Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ BACKEND ЧАСТИНИ ЗАСТОСУНКУ ДЛЯ ДИТЯЧОГО САДОЧКА

DEVELOPMENT AND RESEARCH OF THE BACKEND PART OF THE APPLICATION FOR KINDERGARTEN

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Ніколайчук Є. Р.
Керівник:
к.т.н., доцент
Ліщина Н. М.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«___» _____ 202__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Ніколайчуку Євгенію Руслановичу

1. Тема кваліфікаційної роботи: Розробка та дослідження backend частини застосунку для дитячого садочка
Керівник роботи: Ліщина Наталія Миколаївна, доцент, к.т.н.
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 04 грудня 2025 р.
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки: *аналіз проблематики комунікації в закладах дошкільної освіти та вибір методів дослідження, обґрунтування технологій і реалізація серверної частини застосунку, експериментальне дослідження результативності програмного забезпечення*
5. Перелік графічного матеріалу 11 рисунків, 28 лістингів коду

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Ліщина Н. М.*		
Теоретичне дослідження та практична реалізація	*Ліщина Н. М.*		
Експериментальне дослідження системи	*Ліщина Н. М.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Ліщина Н. М.*		

7. Дата видачі завдання *«02 квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об’єкта проектування	19.11.2025	
5	Практична реалізація об’єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Ніколайчук Є. Р.

Керівник кваліфікаційної роботи _____ Ліщина Н. М.

АНОТАЦІЯ

Ніколайчук Є. Р. Розробка та дослідження backend частини застосунку для дитячого садочка. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 Інженерія програмного забезпечення. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків (згідно зі структурою, затвердженою кафедрою). Робота присвячена проєктуванню та реалізації серверної частини застосунку для дошкільного закладу, що забезпечує централізовану, безпечну та передбачувану комунікацію між адміністраторами, вихователями та батьками.

Основна частина містить аналіз предметної області та вимог; обґрунтування вибору стеку TypeScript, Node.js (Express), MongoDB/Mongoose, Firebase Admin, OpenAPI/Swagger; опис модульної реалізації (Користувачі/Профіль, Groups, Children, Schedule, Feed) із валідацією DTO (Zod), рольовою моделлю доступу (RBAC), санітизацією, обмеженням частоти запитів і централізованим обробленням помилок. Наведено приклади контрактів API у Swagger, фрагменти коду та ілюстративні приклади роботи через Postman. Проведено експериментальну перевірку якості на локальному стенді (успішні сценарії та негативні кейси валідації/доступів), сформульовано переваги обраного рішення порівняно з альтернативами, окреслено ризики та шляхи їх мінімізації. Практичний результат – готовий до інтеграції REST-орієнтований backend із персоналізованим фідом для батьків, розкладом по групах, публікаціями у стрічці та push-сповіщеннями (Expo). Запропонована архітектура забезпечує можливість подальшого масштабування без перегляду базових технічних рішень.

Ключові слова: серверна частина, REST API, TypeScript, Express, MongoDB, Mongoose, Firebase, RBAC, Zod, OpenAPI/Swagger, Expo Push, дошкільна освіта.

ABSTRACT

Nikolaychuk Y. R. Development and Research of the Backend Part of the Application for Kindergarten. Manuscript.

Master's Thesis in Software Engineering, degree program 121«Software Engineering». Lutsk National Technical University. Lutsk, 2025.

The thesis comprises an introduction, three chapters, conclusions, references, and appendices (as per the approved structure). It focuses on designing and implementing the server-side of a mobile application for a kindergarten, enabling centralized, secure, and predictable communication among administrators, teachers, and parents.

The main part presents a domain analysis and requirements, rationale for the chosen stack TypeScript, Node.js (Express), MongoDB/Mongoose, Firebase Admin, OpenAPI/Swagger, and a modular implementation (Users/Profile, Groups, Children, Schedule, Feed) with DTO validation (Zod), role-based access control (RBAC), input sanitization, rate limiting, and centralized error handling. The work includes API contracts documented in Swagger, code fragments, and illustrative Postman interactions. Experimental evaluation on a local setup confirms correct behavior for «happy paths» and negative cases (validation/authorization). The thesis compares alternatives, highlights the advantages of the chosen approach, and outlines risks with mitigation strategies.

The practical outcome is a production-ready REST backend featuring a personalized parent feed, group schedules, feed posts, and push notifications (Expo). The proposed architecture is open to further scaling without revisiting the core technical decisions.

Keywords: backend, REST API, TypeScript, Express, MongoDB, Mongoose, Firebase, RBAC, Zod, OpenAPI/Swagger, Expo Push, early childhood education.

ЗМІСТ

ВСТУП

Актуальність роботи. Цифровізація комунікацій у закладах дошкільної освіти стає звичною потребою, а не «додатковою опцією». Щоденні оголошення, розклад занять, меню/харчування, нагадування та обмін повідомленнями між вихователями й батьками мають бути швидкими, безпечними й зручними. Розпорошення інформації в месенджерах або соціальних мережах ускладнює контроль важливих подій, створює дублікати та втомлює персонал рутинними розсилками. У цих умовах серверна частина спеціалізованого застосунку – це опорний компонент, що забезпечує керований доступ до даних, прозорі правила авторизації й стабільні програмні інтерфейси для мобільного клієнта.

Мета роботи – спроєктувати та реалізувати серверну (backend) частину застосунку для дитячого садка, яка надає рольовий доступ до даних і покриває ключові сценарії щоденної взаємодії: розклад по днях, стрічку оголошень/постів із push-сповіщеннями та персоналізований фід для батьків, у яких у закладі навчаються кілька дітей. Застосовані технічні підходи мають забезпечити безпеку оброблення персональних даних, передбачуваність API та зручність інтеграції з клієнтом.

Об'єкт дослідження – процеси інформаційної взаємодії учасників освітнього процесу в дитячому садку (адміністратор, вихователь, батьки) в контексті серверних інформаційних систем.

Предмет дослідження – методи, моделі та програмні засоби побудови REST-орієнтованого backend-рішення для мобільного застосунку з урахуванням автентифікації, авторизації, валідації даних і документування інтерфейсів.

Для досягнення мети визначено такі завдання:

– проаналізувати предметну область і визначити MVP-сценарії (розклад, оголошення/пости з push, персоналізований фід батьків);

– сформувати функціональні та нефункціональні вимоги (безпека, продуктивність, підтримуваність);

– обґрунтувати архітектуру та стек, спроєктувати модель даних і DTO-валідацію;

– реалізувати серверну частину з керованою автентифікацією (Firebase), рольовою моделлю доступу (RBAC) та уніфікованими контрактами API;

– задокументувати інтерфейси у OpenAPI/Swagger з прикладами запитів/відповідей;

– провести експериментальні перевірки у Postman/Swagger для «щасливих» і негативних кейсів, підсумувати переваги, обмеження та ризики.

Практична цінність полягає у створенні модульної серверної архітектури на основі TypeScript та Express із документно-орієнтованою моделлю даних (MongoDB/Mongoose), керованою автентифікацією (Firebase) та відкритими контрактами API (OpenAPI/Swagger). Таке поєднання дає змогу швидко впровадити MVP, зменшити кількість помилок інтеграції завдяки валідації DTO та уніфікованим статус-кодам, а також закласти фундамент для подальшого масштабування – від розширення набору ролей до додавання нових сценаріїв.

Методологічну основу становлять принципи проєктування веб-API, підходи до захисту даних і практики спостережуваності сервісів. У роботі використано: валідацію вхідних даних на основі схем (Zod), рольову модель доступу з перевіркою прав на рівні маршрутів, санітизацію запитів і обмеження частоти звернень, стиснення відповідей для зменшення трафіку, а також централізоване оброблення помилок. Для підтвердження якості наведено приклади взаємодії через Swagger і Postman із «щасливими» та негативними кейсами.

Наукова новизна має прикладний характер і полягає в адаптації сучасних інженерних практик до потреб саме дошкільного закладу – з акцентом на персоналізований фід для батьків та оперативні push-сповіщення. Сформована архітектура охоплює типові ролі та сценарії, але водночас лишається відкритою до еволюції без перегляду базових рішень.

Таким чином, запропоноване рішення спрямоване на підвищення якості комунікації між учасниками освітнього процесу та спрощення щоденних операцій у дитячому садку – із дотриманням вимог безпеки, передбачуваності поведінки API та зручності подальшого розвитку.

Апробація результатів дослідження. Ключові положення та практичні результати роботи оприлюднено у тезах доповіді «Мобільний застосунок як інструмент цифрової трансформації дошкільної освіти» на науково-практичній конференції «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025), що підтвердило прикладну придатність запропонованої серверної архітектури [1].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Актуальність теми полягає у потребі швидкої, безпечної та централізованої комунікації між учасниками освітнього процесу в закладах дошкільної освіти. Щоденна діяльність дитячого садка включає численні короткі, але важливі інформаційні події: оновлення розкладу занять, публікацію меню/харчування, оголошення для батьків, передачу фото-матеріалів, нагадування про збори або необхідні довідки. За відсутності єдиного інструмента ці повідомлення розпорошуються між різними каналами, губляться в месенджерах і соціальних мережах, що призводить до втрати критичної інформації, пропусків важливих подій і перевантаження вихователів рутинними розсилками.

Практична важливість такої централізації підтверджується результатами національного дослідження UNICEF щодо ставлень і практик батьків та опікунів у дошкільній освіті в Україні, де наголошено на потребі регулярної та доступної комунікації між закладами й сім'ями [2]. Міжнародні дані OECD також підкреслюють, що регулярна взаємодія центрів дошкільної освіти з батьками (інформування про участь дітей, поради щодо підтримки розвитку вдома) корелює з кращими освітніми результатами та залученістю сімей [3].

У центрі цієї роботи – серверна (backend) частина застосунку для дитячого садка, яка забезпечує стандартизований керований доступ до даних і сервісів для трьох базових ролей: адміністратора закладу, вихователя та батьків. Застосунок покриває ключові сценарії щоденної взаємодії: ведення та публікацію розкладу по днях, інформування щодо меню/харчування, підтримку стрічки оголошень із можливістю додавання фото, а також пуш-сповіщення (оперативні повідомлення й нагадування). Окремий практичний кейс – сім'ї, у яких у дитсадку навчаються

двоє й більше дітей у різних групах: батькам потрібен об'єднаний фід, що агрегує пости з усіх релевантних груп у єдиній стрічці без перемикання контекстів.

Сучасні підходи до побудови серверних застосунків для мобільних клієнтів базуються на принципах REST-орієнтованого API, чіткої рольової моделі доступу та безпекової гігієни. Під безпековою гігієною розуміється комплекс запобіжних заходів: санітизація вхідних даних, протидія ін'єкціям і XSS, контроль частоти звернень, налаштування CORS для обмеження джерел запитів, а також аудит доступів і помилок [4-6].

Важливою складовою сучасної серверної розробки є документування інтерфейсів за допомогою OpenAPI/Swagger. Наявність єдиної специфікації виконує роль «контракту» між бекендом і клієнтом, спрощує інтеграцію, пришвидшує ручне й автоматизоване тестування та допомагає тримати актуальність API упродовж життєвого циклу продукту. Додатково – Web UI Swagger полегшує тестування запитів безпосередньо в браузері, що особливо корисно в періоди інтенсивної розробки й налагодження клієнтських екранів [7, 8].

Теорія проєктування веб-інтерфейсів спирається на семантику HTTP та принципи REST, що задають очікувану поведінку методів, статус-кодів і заголовків [9]. Формалізація контрактів через OpenAPI спрощує узгодження між командами та забезпечує відтворюваність взаємодії клієнт–сервер упродовж життєвого циклу продукту [7]. У питаннях безпеки орієнтуються на практики OWASP для API: керування автентифікацією й авторизацією, валідацію та санітизацію вхідних даних, захист від зловживань на рівні частоти запитів, контроль походження запитів [4-6]. Для уніфікації токен-базованої автентифікації застосовується JWT як стандартизований формат передачі атрибутів ідентичності [10]. Формальна перевірка структури вхідних даних через схеми (на кшталт Zod) зменшує інтеграційні помилки та підвищує передбачуваність API-контрактів [11].

