УДК 004.65

ПРОЕКТУВАННЯ ДОКУМЕНТНО-ОРІЄНТОВАНОЇ БАЗИ ДАНИХ FIRESTORE ДЛЯ СИСТЕМ УПРАВЛІННЯ ЗАМОВЛЕННЯМИ

Ткаченко Андрій

Науковий керівник: канд.техн. наук, доцент Баранюк О.Ф.

Центральноукраїнський державний університет імені Володимира Винниченка, м. Кропивницький, Україна

Розглянуто особливості проектування документно-орієнтованої бази даних Firebase Firestore для системи управління замовленнями сервісного центру з технічного обслуговування пристроїв. Проаналізовано архітектурні рішення побудови веб-застосунків на основі патерну односторінкового додатку з використанням бібліотеки React для презентаційного рівня та хмарної платформи Firebase для серверної інфраструктури.. Розроблено модульну архітектуру програмних компонентів з чітким розділенням відповідальності між модулями автентифікації користувачів та роботи з базою даних через інкапсуляцію функціональності в незалежні модулі. Представлено механізми контролю доступу через декларативні правила безпеки Firestore для розмежування прав користувачів відповідно до ролей та забезпечення захисту персональних даних клієнтів. Описано компонентну архітектуру презентаційного рівня з використанням механізму контексту React для управління глобальним станом автентифікації користувача без необхідності явної передачі властивостей через проміжні компоненти.. Визначено перспективи розширення функціональності системи через впровадження сповіщень реального часу, аналітичних модулів та інтеграції платіжних систем для підвищення якості обслуговування клієнтів сервісного центру.

Ключові слова: документно-орієнтована база даних, Firebase Firestore, система управління замовленнями, веб-застосунок, React, хмарна платформа, модульна архітектура, автентифікація користувачів, правила безпеки.

Design of Firestore document-oriented database for order management systems

A. Tkachenko

Scientific supervisor: Candidate of Technical Sciences, associate professor Baranyuk O.F.

Volodymyr Vynnychenko Central Ukrainian State University, Kropyvnytskyi, Ukraine

The features of designing a Firebase Firestore document-oriented database for an order management system of a service center for device maintenance are considered. Architectural solutions for building web applications based on the single-page application pattern using the React library for the presentation layer and the Firebase cloud platform for server infrastructure are

analyzed. A modular architecture of software components with a clear separation of responsibilities between user authentication modules and database operations through encapsulation of functionality in independent modules has been developed. Mechanisms of access control through declarative Firestore security rules for user rights differentiation according to roles and ensuring protection of clients' personal data are presented.. System scalability mechanisms through automatic collection sharding and load distribution in the Firebase cloud infrastructure are considered. Prospects for expanding system functionality through implementation of real-time notifications, analytical modules and payment system integration to improve service center customer service quality are identified.

Keywords: document-oriented database, Firebase Firestore, order management system, web application, React, cloud platform, modular architecture, user authentication, security rules.

Постановка проблеми. Розвиток цифрових технологій спричинив масштабну трансформацію бізнес-процесів сервісних підприємств, що обслуговують технічні пристрої. Сервісні центри стикаються з необхідністю обробки зростаючих обсягів замовлень на ремонт та технічне обслуговування, що вимагає впровадження автоматизованих систем управління заявками клієнтів. Традиційні підходи до організації збереження даних через реляційні бази виявляють обмеження при роботі з динамічними структурами інформації, характерними для сучасних веб-застосунків. Документно-орієнтовані бази даних пропонують альтернативний підхід до персистентності інформації, що базується на збереженні самодостатніх документів зі складною внутрішньою структурою замість розподілу даних між множиною таблиць через механізми нормалізації.

