '"':'&quot;',
    "'":'&#39;'
  }[m]));
}

const aiBtn = document.getElementById('aiBtn');
const aiSummary = document.getElementById('aiSummary');

aiBtn.addEventListener('click', async () => {
  const maxComments = 50;
  const commentsToAnalyze = currentComments.slice(0, maxComments);

  if (commentsToAnalyze.length === 0) {
    aiSummary.textContent = "Немає текстових коментарів для аналізу.";
    return;
  }

  aiBtn.disabled = true;
  aiSummary.innerHTML = "<em>Аналізую відгуки... Це може зайняти 10-15 секунд...</em>";

  const apiKey = "AIzaSyBf9t0gub7C7vOr94h95XIAbuMstpTht6o";
```

```
  const url =
    `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-
flash:generateContent?key=${apiKey}`;


  const prompt = `
Ти помічник викладача. Проаналізуй наступні відгуки студентів про пару.
Зроби дуже коротке резюме (3-4 речення).
Виділи головні плюси та мінуси, якщо вони є.
Відгуки:
` + commentsToAnalyze.join('\n');


  let retries = 3;
  let success = false;


  while (retries > 0 && !success) {
    try {
      const controller = new AbortController();


      const timeoutId = setTimeout(() => controller.abort(), 15000);


      const response = await fetch(url, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          contents: [{
```

```
      parts: [{ text: prompt }]
    }]
  }),
  signal: controller.signal
});

clearTimeout(timeoutId);

if (response.status === 429) {
  throw new Error("TOO_MANY_REQUESTS");
}

const data = await response.json();

if (data.error) {
  if (
    data.error.code === 429 ||
    data.error.status === "RESOURCE_EXHAUSTED"
  ) {
    throw new Error("TOO_MANY_REQUESTS");
  }

  throw new Error(data.error.message);
}

const aiText = data.candidates[0].content.parts[0].text;
```

```
  aiSummary.innerHTML = aiText.replace(/\n/g, '<br>');

  success = true;

} catch (err) {

  if (err.message === "TOO_MANY_REQUESTS") {
    aiSummary.innerHTML = `
      <strong style='color: #dc3545;'>Увага:</strong>
      На даний момент кількість запитів до штучного інтелекту перевищена
      (Too Many Requests). Будь ласка, зачекайте 1-2 хвилини і спробуйте ще раз.
    `;
    break;
  }

  retries--;

  console.warn(
    `Спроба невдала. Залишилось спроб: ${retries}`,
    err
  );

  if (retries === 0) {
    if (err.name === 'AbortError') {
      aiSummary.textContent =
        "Помилка: Сервер AI довго не відповідав (Тайм-аут). Спробуйте пізніше.";
    } else {
```

```
        aiSummary.textContent =
          "Сталася помилка при зверненні до AI: " + err.message;
      }
    } else {
      aiSummary.innerHTML = `
        <em>
          Сервер перевантажений.
          Автоматична повторна спроба...
          (Залишилось: ${retries})
        </em>
      `;

      await new Promise(resolve => setTimeout(resolve, 2000));
    }
  }
}

aiBtn.disabled = false;
});

loadStatus();
```