З огляду на чутливість персональних даних, що опрацьовуються, серверна частина має забезпечувати конфіденційність, цілісність та доступність

інформації. Це означає: захищену аутентифікацію користувачів, відсутність надлишкового зберігання чутливих атрибутів на сервері, контроль авторизації на рівні маршрутів і дій, протоколювання доступів і помилок, а також проактивну профілактику атак. У технічній концепції підкреслено доцільність використання Firebase як провайдера автентифікації: він забезпечує перевірку токенів (JWT) [10], інтеграцію з Google Identity та підтримку OAuth2, що зменшує ризики помилок у власній криптологіці й прискорює онбординг мобільних клієнтів [12].

Окремо слід зазначити вимоги до продуктивності та спостережуваності. У технічних нотатках звертається увага на оптимізацію обміну даними через стиснення відповідей (Compression: gzip/Brotli), що зменшує розмір трафіку, пришвидшує завантаження і покращує досвід користувача, особливо в умовах мобільних мереж з обмеженою пропускною здатністю. Структуроване логування та вимірювання базових метрик (латентність читання стрічки, час публікації поста, швидкість доставки пуш-сповіщень) створюють підґрунтя для подальших експериментів і вдосконалення якості [6, 13, 14].

Індустріальні керівництва та інструменти підтверджують ефект базових технік: стиснення відповідей скорочує мережеві витрати для типових JSON-навантажень; CORS і rate-limit застосовують як «захист за замовчуванням» у веб-API [5], [6]. Документування через Swagger UI пришвидшує ручне тестування та верифікацію прикладів, знижуючи поріг входу для інтеграції клієнтів [8]. Практика токен-орієнтованої автентифікації на стороні сервера поширено реалізується через Firebase Admin SDK із перевіркою підпису й терміну дії токенів [12]. Для зберігання даних документно-орієнтована модель MongoDB у поєднанні з Mongoose підтримує еволюцію схем і тонкі вибірки, що позитивно впливає на латентність читання у сценаріях мобільного клієнта [15, 16]. Інструменти на кшталт Postman залишаються стандартом для «польової» перевірки якості – від «щасливих» шляхів до негативних кейсів із фіксацією кодів відповіді та часу виконання [14].

Валідація даних. На межі сервісу застосовується сувора валідація вхідних даних (DTO-підхід, схеми перевірки, централізоване оброблення помилок). Це

зменшує імовірність логічних помилок і «тихої» корупції даних, блокує неочікувані або небезпечні payload-и ще до звернення до БД та забезпечує клієнту прозорі повідомлення про помилки [11].

Зіставлення предметних потреб дитсадка з теоретичними рекомендаціями та індустріальними практиками показує, що REST-підхід із чіткою семантикою HTTP [9], контрактами OpenAPI [7], токен-базованою автентифікацією [10] та суворою DTO-валідацією [11] найкраще відповідає вимогам керованої й безпечної комунікації. Для нашого MVP це дає передбачувані контракти, контроль доступів на рівні ролей та зниження інтеграційних ризиків без надмірної складності інфраструктури.

Узагальнюючи, предметна область диктує вимоги до:

– централізованого зберігання та надання доступу до розкладів, меню, оголошень і медіа;
– гнучкої підтримки різних ролей і сценаріїв прав доступу;
– оперативних сповіщень (push) для своєчасної реакції батьків;
– агрегації контенту для сімей із кількома дітьми;
– суворої валідації DTO та стандартизованого оброблення помилок;
– безпечної, масштабованої та документованої серверної архітектури.

За підсумком підрозділу конкретизовано контекст і вимоги предметної області; узагальнено релевантні засади для веб-API, безпеки та валідації [4, 7], [9, 10, 11]; окреслено практичні орієнтири щодо технологій і експлуатаційних заходів – Node.js/Express, MongoDB/Mongoose, Firebase Admin, CORS, rate-limit, Swagger UI [5, 6, 8, 12, 13, 15, 16, 17]. Ці висновки формують технічні орієнтири для подальшої реалізації.

Для подальших кроків доречно: конкретизувати функціональні та нефункціональні вимоги до серверної частини; обґрунтувати склад стеку під REST-API з відкритими контрактами та токен-автентифікацією [7, 10]; зафіксувати інваріанти DTO-валідації й політику оброблення помилок із посиланням на стандарти безпеки [4, 11]; підготувати методику

експериментальної перевірки контрактів і часу відповіді з використанням Swagger/Postman [8, 14].

1.2 Огляд і аналіз методів та засобів розробки для вирішення проблеми дослідження

Сукупність інструментів, розглянутих у попередніх технічних матеріалах, формує сучасний «мінімальний, але достатній» стек для побудови бекенд-частини мобільного застосунку. Вона включає Node.js/Express як легку веб-рамку для швидкого створення REST-API [13, 17], MongoDB/Mongoose як документно-орієнтовану БД з гнучкою схемою [15, 16], Firebase Admin/Auth як перевірений механізм аутентифікації [12], а також набір middleware для безпеки і стабільності: express-mongo-sanitize, xss-clean, express-rate-limit, CORS, compression [5, 6, 13]. У сукупності це забезпечує баланс між швидкістю розробки, підтримуваністю й операційною надійністю.

Аутентифікація та автентичність користувача. Вибір провайдера автентифікації визначально впливає на безпеку й масштабованість системи. Застосування Firebase дозволяє спиратися на JWT-токени [10], інтегруватися з Google Identity і налаштовувати верифікацію на сервері без зберігання паролів чи критичних секретів у власній інфраструктурі [12].

Захист API та гігієна обробки даних:

– express-mongo-sanitize видаляє з вхідних запитів до MongoDB небезпечні оператори, захищаючи від ін'єкцій;

– xss-clean очищує вхідні дані від шкідливих HTML-скриптів і запобігає XSS;

– express-rate-limit обмежує кількість запитів із однієї IP-адреси у визначений період, ускладнюючи brute force/DDoS [5];

– CORS дозволяє чітко контролювати походження запитів, що критично при взаємодії клієнта з віддаленим сервером [6];

– compression стискає відповіді (gzip/Brotli), знижуючи трафік і підвищуючи швидкодію [13].

Валідація та обробка помилок. Для формалізації контракту запит/відповідь застосовуються DTO-схеми (наприклад, Zod) [11]. Централізований обробник перетворює помилки валідації на стандартизовані відповіді (HTTP 400) зі структурованими деталями; це прискорює діагностику та підвищує передбачуваність API [9].

Документування API. Підтримка OpenAPI/Swagger є невід'ємною частиною процесу: специфікація забезпечує прозорість контрактів, узгодженість між командами та спрощує тестування завдяки вбудованому інтерфейсу [7], [8]. Це зменшує кількість помилок і полегшує розширення продукту новими сценаріями (наприклад, відмітка відвідуваності чи персональні повідомлення), не порушуючи існуючих взаємодій.

Модульність і підтримуваність. Рекомендовано чітко розділяти відповідальності між шаром маршрутизації, контролерами та сервісною логікою; централізувати обробку помилок і стандартизувати формат відповіді. Така організація проєкту підвищує читабельність і полегшує командну роботу, що вважається невід'ємною ознакою «професійного підходу» до бекенд-розробки.

Підсумок огляду методів і засобів. Сформований технологічний стек Node.js, Typescript, Express, MongoDB/Mongoose, Firebase, Swagger, security middleware створює повноцінну, безпечну та гнучку серверну архітектуру, придатну для інтеграції з клієнтами й послідовного нарощування функцій, що необхідні для комунікації в дитячому садку.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Мета роботи – спроєктувати та реалізувати серверну частину застосунку для дитячого садка, яка забезпечує централізовану та безпечну комунікацію між вихователями й батьками: розклад по днях, меню/харчування, стрічку оголошень із фото та push-сповіщення (включно з нагадуваннями), а також підтримує

об'єднаний фід для батьків, у яких у садочку навчаються кілька дітей. Формулювання мети та завдань відповідає вимогам університетських методичних вказівок до структури першого розділу.

Виходячи з мети дослідження, сформульовано такі завдання:

– проаналізувати предметну область взаємодії в дитячому садку та визначити ключові сценарії MVP: розклад по днях, меню/харчування, стрічка оголошень із фото, push-сповіщення (оперативні та нагадування), об'єднаний фід для батьків із кількома дітьми;

– сформувати вимоги до серверної частини з урахуванням ролей доступу, безпечної обробки персональних даних і роботи з медіафайлами та push-подіями;

– обґрунтувати вибір архітектурних підходів до проєктування REST-орієнтованого API для клієнта з акцентом на простоту інтеграції, відмовостійкість і тестованість; визначити склад технологічного стеку;

– визначити нефункціональні вимоги (безпека, продуктивність, масштабованість, спостережуваність, документованість) і критерії приймання для основних користувацьких сценаріїв;

– розробити серверну частину, що реалізує окреслені сценарії, включно з механізмом push-сповіщень, і забезпечує стандартизовану взаємодію з клієнтом;

– провести експериментальне дослідження результативності: виміряти латентність читання стрічки та публікації поста, оцінити стабільність відправлення push-повідомлень і коректність агрегації фіду для сімей із кількома дітьми;

– підготувати комплект документації API (специфікації, приклади запитів/відповідей) і тестових сценаріїв для відтворюваної перевірки якості.

Висновки до розділу 1

У розділі окреслено контекст та потреби предметної області, визначено рольову модель і базові сценарії взаємодії, узагальнено сучасні підходи до

побудови серверної частини для клієнта, а також сформульовано мету, об'єкт, предмет і завдання дослідження. Акцент зроблено на безпеці, документуванні, продуктивності й підтримуваності рішення та на практичних вимогах – push-сповіщення, робота з медіа, агрегування контенту для батьків із кількома дітьми. Сформований аналітичний фундамент слугуватиме основою для подальшого теоретичного обґрунтування і реалізації в наступному розділі, а також для експериментальної перевірки результативності у завершальній частині роботи.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ BACKEND-ЧАСТИНИ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Мета серверної частини – забезпечити надійний обмін даними між учасниками освітнього процесу дитячого садка, підтримати публікацію оголошень, ведення розкладу по днях, інформування про меню/харчування, відправлення push-сповіщень і формування агрегованої стрічки для батьків із кількома дітьми. З урахуванням цих вимог обрано REST-орієнтований підхід і шарову організацію коду (маршрути → контролери → моделі/сервіси) з поясом проміжних обробників (автентифікація, валідація, безпека, централізовані помилки). Така структура зменшує зв'язаність модулів, спрощує тестування та полегшує масштабування функціоналу.

Технологічний стек. Для транспортного рівня застосовано Node.js з Express і TypeScript для підвищення підтримуваності коду. Рівень даних реалізовано на MongoDB із Mongoose, що відповідає динамічній природі сутностей (розклад, оголошення). Автентифікацію делеговано Firebase Admin (верифікація JWT без зберігання паролів на сервері), а доставлення push-сповіщень − сервісу Expo. Документування інтерфейсів виконується за допомогою OpenAPI/Swagger, що формує «контракт» між клієнтом і сервером та прискорює інтеграцію.

Організація репозиторію. Структура каталогу відображає розподіл відповідальностей: конфігурація, маршрути, контролери, схеми даних, middleware, DTO-валідатори, сервіси інтеграцій, інструменти логування і документації.

Мовою програмування обрано TypeScript, оскільки статична типізація знижує ризик неузгодженостей між DTO та фактичними даними, а також робить контракт «клієнт – сервер» прозорішим під час рефакторингів. Express узято як легковагову рамку з мінімумом примусових абстракцій: на етапі MVP це дає

короткий цикл змін, меншу криву входу та можливість поступово вводити дисципліновану шарову структуру без жорсткого фреймворкового каркасу. Від NestJS свідомо відмовлено на користь простоти – за поточних вимог надбудови поверх Express не додають суттєвої цінності, але ускладнюють каркас.