Хмарна платформа Firebase надає комплексну екосистему сервісів для побудови веб-застосунків, включаючи документно-орієнтовану базу даних Firestore з підтримкою реального часу синхронізації та складних запитів до структурованих даних. Архітектура односторінкових застосунків з бібліотекою React для презентаційного рівня та Firestore для рівня персистентності дозволяє створювати високореактивні інтерфейси без необхідності підтримки власної серверної інфраструктури. Проектування схеми бази даних для системи управління замовленнями сервісного центру вимагає врахування специфіки документної моделі даних, де відсутність жорсткої схеми надає гнучкість

структури документів, проте потребує ретельного планування для забезпечення ефективності запитів та підтримки цілісності інформації.

Розробка систем обробки замовлень для сервісних центрів супроводжується необхідністю балансування між нормалізацією даних для уникнення дублювання та денормалізацією для оптимізації продуктивності читання. Firestore застосовує горизонтальне масштабування через автоматичне шардування колекцій, що дозволяє обробляти великі обсяги документів без деградації швидкодії. Механізми контролю доступу через декларативні правила безпеки забезпечують розмежування прав користувачів безпосередньо на рівні бази без залежності від коректності реалізації авторизації в клієнтському коді.

Трирівнева архітектура системи з чітким розділенням презентаційного шару, бізнес-логіки та рівня персистентності створює основу для підтримуваного та масштабованого програмного забезпечення. Модульна організація коду з інкапсуляцією функціональності автентифікації та роботи з базою даних дозволяє незалежно розвивати компоненти системи. Асинхронна природа операцій з Firestore через конструкції async/await забезпечує неблокуючу взаємодію з базою даних та коректну обробку помилок на кожному етапі виконання запитів.

Впровадження систем управління замовленнями в сервісних центрах потребує врахування вимог до швидкості відгуку інтерфейсу, безпеки даних клієнтів та можливості розширення функціоналу відповідно до зростаючих потреб бізнесу. Вибір технологічного стеку та архітектурних рішень визначає довгострокову успішність проєкту через вплив на витрати підтримки, можливості масштабування та гнучкість адаптації до змін бізнес-вимог.

Аналіз досліджень і публікацій. Дослідження О. І. Желтобрюхова присвячене розробці інформаційної системи прийому замовлень у сервіс-центрі з обслуговування мобільних пристроїв, де проаналізовано архітектурні підходи до організації взаємодії клієнтської та серверної частин додатку [1]. А. Брітвін, Дж. Алравашде та Р. Ткачук розглянули принципи побудови клієнт-серверних систем для парсингу даних з веб-сторінок, що демонструє сучасні підходи до

організації розподілених застосунків [2]. М. Фаулер у своїй праці систематизував методології моделювання об'єктно-орієнтованих систем через уніфіковану мову моделювання, що застосовується при проектуванні архітектури баз даних та програмних компонентів [3].

Дж. Кейсон дослідив особливості безсерверної архітектури для веб-застосунків, де хмарні сервіси забезпечують автоматичне масштабування без необхідності адміністрування інфраструктури [4]. Дж. Катцер та Дж. Кейл проаналізували сучасні фреймворки для розробки клієнтських систем з акцентом на реактивні бібліотеки інтерфейсів [5]. Б. Леклерк та Дж. Кейл розглянули застосування технологій великих даних для оптимізації роботи сервісних центрів через аналітику поведінки клієнтів та прогнозування навантаження [6]. А. Пейн та П. Фроу систематизували методології стратегічного управління клієнтськими відносинами через інтеграцію систем управління взаємовідносинами з клієнтами [8]. Н. Х. Тієн, Н. Т. Х. Дунг, Т. Т. Т. Транг та П. Б. Нгок провели оцінку задоволеності клієнтів застосунками сервісних центрів на основі показників якості програмного забезпечення [10].

Метою статті виступає проектування документно-орієнтованої бази даних Firestore для системи управління замовленнями сервісного центру з технічного обслуговування пристроїв.