MongoDB із Mongoose відповідає документно-орієнтованій природі домену: сутності на кшталт груп, дітей, розкладу та постів мають вкладені структури й еволюційні схеми. Це спрощує зміни моделі без складних міграцій і добре поєднується з тонкими вибірками та проєкціями полів, що важливо для мобільного сценарію з обмеженою пропускною здатністю мережі. Реляційна СКБД не дає тут явної переваги, натомість потребувала б додаткового проєктування схем і підтримки міграцій, що збільшує витрати на MVP.

Автентифікацію делеговано Firebase Admin з перевіркою JWT на сервері – це скорочує поверхню ризику, пов'язану зі зберіганням паролів та власною криптологікою, і забезпечує надійні життєві цикли токенів. Авторизацію реалізовано через рольову модель доступу з перевірками на рівні маршрутів – такий підхід відповідає функціям персоналу дитсадка й батьків та спрощує подальше розширення ролей у міру зростання продукту.

Якість і безпека тримаються на «поясі» проміжних обробників: схемна валідація DTO на вході, санітизація запитів, контроль походження (CORS), обмеження частоти звернень, централізоване перетворення помилок у передбачувані відповіді. Це зменшує площу атаки, блокує некоректні або небезпечні payload-и до звернення до БД та робить реакції сервісу однаково читабельними для клієнта й засобів тестування.

Контракти API підтримуються у форматі OpenAPI/Swagger – єдина специфікація виступає «джерелом правди» для клієнта, скорочує комунікаційні накладні витрати та полегшує перевірку як «щасливих» сценаріїв, так і негативних кейсів безпосередньо з браузера або через Postman. Це корисно під час паралельної розробки екранів та бекенда.

Від мікросервісної архітектури відмовлено – вона додала б витрат на оркестрацію, спостережуваність і мережеві накладні, не розв'язуючи поточних

задач. Від зовнішнього BaaS для даних теж утрималися – важливо зберегти контроль над доменною логікою, політиками доступу та еволюцією контрактів.

Зведено, обраний підхід випливає з предметної області – він поєднує простоту, передбачуваність і безпеку, мінімізує технічний борг і дає чіткий шлях масштабування без ускладнення архітектури (ліст. 2.1).

Лістинг 2.1 – Структура проєкту (фрагмент дерева каталогів)

```
src/
  config/             # БД, Firebase, CORS, Swagger
  controllers/        # Доменна логіка
  routes/             # REST-маршрути (auth, users, groups,
children, schedule, feed, notifications)
  models/             # Mongoose-схеми (Group, Child, ScheduleEvent,
User, FeedPost)
  middleware/         # auth, roles, validation, errorHandler
  services/           # інтеграції (firebaseAdmin, expoPushService)
  dto/                # Zod-схеми DTO
  types/              # доменні типи (TS)
  utils/              # утиліти
  index.ts            # ініціалізація застосунку
  swagger.ts          # Swagger UI (/api-docs)
  logger.ts           # morgan/winston
```

Кінець лістингу 2.1

Ініціалізація та політики безпеки. Кожен запит проходить через набір стандартних засобів захисту (HTTP-заголовки, CORS, XSS-фільтрація, санітизація для Mongo, лімітування частоти, стиснення), після чого передається до доменної логіки. Такий конвеєр підвищує стійкість до типових атак і покращує продуктивність у мобільних мережах (ліст. 2.2).

Лістинг 2.2 – Ініціалізація застосунку та політик (фрагмент src/index.ts)

```
import helmet from "helmet";
import compression from "compression";
import morgan from "morgan";
import rateLimit from "express-rate-limit";
import mongoSanitize from "express-mongo-sanitize";
import xss from "xss-clean";
```

```
import { connectDB } from "config/db.js";
import { corsConfig } from "config/cors.js";
// ...
app.use(express.json({ limit: "1mb" }));
app.use(cors(corsConfig));
app.use(helmet());
app.use(xss());
app.use(mongoSanitize());
app.use(compression());
app.use(rateLimit({ windowMs: 60_000, max: 120 }));
app.use(morgan("combined", { stream }));

connectDB();
// ... реєстрація /api/... маршрутів і error handler
```

Кінець лістингу 2.2

Автентифікація та контекст користувача. Перевірку автентичності здійснює Firebase Admin; після верифікації до запиту додається контекст користувача з роллю та Mongo-ідентифікатором. Це уніфікує доступ до даних і спрощує авторизаційні перевірки на рівні маршрутів (ліст. 2.3).

Лістинг 2.3 – Перевірка Firebase-токена (фрагмент src/middleware/auth.ts)

```
export const verifyFirebaseToken = async (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith("Bearer ")) {
    return res.status(401).json({ message: "No auth token" });
  }
  const idToken = authHeader.split(" ")[1];
  // ... перевірка токена через admin та співставлення з UserModel
  req.user = { ...decoded, firebaseUid, mongoId:
mongoUser._id.toString(), role: mongoUser.role };
  next();
};
```

Кінець лістингу 2.3

Контроль доступу за ролями. Після верифікації токена формується контекст користувача з роллю, що дозволяє застосувати політику RBAC на рівні маршрутів. Адміністратору надаються повні права; для інших ролей діють явні дозволи залежно від призначення ендпоїнтів. У разі відсутності ролі або недостатніх прав сервер повертає відповідь 403 (Forbidden) (ліст. 2.4).

Лістинг 2.4 – Middleware контролю ролей (фрагмент src/middleware/roles.ts)

```
import { Request, Response, NextFunction } from "express";

 export const requireRole = (...allowedRoles: string[]) => {
   return (req: Request, res: Response, next: NextFunction) => {
     const userRole = req.user?.role;

     if (!userRole)
       return res.status(403).json({ message: "Forbidden: Role
missing" });

     if (userRole === "admin") return next();

     if (allowedRoles.includes(userRole)) {
       return next();
     }

     return res
       .status(403)
       .json({ message: "Forbidden. You don't have permissions."
});
   };
 };
```

Кінець лістингу 2.4

Валідація DTO та централізовані помилки. На межі API застосовується сувора валідація структури запитів (Zod), а всі збої перетворюються на уніфіковані відповіді. Це підвищує прозорість контрактів і спрощує діагностику (ліст. 2.5, ліст. 2.6).

Лістинг 2.5 – Валідація запиту (фрагмент src/middleware/validation.ts)

```
     export const validate = (schema: ZodObject<ZodRawShape>) =>
(req,res,next) => {
     const parsed = schema.safeParse({ body: req.body, params:
req.params, query: req.query });
          if (!parsed.success) {
          return res.status(400).json({
                    success: false, message: "Validation error",
errors: parsed.error.format()
          });
          }
          next();
     };
```

Кінець лістингу 2.5

Лістинг 2.6 – Централізований обробник помилок (фрагмент src/middleware/errorHandler.ts)

```
export const errorHandler = (err, req, res, next) => {
  console.error(err);
  const statusCode = err.statusCode || 500;
  res.status(statusCode).json({
    success: false,
    message: err.message || "Server Error",
    stack: process.env.NODE_ENV === "production" ? null :
err.stack,
  });
};
```

Кінець лістингу 2.6

Документування інтерфейсів. Інтеграція Swagger надає єдину специфікацію OpenAPI і веб-інтерфейс для перевірки маршрутів. Це знижує вартість змін і забезпечує узгодженість між командами (ліст. 2.7).

Лістинг 2.7 – Підключення Swagger UI (фрагмент src/swagger.ts)

```
const options = {
  definition: {
    openapi: "3.0.0",
    info: { title: "Kindergarten API", version: "1.0.0" },
    // ... security: bearerAuth тощо
  },
  apis: ["./src/routes/*.ts"],
};

const swaggerSpec = swaggerJSDoc(options);

export const swaggerDocs = (app: Express) => {
  app.use("/api-docs", swaggerUi.serve,
swaggerUi.setup(swaggerSpec));
};
```

Кінець лістингу 2.7

Відправлення push-сповіщень. Для доставлення повідомлень використовується офіційне HTTP API Expo. Такий вибір мінімізує інфраструктурні ризики та спрощує підтримку (ліст. 2.8).

Лістинг 2.8 – Сервіс Expo Push (фрагмент src/services/expoPushService.ts)

```
    export const sendExpoPush = async (tokens: string[], message:
Omit<ExpoMessage,"to">) => {
      if (!tokens.length) return;
      const notifications = tokens.map((token) => ({
        to: token, sound: "default", title: message.title, body:
message.body, data: message.data ?? null,
      }));
      await fetch("https://exp.host/--/api/v2/push/send", {
        method: "POST",
        headers: { "Content-Type": "application/json", Accept:
"application/json" },
        body: JSON.stringify(notifications),
      });
    };
```

Кінець лістингу 2.8

Запропонована комбінація архітектурних рішень і засобів – REST, шарова організація коду, TypeScript, Express, MongoDB/Mongoose, Firebase Admin, Expo Push, OpenAPI/Swagger – відповідає вимогам предметної області та сприяє надійній експлуатації. Валідація DTO, стандартні засоби захисту, перевірка ролей (RBAC) на рівні маршрутів та централізована обробка помилок забезпечують якість і передбачуваність API, а документування інтерфейсів – узгодженість із клієнтом і скорочення часу інтеграції.

2.2 Модуль «Користувачі / Профіль» – модель даних, DTO-валідація, ендпоїнти та приклади

Модуль відповідає за роботу з профілем користувача – зберігання ключових атрибутів (роль, контактні дані, Expo-токен) та надання прикладних ендпоїнтів для читання й оновлення власних налаштувань. Контекст автентифікованого запиту передається з middleware, тож усі операції цього модуля спираються на вже підготовлені поля req.user і політику доступу за роллю.

Доступ і ролі. Отримання власного профілю доступне кожному автентифікованому користувачу. Адміністратор має розширені права (напр.,

перегляд/керування іншими користувачами), вихователь і батьки – тільки в межах власних операцій. У разі відсутності токена повертається 401, за нестачею прав – 403.

Профіль користувача в БД це мінімалістична схема Mongoose, де фіксуються службові атрибути, роль доступу та канал нотифікацій. Автентифікацію делеговано Firebase, тому сервер зберігає тільки те, що потрібно для авторизації, персоналізації та зв'язку з Expo (ліст. 2.9).

Лістинг 2.9 – Модель користувача: ключові поля профілю та роль доступу (фрагмент з src/models/User.ts)

```
const UserSchema = new Schema<IUser>(
  {
    firebaseUid: { type: String, required: true, unique: true },
    role: {
      type: String,
      enum: ["parent", "teacher", "admin", "superadmin"],
      default: "parent",
    },

    email: { type: String, required: true },
    name: { type: String },
    picture: { type: String },
    phone: { type: String },
    provider: { type: String },
    expoPushToken: { type: String },
    createdAt: { type: Date, default: Date.now },
    updatedAt: { type: Date, default: Date.now },
  }
);
export const UserModel = model<IUser>("User", UserSchema);
```

Кінець лістингу 2.9

Валідаційні схеми обмежують поверхню змін профілю. Можна оновлювати лише безпечні поля, перевіряються формат URL аватарки та довжина номера телефону. Помилки повертаються у структурованому вигляді ще до виклику контролера (ліст. 2.10).

Лістинг 2.10 – Перевірка вхідних даних: налаштування профілю (фрагмент з src/dto/users.dto.ts)

```
import { z } from "zod";

export const updateUserSettingsSchema = z.object({
  body: z.object({
    name: z.string().min(1, "Name is required").optional(),
    picture: z.url("Invalid picture URL").optional(),
    phone: z
      .string()
      .min(5, "Phone too short")
      .max(20, "Phone too long")
      .optional(),
  }),
});
```

Кінець лістингу 2.10

Маршрутизатор користувача зводить воєдино Bearer-автентифікацію, DTO-валідацію та уніфікацію асинхронних помилок. Перелік усіх користувачів за потреби доповнюється перевіркою ролі адміністратора на рівні реєстрації маршруту (ліст. 2.11).