Виклад основного матеріалу (результатів) дослідження. Функціональна архітектура розробленої системи обробки замовлень сервісного центру грунтується на модульному підході, що забезпечує чітке розділення відповідальності між компонентами та ефективну взаємодію різних рівнів додатку. Основою функціональної моделі виступає трирівнева архітектура, де презентаційний шар реалізований засобами бібліотеки React, бізнес-логіка інкапсульована в спеціалізованих модулях для роботи з автентифікацією та даними, а рівень персистентності представлений хмарною базою даних Firebase Firestore [9]. Кожен рівень виконує строго визначені функції та взаємодіє з іншими через чітко специфіковані інтерфейси, що забезпечує слабку зв'язаність компонентів та високу підтримуваність системи загалом.

Архітектурне рішення базується на принципах розділення відповідальності та інкапсуляції, де кожен модуль має єдину, добре визначену мету та надає зовнішньому світу тільки необхідний мінімум функціональності через публічні програмні інтерфейси. Такий підхід дозволяє незалежно розробляти, тестувати та модифікувати окремі компоненти без впливу на інші частини системи, що значно спрощує процес супроводу та розширення функціоналу в майбутньому [1].

Модуль автентифікації користувачів реалізує повний цикл управління обліковими записами, включаючи реєстрацію нових користувачів з валідацією вхідних даних, автентифікацію через електронну пошту та пароль, відновлення доступу за допомогою механізму скидання пароля, а також розмежування прав доступу на основі ролей. Функція реєстрації користувача приймає чотири параметри та виконує комплексну послідовність дій, що включає створення облікового запису в системі автентифікації Firebase Authentication, оновлення профілю з додаванням відображуваного імені, та формування детального запису у базі даних Firestore з повною інформацією про користувача [2]. Асинхронна природа операцій забезпечується через використання конструкції async/await, що дозволяє виконувати послідовні операції без блокування основного потоку виконання та забезпечує коректну обробку помилок на кожному етапі процесу реєстрації.

Обробка помилок реалізована через конструкцію try-catch, де будь-яка помилка на будь-якому етапі процесу реєстрації перехоплюється та повертається у структурованому вигляді з прапорцем успішності операції та текстом повідомлення про помилку. Процес автентифікації реалізований через функцію loginUser, що забезпечує безпечну перевірку облікових даних користувача та встановлення сесії через механізм токенів доступу [4]. Функція використовує вбудовані можливості Firebase Authentication для валідації пароля шляхом порівняння хешованих значень на серверній стороні, що гарантує, що фактичні паролі ніколи не передаються в незашифрованому вигляді через мережу та не зберігаються в незахищеному вигляді в базі даних. Після успішної

автентифікації система генерує JWT-токен, який зберігається в локальному сховищі браузера та автоматично додається до всіх наступних запитів до серверних сервісів для ідентифікації користувача без необхідності повторного введення облікових даних.

Механізм токенів забезпечує баланс між безпекою та зручністю використання, дозволяючи користувачу залишатися автентифікованим протягом тривалого періоду часу без компромісу щодо захисту даних [5].

Використання змінних оточення для зберігання списку адміністраторів забезпечує гнучкість конфігурації системи, дозволяючи змінювати склад адміністраторів без необхідності модифікації програмного коду та повторного розгортання додатку. Модуль роботи з базою даних Firestore інкапсулює всі операції створення, читання та оновлення документів, забезпечуючи єдину точку доступу до персистентного сховища та абстрагуючи клієнтський код від деталей взаємодії з базою даних. Функція створення профілю користувача демонструє типовий патерн роботи з документами Firestore, де кожен користувач представлений окремим документом у колекції users з унікальним ідентифікатором, що відповідає ідентифікатору з системи автентифікації Firebase Authentication [7].

Використання однакового ідентифікатора в обох системах спрощує зв'язування даних та забезпечує консистентність ідентифікації користувача в різних частинах системи.