Лістинг 2.11 – Маршрути користувача: Bearer-захист, валідація та підготовка до RBAC (фрагмент з src/routes/users.routes.ts)

```
router.get("/", verifyFirebaseToken, asyncHandler(listUsers));
router.get("/me", verifyFirebaseToken, asyncHandler(getMe));

router.patch(
  "/me/settings",
  verifyFirebaseToken,
  validate(updateUserSettingsSchema),
  asyncHandler(updateSettings)
);
```

Кінець лістингу 2.11

Контролери працюють у контексті автентифікованого користувача: отримання власного профілю і оновлення налаштувань. Відповіді уніфіковані – 200 для успіху, 401 коли немає контексту, 404 якщо профіль не знайдено; формат помилок узгоджено з централізованим обробником (ліст. 2.12).

Лістинг 2.12 – Контролери профілю: отримання, оновлення та збереження (фрагмент з src/controllers/users.controller.ts)

```
export const getMe = async (req: Request, res: Response) => {
       const id = req.user?.mongoId;
       if (!id) return res.status(401).json({ message: "Unauthorized"
});
       const user = await UserModel.findOne({ _id: id }).select("-__v");
       if (!user) return res.status(404).json({ message: "User not
found" });
       res.json({ success: true, user });
     };

     export const updateSettings = async (req: Request, res: Response)
=> {
            const id = req.user?.mongoId;
       const user = await UserModel.findOneAndUpdate(
         { _id: id },
         { $set: req.body },
         { new: true }
       ).select("-__v");
       return res.json({ success: true, user });
     };
```

Кінець лістингу 2.12

Авторизований користувач може читати та оновлювати власний профіль і зберігати свій Expo-токен; операції над усім списком користувачів призначені для адміністратора (перевірка ролі на рівні маршрутів). Типові відповіді – 401 для запитів без токена, 403 за недостатніх прав, 400 у разі помилок валідації, 404 коли запис не знайдено, 200 для успішних операцій (рис. 2.1).

Рисунок 2.1 – Документація API (Swagger) - Users

2.3 Модуль «Групи» – модель, валідація, ендпоїнти

У модулі груп зберігаються базові атрибути навчальної групи (назва, кімната), а також зв'язки з вихователями та дітьми. Саме тут формується «контекст» для розкладу та стрічки: події розкладу прив'язуються до групи, а фід батьків агрегує контент за групами їхніх дітей. Операції створення/оновлення груп доступні персоналу (адміністратор і вихователь), читання – автентифікованим користувачам.

Модель групи фіксує назву, опціональну кімнату, посилання на вихователів та дітей, а також список ідентифікаторів подій розкладу. Така структура дає змогу швидко будувати запити для розкладу та фіду (ліст. 2.13).

Лістинг 2.13 – Модель групи: основні поля та зв'язки (фрагмент з src/models/Group.ts)

```
import { model, Schema, Types } from "mongoose";
import { IGroup } from "types/Group.js";

const GroupSchema = new Schema<IGroup>(
  {
    name: { type: String, required: true },
    room: { type: String },
    teacherIds: [{ type: Types.ObjectId }],
    childIds: [{ type: Types.ObjectId, ref: "Child" }],
    schedule: [{ type: Schema.Types.ObjectId, ref: "ScheduleEvent"
}],
  },
  { timestamps: true }
);

export const GroupModel = model<IGroup>("Group", GroupSchema);
```

Кінець лістингу 2.13

Валідаційні схеми описують дозволені операції: створення групи, часткове оновлення атрибутів та окрема операція керування складом дітей (масове оновлення childIds). Це обмежує «поверхню» змін і унеможливлює випадкове пошкодження зв'язків (ліст. 2.14).

Лістинг 2.14 – Валідація операцій створення/оновлення та керування складом (фрагмент з src/dto/group.dto.ts)

```
import { z } from "zod";

export const createGroupSchema = z.object({
  body: z.object({
    name: z.string().min(1),
    room: z.string().optional(),
    teacherIds: z.array(z.string()).optional(),
  }),
});

export const updateGroupSchema = z.object({
  params: z.object({ id: z.string().min(1) }),
  body: z.object({
    name: z.string().min(1).optional(),
    room: z.string().optional(),
    teacherIds: z.array(z.string()).optional(),
    childIds: z.array(z.string()).optional(),
  }),
});

export const updateGroupChildrenSchema = z.object({
  params: z.object({ id: z.string().min(1) }),
  body: z.object({ childIds: z.array(z.string()).min(0) }),
});
```

Кінець лістингу 2.14

Маршрутизатор груп поєднує автентифікацію, RBAC та DTO-валідацію. Створення, редагування, видалення та керування складом доступні ролям admin і teacher; читання переліку груп – для авторизованих користувачів (ліст. 2.15).

Лістинг 2.15 – Маршрути груп: захист Bearer, RBAC і валідація (фрагмент з src/routes/group.routes.ts)

```
router.get("/", verifyFirebaseToken, asyncHandler(listGroups));
      router.get("/:id", verifyFirebaseToken, asyncHandler(getGroup));
      router. post(
          "/",
          verifyFirebaseToken,
          requireRole("admin", "teacher"),
          validate(createGroupSchema),
          asyncHandler(createGroup)
      );
```

```
router.patch(
    "/:id",
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    validate(updateGroupSchema),
    asyncHandler(updateGroup)
);
router.delete(
    "/:id",
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    asyncHandler(deleteGroup)
);
router.patch(
    "/:id/children",
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    validate(updateGroupChildrenSchema),
    asyncHandler(updateGroupChildren)
);
```

Кінець лістингу 2.15

Контролери реалізують звичні CRUD-операції й службову операцію керування складом групи. Для ілюстрації наведено приклад читання переліку груп і отримання групи за id – відповіді уніфіковані, і з таким самим підходом налаштовано створення/оновлення/видалення та масове оновлення childIds (ліст. 2.16).

Лістинг 2.16 – Контролер переліку груп: читання з уніфікованою відповіддю (фрагмент з src/controllers/group.controller.ts)

```
//...

export const listGroups = async (req: Request, res: Response) => {
  const groups = await GroupModel.find().lean();
  res.json({ success: true, groups });
};

export const getGroup = async (req: Request, res: Response) => {
 const group = await GroupModel.findById(req.params.id)
   .populate("childIds")
   .lean();
 if (!group)
   return res.status(404).json({ success: false, message: "Not
found" });
```

```
 res.json({ success: true, group });
};

//...
```

Кінець лістингу 2.16

Нижче – стислий опис інших операцій контролера Groups (рис. 2.2).

Створення (createGroup) – приймає назву/кімнату/опціональні зв'язки, перевіряє роль (admin/teacher), створює документ; за наявності childIds синхронізує child.groupId. Відповідь – 200/201, помилки – 400/403.

Отримання (getGroup) – повертає групу за id для авторизованих; якщо не знайдено – 404.

Оновлення атрибутів (updateGroup) – точкові зміни назви/кімнати/вихователів із валідацією; доступ – admin/teacher; відповіді – 200 / 400 / 404.

Видалення (deleteGroup) – для admin/teacher; видаляє групу (можливі додаткові бізнес-перевірки), відповіді – 200 / 404 / 403.

Керування складом дітей (updateGroupChildren) – масово оновлює childIds і двосторонні посилання: видаляє дітей із групи, додає нових, синхронізує child.groupId. Відповіді – 200 / 400 / 404 / 403.

Рисунок 2.2 – Документація API (Swagger) - Groups

Модуль Groups забезпечує опору для сценаріїв розкладу й фіду, підтримуючи прозорі зв'язки та консистентність даних. Поєднання DTO-валідації, RBAC і чітких відповідей API спрощує інтеграцію фронтенд клієнта та знижує ризики помилок при зміні складу груп.

2.4 Модуль «Діти» – модель, валідація, ендпоїнти

Сутність «дитина» пов'язує профілі батьків із навчальною групою й виступає базою для подальших сценаріїв: формування стрічки батьків, побудови розкладу та адресації сповіщень. Модель містить базові персональні поля та посилання на батьків і групу. Операції створення/оновлення виконує персонал (admin, teacher), читання доступне авторизованим користувачам; для батьків застосовується фільтрація за їхніми ідентифікаторами.

Модель дитини фіксує ім'я, прізвище, дату народження, список батьків та прив'язку до групи. Така структура мінімізує надлишковість і прискорює пошук як у напрямку «дитина → група», так і «батько/мати → діти» (ліст. 2.17).

Лістинг 2.17 – Модель дитини: персональні поля, батьки та прив'язка до групи (фрагмент з src/models/Child.ts)

```
import { Schema, model } from "mongoose";
     import { IChild } from "types/Child.js";

     const ChildSchema = new Schema<IChild>(
       {
         firstName: { type: String, required: true },
         lastName:  { type: String, required: true },
         birthDate: { type: Date },
         parentIds: [{ type: Schema.Types.ObjectId, ref: "User",
 required: true }],
         groupId:   { type: Schema.Types.ObjectId, ref: "Group",
 required: false },
       },
       { timestamps: true }
     );

     export const ChildModel = model<IChild>("Child", ChildSchema);
```

Кінець лістингу 2.17

Валідаційні схеми регламентують створення та оновлення: обов'язковість імені/прізвища, коректність формату дати, наявність хоча б одного з батьків. Це запобігає over-posting і забезпечує передбачувані помилки ще до взаємодії з контролером (ліст. 2.18).

Лістинг 2.18 – Валідація створення та оновлення дитини (фрагмент з src/dto/child.dto.ts)

```
import { z } from "zod";

      export const createChildSchema = z.object({
        body: z.object({
          firstName: z.string().min(1, "First name is required"),
          lastName:  z.string().min(1, "Last name is required"),
          birthDate: z.string().optional().refine(
            v => v === undefined || !isNaN(Date.parse(v)),
            "Invalid date format"
          ),
          parentIds: z.array(z.string()).min(1, "At least one parent is
  required"),
          groupId:   z.string().optional(),
        }),
      });

      export const updateChildSchema = z.object({
        params: z.object({ id: z.string().min(1) }),
        body: z.object({
          firstName: z.string().min(1).optional(),
          lastName:  z.string().min(1).optional(),
          birthDate: z.string().optional().refine(
            v => v === undefined || !isNaN(Date.parse(v)),
            "Invalid date format"
          ),
          parentIds: z.array(z.string()).optional(),
          groupId:   z.string().optional(),
        }),
      });
```

Кінець лістингу 2.18

Маршрутизатор дітей поєднує автентифікацію, RBAC і DTO-валідацію. Створення, редагування й видалення доступні для admin і teacher; читання переліку/конкретного запису – авторизованим користувачам, причому для батьків контролер застосовує фільтр за їхнім id (ліст. 2.19).

Лістинг 2.19 – Маршрути дітей: Bearer-захист, RBAC і валідація (фрагмент з src/routes/child.routes.ts)

```
router.get("/",  verifyFirebaseToken, asyncHandler(listChildren));

    router.get("/:id", verifyFirebaseToken, asyncHandler(getChild));

    router.post("/",
      verifyFirebaseToken,
      requireRole("admin", "teacher"),
      validate(createChildSchema),
      asyncHandler(createChild)
    );

    router.patch("/:id",
      verifyFirebaseToken,
      requireRole("admin", "teacher"),
      validate(updateChildSchema),
      asyncHandler(updateChild)
    );

    router.delete("/:id",
      verifyFirebaseToken,
      requireRole("admin", "teacher"),
      asyncHandler(deleteChild)
    );
```

Кінець лістингу 2.19

Контролери модуля реалізують стандартний набір CRUD-операцій; для ілюстрації наведено приклад отримання переліку з уніфікованою відповіддю (ліст. 2.20), тоді як інші дії подано стисло на рисунку 2.3.