Оператор розпилення застосовується для об'єднання вхідних даних користувача з системними полями, такими як мітка часу створення, що генерується серверною функцією serverTimestamp для забезпечення точності та консистентності часових даних незалежно від налаштувань локального часу на клієнтських пристроях. Центральним елементом бізнес-логіки системи виступає функціонал управління заявками на ремонт, реалізований через набір спеціалізованих функцій для створення, читання та оновлення заявок [8]. Функція створення нової заявки генерує унікальний ідентифікатор документа автоматично через виклик методу doc з колекцією без вказівки конкретного

ідентифікатора, що делегує відповідальність за генерацію унікальних ідентифікаторів системі Firestore та гарантує відсутність конфліктів при одночасному створенні множинних заявок різними користувачами.

Початковий статус заявки автоматично встановлюється як pending для позначення, що заявка створена та очікує розгляду адміністратором сервісного центру. Структура даних заявки включає обов'язкові поля ідентифікатора користувача для зв'язування заявки з автором, інформації про пристрій з вказівкою типу та моделі, текстового опису проблеми з пристроєм, рівня терміновості ремонту, поточного статусу заявки в життєвому циклі обробки, а також міток часу створення та останнього оновлення для відстеження історії змін [10]. Отримання заявок користувача реалізовано через функцію getUserRepairRequests, що формує складний запит до бази даних з використанням програмного інтерфейсу запитів Firestore Query API для фільтрації документів за ідентифікатором користувача та сортування результатів у зворотному хронологічному порядку. Виконання фільтрації та сортування на серверній стороні бази даних мінімізує обсяг даних, що передаються через мережу, та зменшує навантаження на клієнтський додаток, перекладаючи обчислювально інтенсивні операції на потужні серверні ресурси хмарної платформи.

Конструкція запиту включає метод where для специфікації умови фільтрації, де відбираються тільки документи з полем userId, що дорівнює ідентифікатору поточного користувача, та метод orderBy для визначення порядку сортування результатів за полем createdAt у напрямку спадання, що гарантує відображення найновіших заявок на початку списку для зручності користувача. Результати запиту обробляються через ітерацію по знімку запиту з формуванням масиву об'єктів, де кожен об'єкт містить ідентифікатор документа та всі його поля даних, об'єднані через оператор розпилення для створення плоскої структури без вкладеності [3]. Адміністративний функціонал системи включає можливість перегляду всіх заявок у системі незалежно від автора через функцію getAllRepairRequests, що виконує запит до колекції заявок без

фільтрації за користувачем, але зберігає сортування за датою створення для забезпечення хронологічного відображення заявок.

Механізм оновлення статусу заявки updateRepairRequestStatus використовує метод updateDoc для часткового оновлення документа, змінюючи тільки поле статусу та мітку часу останнього оновлення без необхідності повного перезапису всіх полей документа, що оптимізує використання мережевої пропускної здатності та прискорює операції запису до бази даних [6]. Функція додавання відгуку адміністратора addFeedbackToRequest демонструє більш складний сценарій оновлення документа, де одночасно змінюється статус заявки на reviewed для позначення, що заявка переглянута адміністратором, додається вкладений об'єкт з відгуком, що містить текст повідомлення від адміністратора, електронну адресу адміністратора для ідентифікації автора відгуку, та серверну мітку часу створення відгуку, а також оновлюється загальна мітка часу останньої модифікації документа заявки.

Конфігураційний модуль системи інкапсулює процес ініціалізації комплекту засобів розробки програмного забезпечення Firebase SDK та експортує готові до використання екземпляри сервісів автентифікації та бази даних для використання в інших модулях додатку [9]. Використання змінних оточення для зберігання конфігураційних параметрів Firebase забезпечує безпечне управління чутливими даними, такими як ключі програмного інтерфейсу застосунків та ідентифікатори проєктів, запобігаючи їхньому випадковому розголошенню через системи контролю версій та дозволяючи використовувати різні конфігурації для різних середовищ розробки, тестування та продакшену без необхідності модифікації програмного коду додатку.