Лістинг 2.20 – Контролер переліку дітей: читання з уніфікованою відповіддю (фрагмент з src/controllers/child.controller.ts)

```
    import { Request, Response } from "express";
    import { ChildModel } from "models/Child.js";
    export const listChildren = async (req: Request, res: Response) =>
{
      // для батьків тут може застосовуватись фільтрація за їхнім
userId
      const children = await ChildModel.find().lean();
      res.json({ success: true, children });
    };
    // ...
```

Кінець лістингу 2.20

Нижче – стислий опис інших операцій контролера Children.

Створення (createChild) – перевіряє роль (admin/teacher), створює запис, за потреби прив'язує до groupId. Відповіді – 200 / 201, помилки – 400 / 403.

Отримання (getChild) – повертає запис за id для авторизованих; якщо не знайдено – 404.

Оновлення (updateChild) – точкові зміни валідованих полів; доступ – admin/teacher; відповіді – 200 / 400 / 404.

Видалення (deleteChild) – для admin/teacher; видаляє документ; відповіді – 200 / 404 / 403.

Модифікації виконують admin і teacher; читання доступне авторизованим користувачам, для батьків застосовується фільтрація за їхніми дітьми. Типові відповіді – 401 (без токена), 403 (недостатні права), 400 (помилка валідації), 404 (не знайдено), 200/201 (успіх).

Рисунок 2.3 – Документація API (Swagger) – Children

Модуль Children стандартизує зберігання й доступ до даних про дітей, забезпечує консистентність зв'язків з батьками та групами й формує основу для побудови стрічки та розкладу. Поєднання DTO-валідації та RBAC підвищує надійність операцій і передбачуваність API.

2.5 Модуль «Розклад» – модель, валідація, ендпоїнти

Розклад подано як набір подій, прив'язаних до групи: кожна подія має тип (урок, прогулянка, сон, прийом їжі чи довільний), назву та часові межі. Це дозволяє гнучко моделювати щоденну активність і не фіксуватися на жорсткій «сітці днів». Створення й редагування подій доступне персоналу (admin, teacher), читання – усім авторизованим користувачам у межах дотичних груп.

Модель розкладу зберігає посилання на групу, тип активності, назву, описи та часові межі (ліст. 2.21).

Лістинг 2.21 – Модель події розкладу: прив'язка до групи, тип і часові межі (фрагмент з src/models/ScheduleEvent.ts)

```
    const ScheduleEventSchema = new Schema<IScheduleEvent>(
      {
        groupId: { type: Schema.Types.ObjectId, ref: "Group", required:
true },
        type: { type: String, enum:
["meal","sleep","walk","lesson","custom"], default: "custom" },
        title: { type: String, required: true },
        description: { type: String },
        startTime: { type: String, required: true },
        endTime: { type: String, required: true },
      },
      { timestamps: true }
    );

    export const ScheduleEventModel = mongoose.model<IScheduleEvent>(
      "ScheduleEvent",
      ScheduleEventSchema
    );
```

Кінець лістингу 2.21

Валідація DTO гарантує коректність параметрів маршруту та полів тіла запиту: обов'язковий groupId, валідні часові значення, перелік типів подій і узгодженість startTime ≤ endTime (ліст. 2.22).

Лістинг 2.22 – Валідація створення/оновлення події розкладу (фрагмент з src/dto/schedule.dto.ts)

```
export const createScheduleEventSchema = z.object({
       params: z.object({ groupId: z.string().min(1, "groupId is
 required") }),
       body: z.object({
         title: z.string().min(1, "Title is required"),
         description: z.string().optional(),
         startTime: z.string().refine((v) => !isNaN(Date.parse(v)),
 "Invalid start date"),
         endTime: z.string().refine((v) => !isNaN(Date.parse(v)),
 "Invalid end date"),
         type:
 z.enum(["meal","sleep","walk","lesson","custom"]).optional(),
       }),
     });

     export const updateScheduleEventSchema = z.object({
       params: z.object({
         groupId: z.string().min(1),
         eventId: z.string().min(1),
       }),
       body: z.object({
         title: z.string().min(1).optional(),
         description: z.string().optional(),
         startTime: z.string().optional(),
         endTime: z.string().optional(),
         type:
 z.enum(["meal","sleep","walk","lesson","custom"]).optional(),
       }),
     });
```

Кінець лістингу 2.22

Маршрути розкладу поєднують Bearer-автентифікацію, RBAC і DTO-валідацію. Створення, оновлення та видалення подій доступні admin і teacher; читання – авторизованим користувачам (ліст. 2.23).

Лістинг 2.23 – Маршрути розкладу: захист, ролі й валідація (фрагмент з src/routes/schedule.routes.ts)

```
router.get("/:groupId", verifyFirebaseToken,
asyncHandler(getGroupSchedule));

     router.post(
       "/:groupId/events",
```

```
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    validate(createScheduleEventSchema),
    asyncHandler(createScheduleEvent)
  );

  router.patch(
    "/:groupId/events/:eventId",
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    validate(updateScheduleEventSchema),
    asyncHandler(updateScheduleEvent)
  );

  router.delete(
    "/:groupId/events/:eventId",
    verifyFirebaseToken,
    requireRole("admin", "teacher"),
    asyncHandler(deleteScheduleEvent)
  );
```

Кінець лістингу 2.23

Контролери охоплюють логіку створення, читання, оновлення та видалення з додатковою перевіркою належності події до групи (рис. 2.4). Для ілюстрації наведено приклад створення – саме тут видно зв'язування події з групою та оновлення колекції розкладу в документі групи (ліст. 2.24).

Лістинг 2.24 – Контролер створення події: зв'язування з групою та оновлення розкладу (фрагмент з src/controllers/schedule.controller.ts)

```
// ...

    export const createScheduleEvent = async (req: Request, res:
Response) => {
      const { groupId } = req.params;
      const { title, description, startTime, endTime, type } =
req.body;

      const group = await GroupModel.findById(groupId);
      if (!group) return res.status(404).json({ success: false,
message: "Group not found" });

      const event = await ScheduleEventModel.create({
        groupId,
        title,
        description,
```

```
      startTime,
      endTime,
      type: type ?? "custom",
    });

    group.schedule.push(event._id);
    await group.save();

    return res.status(200).json({ success: true, event });
  };

  // ...
```

Кінець лістингу 2.24

Нижче – стислий опис інших операцій контролера Children:

– getGroupSchedule – повертає події групи у хронологічному порядку; 404, якщо групу не знайдено;

– updateScheduleEvent – точкове оновлення валідованих полів; додатково перевіряє, що подія належить саме цій групі; відповіді – 200 / 400 / 404 / 403;

– deleteScheduleEvent – видаляє подію та синхронно прибирає її з масиву group.schedule; відповіді – 200 / 404 / 403.

Модифікації подій доступні admin і teacher; читання – авторизованим користувачам. Типові відповіді – 401 (без токена), 403 (недостатні права), 400 (помилка валідації), 404 (не знайдено), 200 (успіх).

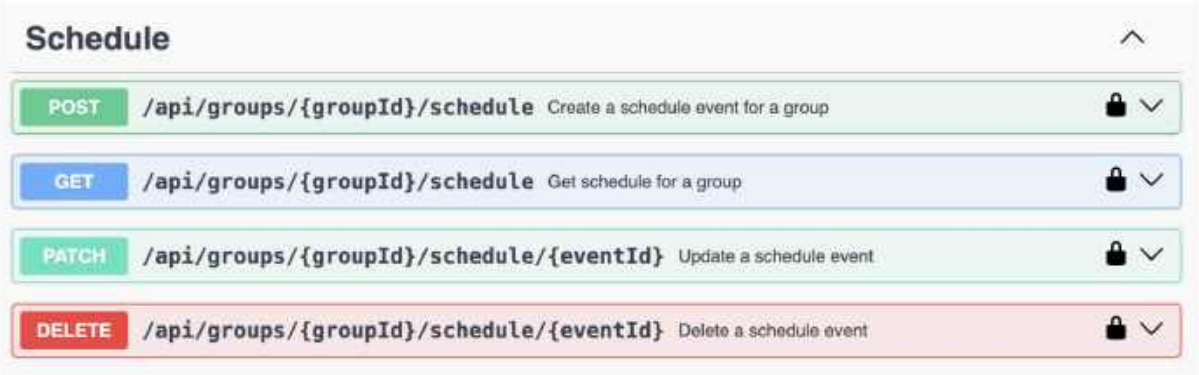

Рисунок 2.4 – Документація API (Swagger) - Schedule

Модуль Schedule надає гнучку модель подій і прозорі операції керування розкладом. Поєднання DTO-валідації, RBAC та перевірок належності подій до

групи забезпечує коректність даних і передбачуваність API, що спрощує інтеграцію клієнта.

2.6 Модуль «Стрічка» – модель, валідація, ендпоїнти

Стрічка агрегує пости вихователів за групами й повертає батькам персоналізований фід: якщо у користувача кілька дітей у різних групах, публікації об'єднуються у зворотному хронологічному порядку. Створення постів виконують ролі admin і teacher, читання фіду доступне кожному авторизованому користувачу; добірка для батьків визначається групами їхніх дітей.

Модель поста охоплює автора (вихователя), групу, заголовок, текст, опційні медіа-посилання та тип повідомлення. Такий склад полів покриває щоденну комунікацію без зайвої складності (ліст. 2.25).

Лістинг 2.25 – Модель поста: автор, група, вміст і тип повідомлення (фрагмент з src/models/Feed.ts)

```
const FeedPostSchema = new Schema<IFeedPost>(
       {
         teacherId: { type: Schema.Types.ObjectId, ref: "User",
 required: true },
         groupId:   { type: Schema.Types.ObjectId, ref: "Group",
 required: true },
         title:     { type: String, required: true },
         text:      { type: String },
         mediaUrls: [{ type: String }],
         type: {
           type: String,
           enum: ["post","announcement","reminder","photo"],
           default: "post",
         },
       },
       { timestamps: true }
     );

     export const FeedPostModel = model<IFeedPost>("FeedPost",
 FeedPostSchema);
```

Кінець лістингу 2.25

Валідаційна схема гарантує, що пост має принаймні текст або одне медіа; перевіряються також groupId, заголовок і формат колекції mediaUrls (ліст. 2.26).

Лістинг 2.26 – Валідація створення поста: обов'язкові поля та інваріант текст або медіа (фрагмент з src/dto/feed.dto.ts)

```
export const createFeedPostSchema = z.object({
       body: z.object({
         groupId: z.string().min(1, "groupId is required"),
         title:   z.string().min(1, "Title is required"),
         text:    z.string().min(1).optional(),
         mediaUrls: z.array(z.string().min(1)).optional(),
         type:
 z.enum(["post","announcement","reminder","photo"]).optional(),
       })
       .refine(
         (data) =>
           (data.text && data.text.trim().length > 0) ||
           (data.mediaUrls && data.mediaUrls.length > 0),
         { message: "Either text or at least one mediaUrl is required",
 path: ["text"] }
       ),
     });
```

Кінець лістингу 2.26

Маршрутизатор стрічки. Маршрути об'єднують Bearer-автентифікацію, RBAC і DTO-валідацію. Читання персонального фіду доступне кожному авторизованому користувачу. Створення, оновлення та видалення постів виконується ролями admin/teacher із перевіркою тіла запиту (ліст. 2.27).

Лістинг 2.27 – Маршрути стрічки: створення, оновлення, видалення поста та читання персонального фіду (фрагмент з src/routes/feed.routes.ts)

```
router.get("/", verifyFirebaseToken, asyncHandler(getMyFeed));

     router.post("/",
       verifyFirebaseToken,
       requireRole("admin","teacher"),
       validate(createFeedPostSchema),
       asyncHandler(createFeedPost)
     );

     router.patch("/:id",
```

```
    verifyFirebaseToken,
    requireRole("admin","teacher"),
    validate(updateFeedPostSchema),
    asyncHandler(updateFeedPost)
  );

  router.delete("/:id",
    verifyFirebaseToken,
    requireRole("admin","teacher"),
    asyncHandler(deleteFeedPost)
  );
```

Кінець лістингу 2.27

Контролери реалізують створення з відправленням push-сповіщень, оновлення, видалення поста та побудову персонального фіду для поточного користувача (ліст. 2.28).

Лістинг 2.28 – Контролери: створення поста з push-сповіщенням та персональний фід (фрагмент з src/controllers/feed.controller.ts)

```
// ...
  export const createFeedPost = async (req: Request, res: Response)
=> {
    const teacherId = req.user!.mongoId;
    const { groupId, title, text, mediaUrls = [], type } = req.body;

    const post = await FeedPostModel.create({
      teacherId, groupId, title, text, mediaUrls, type,
    });

    // сповіщення батьків групи
    const parents = await UserModel.find({ /* відбір батьків групи */
}).lean();
    await sendExpoPush(parents, { title, body: text ?? "New post" });

    return res.status(200).json({ success: true, post });
  };

  export const getMyFeed = async (req: Request, res: Response) => {
    const children = await ChildModel.find({ parentIds:
req.user?.mongoId }, { groupId: 1 }).lean();
    if (!children.length) return res.json({ feed: [] });

    const groupIds = [...new Set(children.map(c =>
c.groupId?.toString()).filter(Boolean))];

    const feed = await FeedPostModel.find({ groupId: { $in: groupIds
```

```
} })
        .sort({ createdAt: -1 })
        .lean();

      res.json({ feed });
    };

    // ...
```

Кінець лістингу 2.28

Нижче – стислий опис інших операцій контролера Feed (рис. 2.5):

– updateFeedPost – часткові зміни полів title, text, mediaUrls, type з перевіркою інваріанта «текст або медіа»; доступ – admin/teacher (за політики авторства – автор або адміністратор). Відповіді – 200 / 400 / 401 / 403 / 404;

– deleteFeedPost – видалення запису за ідентифікатором; доступ – admin/teacher (за політики авторства – автор або адміністратор). Відповіді – 200 / 403 / 404.

Створення/оновлення/видалення – ролі admin і teacher (за політики авторства – автор або адміністратор); читання фіду – авторизовані користувачі. Типові відповіді – 401 (без токена), 403 (недостатні права), 400 (помилка валідації), 404 (не знайдено), 200 (успіх).

Рисунок 2.5 – Документація API (Swagger) - Feed

Модуль Feed забезпечує повний цикл керування контентом – створення, читання, оновлення та видалення постів – із чіткими правилами доступу для ролей admin і teacher та персоналізованою видачею фіду для батьків. Поєднання

DTO-валідації, RBAC і уніфікованих відповідей формує передбачувану поведінку API та знижує ризики помилок. Інтеграція з push-сповіщеннями оперативно доставляє важливі оновлення адресним отримувачам. Симетрія CRUD-операцій спрощує інтеграцію клієнта й підтримку рішення.

Висновки до розділу 2

Реалізовано цілісну серверну архітектуру на TypeScript, Express із шаровою організацією коду (моделі – DTO – маршрути – контролери – middleware), що спрощує підтримку та розширення функціоналу.

Моделювання даних виконано на MongoDB/Mongoose з чіткими зв'язками між сутностями (Users ↔ Children ↔ Groups ↔ Schedule ↔ Feed). Структури мінімалістичні, але достатні для сценаріїв MVP; забезпечено консистентність двосторонніх прив'язок (зокрема, у Groups ↔ Children).

Валідацію організовано через Zod DTO з інваріантами на рівні предметної логіки (наприклад, для Feed: «текст або принаймні одне медіа»; для Schedule: коректність часових полів). Це зменшує поверхню помилок і забезпечує передбачувані відповіді ще до доступу до БД.

Безпека та доступ реалізовані послідовно: Bearer-перевірка токена (middleware), RBAC на рівні маршрутів (requireRole(…)), санітизація та обмеження запитів – у поєднанні з централізованим обробленням помилок. Операції модифікації даних обмежені ролями admin/teacher; читання персонального контенту для батьків відфільтровано за їхніми дітьми.

Контракти API документовано через OpenAPI/Swagger; у розділі наведено «щасливі» та негативні кейси для ключових ендпоїнтів. Це виконує роль «живого контракту», пришвидшує інтеграцію клієнта та полегшує тестування.

Функціональні модулі оформлено уніфіковано:

– Users / Профіль – читання та часткове оновлення власних даних, збереження Expo-токена;

– Groups – CRUD із керуванням складом дітей та підтримкою консистентності;

– Children – CRUD для сутності «дитина» з прив'язкою до батьків і групи;

– Schedule – CRUD для подій розкладу з валідацією часових меж і належності до групи;

– Feed – CRUD для постів (створення, читання, оновлення, видалення) з персональним фідом для батьків та інтеграцією push-сповіщень.

Окремо виділено службові компоненти для автентифікації й push-нотифікацій (підрозділи розділу 2 посилаються на відповідні middleware/сервіси).

Продуктивність і спостережуваність підтримуються за рахунок lean-запитів до MongoDB, мінімальних проєкцій полів, стиснення відповідей та уніфікованого логування запитів/помилок.

Код-лістинги у розділі подано вибірково – лише ключові фрагменти моделей, DTO, маршрутів і контролерів; деталізація поведінки підтверджена скріншотами зі Swagger із фокусом на коди статусів і повідомлення про помилки.

Бекенд частина застосунку відповідає вимогам предметної області та методичним критеріям якості: забезпечено безпечні контракти, передбачувану валідацію, прозорий розподіл доступів і готовність до подальшого масштабування. Уніфікація підходів (DTO-валідація, RBAC, централізовані помилки, Swagger) робить API стабільним для експлуатації та зручним для інтеграції з клієнтом.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЯКОСТІ РІШЕННЯ

3.1 Методика проведення дослідження

Застосунок спрямований на оперативну, безпечну й керовану комунікацію між учасниками освітнього процесу дитячого садка. Цільові ролі – адміністратор, вихователь, батьки. У фокусі MVP – розклад по днях, оголошення/пости в групах, нагадування, персоналізований фід для батьків із кількома дітьми, push-сповіщення.

Для оцінювання бекенду визначено критерії якості:

– безпека – автентичність і авторизація, валідація DTO, санітизація, CORS, rate-limit, уніфіковані помилки;

– продуктивність – час відповіді ключових ендпоїнтів, стиснення відповідей, ефективні запити до БД;

– масштабованість – безстанова логіка, горизонтальне масштабування, ізоляція модулів;

– підтримуваність – шарова структура, читабельність коду, Swagger-контракти, консистентні коди відповідей;

– спостережуваність – логування запитів/помилок, вимірювання P50/P95 часу відповіді;

– зручність інтеграції – стабільні URL, чіткі схеми/DTO, приклади у Swagger, прогнозовані статус-коди.

Здійснимо порівняльний аналіз альтернативних підходів.

Архітектура:

– Node.js, Express (обрано) – низький поріг входу, багата екосистема, зручний REST для мобільних клієнтів, легка інтеграція з Firebase та MongoDB;

– .NET/Java моноліт – сильні інструменти типізації/профілювання, утім вищі стартові витрати для MVP і повільніший цикл ітерацій;

– Serverless-підхід – еластичність під навантаження, але підвищена складність локальної розробки й відлагодження, а також «розмитість» контрактів між функціями.

Зберігання даних:

– MongoDB/Mongoose (обрано) – документно-орієнтована модель добре відображає сутності «група/дитина/пост», природні зв'язки, гнучке еволюціонування схем у процесі розробки;

– реляційна СУБД – сильні транзакційні гарантії й сувора нормалізація, утім вища «ціна» змін схеми на етапі активного MVP, складніша еволюція моделі.

Автентифікація:

– керована автентифікація (Firebase, обрано) – надійна перевірка токенів, менше ризиків у криптології, швидкий онбординг клієнта;

– власна реалізація – повний контроль і кастомні флоу, але значні витрати на безпеку, тестування та підтримку.

Документування та інтерфейси:

– OpenAPI/Swagger (обрано) – «живий контракт», автогенерація прикладів, ручне тестування;

– розрізнені Postman-колекції без єдиної специфікації – швидкий старт, але слабша узгодженість і гірша відтворюваність.

Обраний стек TypeScript, Express, MongoDB/Mongoose, Firebase, Swagger забезпечує оптимальний баланс між швидкістю розробки, безпекою та підтримуваністю для умов MVP дитсадка. Альтернативи вигідні у великих проєктах із жорсткими нефункціональними вимогами, але на етапі MVP підвищують складність і ТСО (Сукупна вартість володіння).

3.2 Обробка та аналіз отриманих результатів

Опишемо нефункціональні характеристики та переваги рішення.

Безпека. Валідація DTO (Zod) на вході, санітизація запитів до БД, rate-limit, CORS, RBAC на рівні маршрутів, централізований обробник помилок. Це зменшує поверхню атак і гарантує контрольовані відмови.

Продуктивність. Стиснення відповідей (gzip/Brotli), «тонкі» вибірки (lean, проєкції полів), уникнення зайвих популяцій, прості індекси. Це скорочує латентність і навантаження на мережу.

Масштабованість. Безстанова логіка API, токени замість сесій, відсутність тісних зв'язків між модулями, можливість горизонтального масштабування бекенду і реплікації БД.

Підтримуваність. Шарова організація (моделі – DTO – маршрути – контролери – middleware), уніфіковані статус-коди й повідомлення про помилки, єдина специфікація OpenAPI. Це спрощує код-рев'ю, тестування й навчання нових учасників.

Спостережуваність. Логування HTTP-запитів і винятків, фіксація P50/P95 для ключових ендпоїнтів, можливість інтеграції з APM у подальших ітераціях.

Зручність інтеграції. Однозначні маршрути, стабільні схеми, приклади у Swagger, негативні кейси з поясненням помилок – мобільна команда швидше підключає екрани.

Здійснимо експериментальну перевірку якості.

Стенд і передумови. Бекенд запущено локально (Node.js, MongoDB) із фіксованими версіями залежностей. Перевірка виконувалась через Postman: для кожного ключового ендпоїнта зроблено один контрольний запит, зафіксовано код відповіді та час виконання (Response Time). Такий «польовий» підхід достатній, аби підтвердити коректність контрактів і базову швидкодію в локальному середовищі.

Ключові сценарії та результати:

– персональний фід батьків – GET /api/feed/my – 200 OK ≈ 145 мс. Відповідь містить пости з груп, до яких належать діти поточного користувача (персоналізований фід) (рис. 3.1);

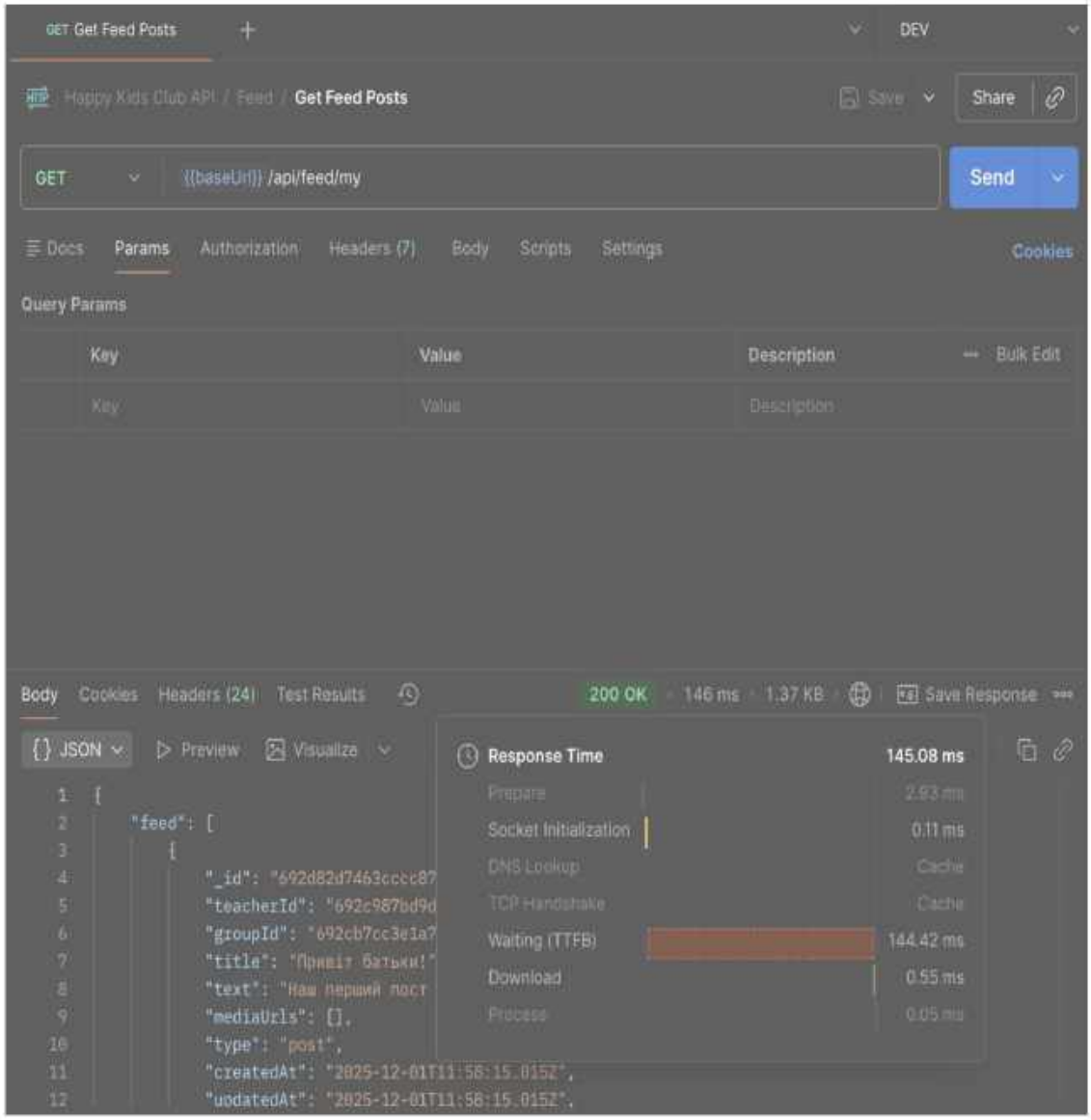

Рисунок 3.1 – Результат виконання GET api/feed/my ендпойнту

– створення поста у стрічці – POST /api/feed – 200 ≈ 217 мс. Тіло відповідає специфікації, інваріант «текст або медіа» дотримано (рис. 3.2);

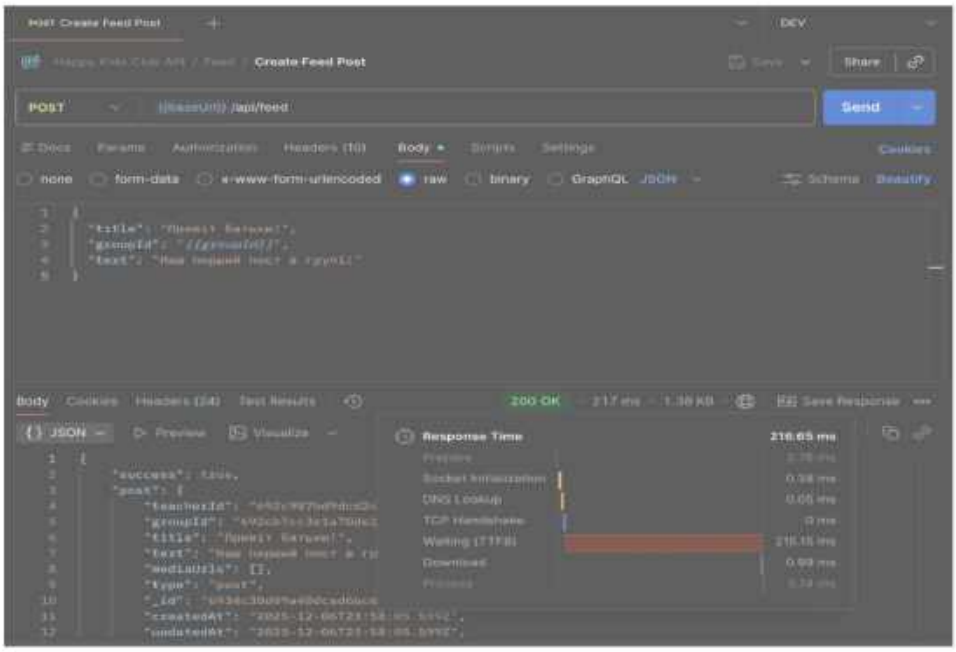

Рисунок 3.2 – Результат виконання POST api/feed ендпойнту

– розклад групи – GET /api/groups/:groupId/schedule – 200 OK ≈ 179 мс. Повертається повний список подій із коректним упорядкуванням (рис. 3.3).

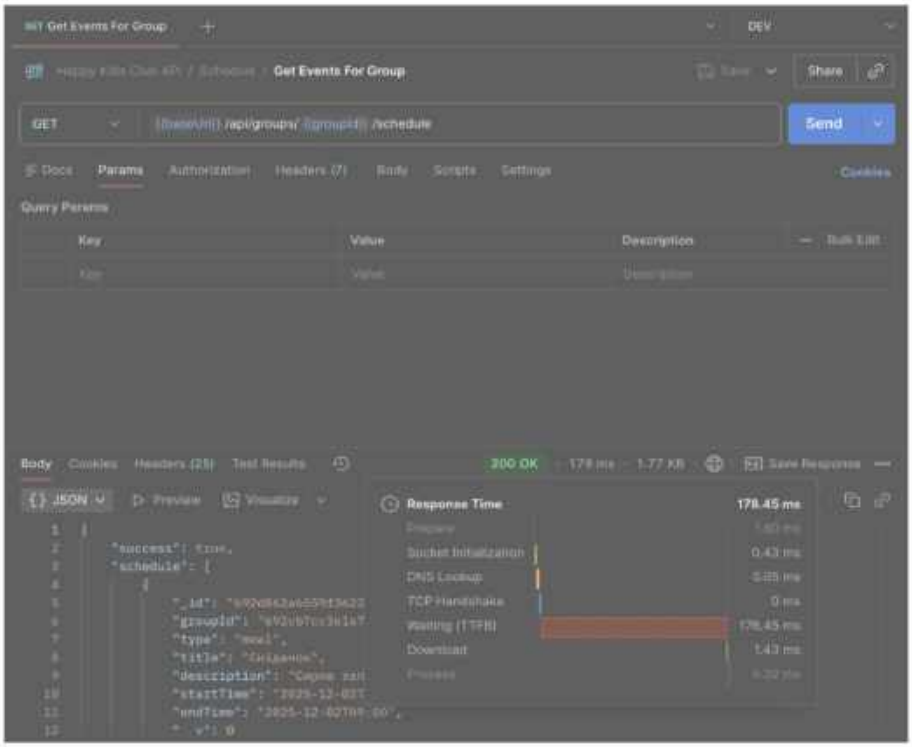

Рисунок 3.3 – Результат виконання GET /api/groups/:groupId/schedule ендпойнту

Негативні кейси – приклади відмов:

– без токена – 401 Unauthorized (рис. 3.4);

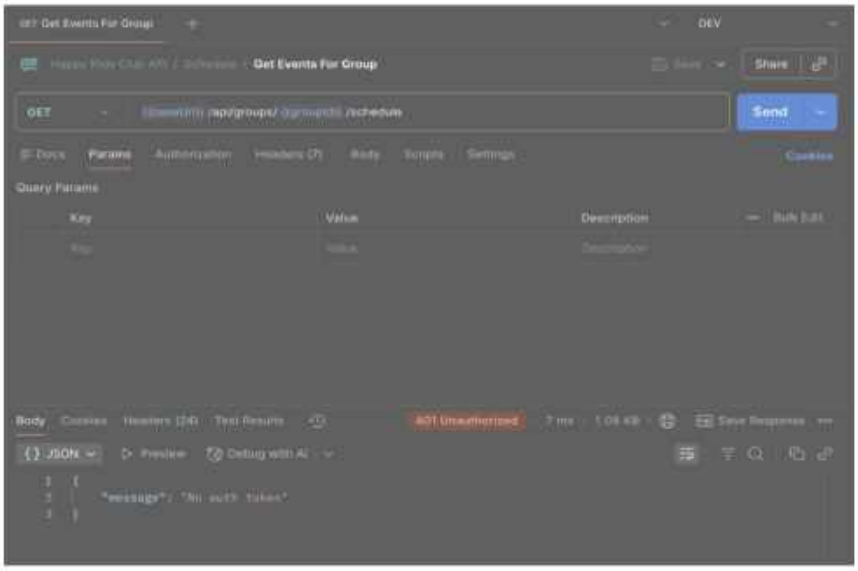

Рисунок 3.4 – Результат виконання GET /api/groups/:groupId/schedule ендпойнту без токена авторизації

– недостатні права – 403 Forbidden (рис. 3.5);

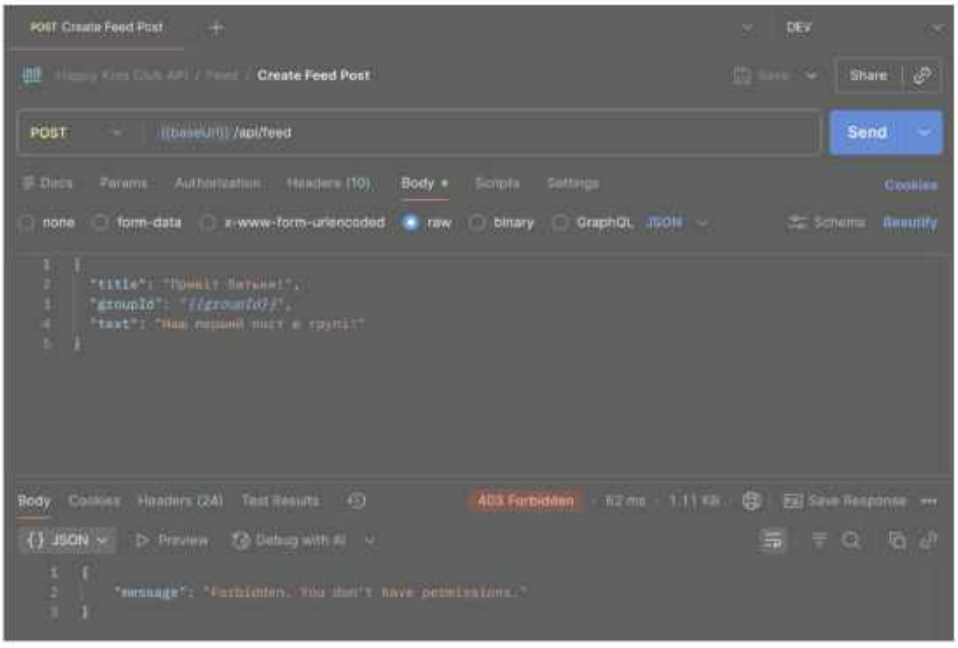

Рисунок 3.5 – Результат виконання POST /api/feed ендпойнту з parent токеном

– помилка валідації – 400 Bad Request із деталізацією (рис. 3.6).

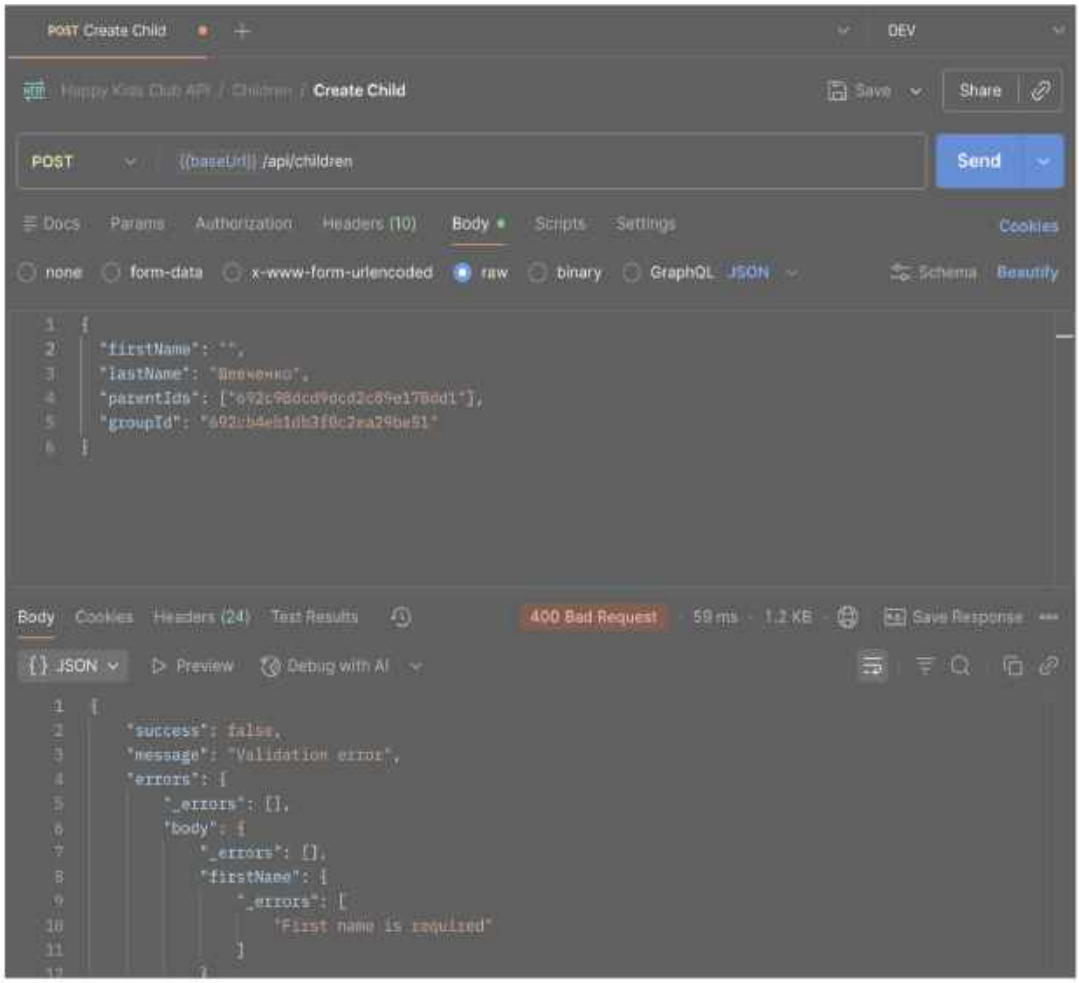

Рисунок 3.6 – Результат виконання POST /api/children ендпойнту з помилкою валідації поля firstName

Інтерпретація. Одиничні вимірювання в локальному оточенні підтверджують: контракти API коректні, а час відповіді ключових ендпоїнтів – у межах очікуваного для дев-середовища. Відмови 4xx відтворюються лише у навмисно змодельованих сценаріях (валідація, RBAC), що свідчить про передбачувану поведінку системи. Для продуктивної інфраструктури передбачено аналогічну методику з більшим числом вимірювань.

Висновки до розділу 3

Обраний стек TypeScript, Express, MongoDB/Mongoose, Firebase, OpenAPI/Swagger забезпечує збалансовану комбінацію швидкої розробки, безпеки та підтримуваності для MVP дитячого садка. Порівняння з альтернативами підтвердило: для поточного обсягу функціоналу документно-орієнтована модель і керована автентифікація дають кращу гнучкість і нижчу сукупну вартість володіння, ніж реляційні схеми з «важкими» фреймворками або повний BaaS.

Нефункціональні властивості підкріплено конкретними технічними заходами: Zod-валідація DTO, RBAC, санітизація, rate-limit і CORS підсилюють безпеку; lean-вибірки, мінімальні проєкції та стиснення відповідей покращують продуктивність; шарова організація коду, уніфіковані статус-коди та єдина специфікація OpenAPI спрощують супровід і інтеграцію. Логування запитів і помилок формує базу для подальшої спостережуваності та оптимізації.

Експериментальні перевірки у Postman/Swagger засвідчили коректність контрактів і передбачувану поведінку API: «щасливі» сценарії стабільно повертають 200/201, а «керовані» відмови (400/401/403/404) відтворюються у відповідних негативних кейсах. Заміряний у локальному середовищі час відповіді відповідає очікуванням для дев-стенду та підтверджує готовність рішення до інтеграції з клієнтом.

Ключові ризики – залежність від провайдера автентифікації, еволюція схем БД, можливе зростання навантаження, надійність push-каналу – мають чіткі контрзаходи: інкапсуляцію інтеграцій, контрольовані міграції та версіонування API/DTO, пагінацію й індекси, повторні спроби та чистку недійсних токенів. Архітектура лишається відкритою до подальшого масштабування – від додавання підролей і нових сценаріїв до впровадження кешування та APM у продакшн-оточенні.

Запропонована серверна частина відповідає вимогам предметної області та визначеним критеріям якості – надійні контракти, прозора авторизація й

валідація, стабільні результати перевірок – і має зрозумілий шлях еволюції без перегляду базових архітектурних рішень.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи є проєктування та реалізація серверної частини застосунку для дитячого садка, призначеної для надійної та керованої комунікації між вихователями й батьками. Сформульована мета досягнута: забезпечено рольовий доступ до даних, підтримано розклад по днях, стрічку оголошень/постів із push-сповіщеннями та персоналізований фід для батьків, у яких у закладі навчаються кілька дітей.

У ході дослідження проаналізовано предметну область і сучасні підходи до побудови веб-API, обґрунтовано вибір архітектурних рішень і стеку засобів. Обрана конфігурація – REST на семантиці HTTP, шарова організація коду, TypeScript, Express, MongoDB/Mongoose, автентифікація через Firebase Admin, документування контрактів у форматі OpenAPI/Swagger – відповідає вимогам безпеки, передбачуваності та підтримуваності для сценаріїв дошкільної освіти.

Практичну реалізацію виконано з акцентом на якість і безпеку: застосовано схемну DTO-валідацію, санітизацію вхідних даних, контроль походження запитів і обмеження частоти звернень, централізоване оброблення помилок, а також політику доступу на рівні ролей. API задокументовано єдиною специфікацією, що спростило взаємодію з клієнтською частиною та прискорило інтеграційні перевірки.

Експериментальна перевірка за допомогою Swagger і Postman показала коректну роботу «щасливих» сценаріїв і передбачувану обробку негативних кейсів; коди відповідей відповідають контрактам, час виконання контрольних запитів – очікуванням для локального стенду. Сукупний результат підтверджує відповідність реалізації сформульованим функціональним і нефункціональним вимогам.

Практична цінність роботи полягає в готовності серверної частини до інтеграції з клієнтом і в можливості еволюції без перегляду базових рішень. Запропонована архітектура забезпечує чіткий шлях масштабування – від

індексації та пагінації до кешування, асинхронної обробки push-подій і розширення набору ролей – зберігаючи сумісність контрактів і керованість змін.

Новизна має прикладний характер і полягає в адаптації сучасних інженерних практик під потреби саме дитсадка: персоналізований фід для батьків із кількома дітьми, керовані push-сповіщення, сувора валідація даних і рольовий контроль доступу на рівні маршрутів. Апробацію результатів здійснено в межах наукового обговорення та публікації тез, що підтверджує практичну придатність підходу.

Виконання сформульованих завдань підтверджено результатами роботи:

– проаналізовано предметну область і виділено MVP-сценарії (розклад, оголошення з push, персоналізований фід);

– зафіксовано функціональні та нефункціональні вимоги з акцентом на безпеку, продуктивність і підтримуваність;

– обґрунтовано архітектуру та стек, спроєктовано модель даних і схеми DTO-валідації;

– реалізовано серверну частину з Firebase-автентифікацією, RBAC та уніфікованими контрактами API;

– підготовлено й підтримано специфікацію OpenAPI/Swagger з прикладами запитів/відповідей;

– проведено інтеграційні перевірки у Swagger/Postman для «щасливих» і негативних кейсів з очікуваними кодами відповідей і часом виконання.

Підсумовуючи, поставлена мета досягнута, завдання виконано в повному обсязі, а розроблене серверне рішення відповідає потребам предметної області та створює надійну основу для подальшого впровадження й масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ніколайчук Є. Р., Ніколайчук С. Р., Ліщина Н. М. Мобільний застосунок як інструмент цифрової трансформації дошкільної освіти. Тези доповідей X Міжнародної науково-практичної конференції «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025), 23-24 травня 2025 р. С. 220-223.

2. Parents' and Caregivers' Attitudes and Engagement Practices in Early Childhood Education in Ukraine. UNICEF, 2024. URL: https://knowledge.unicef.org/ukraine/resource/parents-and-caregivers-attitudes-and-engagement-practices-early-childhood-education (дата звернення: 01.10.2025).

3. Engaging parents and guardians in early childhood education and care centres. OECD Policy Brief, 2024. URL: https://www.oecd.org/en/publications/engaging-parents-and-guardians-in-early-childhood-education-and-care-centres_d05dd1cf-en.html (дата звернення: 01.10.2025).

4. OWASP API Security Top 10. URL: https://owasp.org/www-project-api-security/ (дата звернення: 01.11.2025).

5. Express-rate-limit – документація (GitHub). URL: https://github.com/nfriedly/express-rate-limit (дата звернення: 1.12.2025).

6. MDN Web Docs – CORS (Cross-Origin Resource Sharing). URL: https://developer.mozilla.org/docs/Web/HTTP/CORS (дата звернення: 1.12.2025).

7. OpenAPI Specification 3.0/3.1 – офіційний сайт. URL: https://spec.openapis.org/ (дата звернення: 21.10.2025).

8. Swagger UI – інструменти документування API. URL: https://swagger.io/tools/swagger-ui/ (дата звернення: 23.10.2025).

9. HTTP Semantics, RFC 9110 – IETF. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 05.11.2025).

10. JSON Web Token (JWT), RFC 7519 – IETF. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 01.11.2025).

11. Zod – валідація схем для TypeScript/JS. URL: https://zod.dev/ (дата звернення: 26.10.2025).

12. Firebase Admin SDK – автентифікація та серверні інструменти. URL: https://firebase.google.com/docs/admin (дата звернення: 05.10.2025).

13. Node.js – офіційна документація. URL: https://nodejs.org/ (дата звернення: 01.10.2025).

14. Postman Learning Center – довідник і приклади. URL: https://learning.postman.com/ (дата звернення: 15.11.2025).

15. MongoDB Manual – довідник і керівництва. URL: https://www.mongodb.com/docs/ (дата звернення: 10.10.2025).

16. Mongoose – офіційна документація ODM. URL: https://mongoosejs.com/docs (дата звернення: 15.10.2025).

17. Express.js – API Reference 4.x. URL: https://expressjs.com/ (дата звернення: 17.10.2025).

18. Expo – Push Notifications (огляд і керівництво). URL: https://docs.expo.dev/push-notifications/overview/ (дата звернення: 17.11.2025).