Структура конфігураційного об'єкта включає шість обов'язкових полів, кожне з яких зчитується з відповідної змінної оточення через об'єкт process.env, що надається середовищем виконання Node.js та доступний під час побудови додатку через інструментарій Create React App.

Взаємодія між презентаційним рівнем та рівнем бізнес-логіки здійснюється через механізм контексту React, що дозволяє передавати

глобальний стан автентифікації користувача та пов'язані функції управління сесією через дерево компонентів без необхідності явної передачі властивостей через кожен проміжний компонент [4]. Контекст автентифікації AuthContext інкапсулює інформацію про поточного користувача, його статус автентифікації, роль в системі та надає методи для виконання операцій входу, виходу та реєстрації, які можуть бути викликані з будь-якого компонента додатку без необхідності імпортування модулів роботи з автентифікацією безпосередньо в компонентах інтерфейсу.

Таблиця 1

Структура колекції repairRequests у базі даних Firestore

Поле	Тип даних	Опис	Обов'язкове
id	String	Унікальний ідентифікатор заявки	Так (авто)
userId	String	Ідентифікатор користувача-автора заявки	Так
userEmail	String	Електронна адреса користувача для швидкого доступу	Так
userName	String	Відображуване ім'я користувача	Так
deviceType	String	Тип пристрою (iPhone, iPad, MacBook, Apple Watch)	Так
deviceModel	String	Конкретна модель пристрою	Так
problemDescription	String	Детальний опис проблеми	Так
urgency	String	Рівень терміновості (low, normal, high, urgent)	Так
status	String	Поточний статус (pending, reviewed, in_progress, completed, cancelled)	Так
createdAt	Timestamp	Дата та час створення заявки	Так
updatedAt	Timestamp	Дата та час останнього оновлення	Так
feedback.message	String	Текст відгуку адміністратора	Ні
feedback.adminEmail	String	Електронна адреса адміністратора, що залишив відгук	Ні
feedback.timestamp	Timestamp	Час створення відгуку	Ні

Функціональна модель системи демонструє добре продуману архітектуру з чітким розділенням відповідальності між модулями, що забезпечує високу підтримуваність коду та можливість незалежного розвитку окремих компонентів, створюючи надійну основу для майбутнього розширення функціоналу відповідно до зростаючих потреб бізнесу сервісного центру та очікувань користувачів системи обробки замовлень.

Висновки та перспективи подальших пошуків у напрямі дослідження. Проведене дослідження дозволило розробити функціональну архітектуру системи обробки замовлень сервісного центру на основі документно-орієнтованої бази даних Firebase Firestore з використанням трирівневої архітектури та модульного підходу до організації програмних компонентів. Спроектована структура колекцій users та repairRequests забезпечує ефективне зберігання та швидкий доступ до інформації про користувачів та заявки на ремонт через оптимізовані запити з фільтрацією та сортуванням на серверній стороні бази даних. Реалізовані механізми автентифікації користувачів через Firebase Authentication та розмежування прав доступу через декларативні правила безпеки Firestore гарантують захист персональних даних клієнтів та контрольований доступ до функціоналу системи відповідно до ролей користувачів.

Впроваджена стратегія денормалізації даних шляхом дублювання інформації користувача безпосередньо в документах заявок дозволила мінімізувати кількість запитів до бази даних при відображенні списків заявок та підвищити загальну продуктивність системи. Модульна організація коду з інкапсуляцією функціональності в окремі модулі для роботи з автентифікацією та базою даних забезпечує високу підтримуваність системи та можливість незалежного тестування компонентів без залежності від зовнішніх сервісів. Використання асинхронних операцій через конструкції async/await та хмарної інфраструктури Firebase забезпечує масштабованість системи під зростаючим навантаженням без необхідності адміністрування серверного обладнання.

Перспективи подальших досліджень включають розширення функціоналу системи через впровадження механізмів реального часу сповіщень про зміни статусу заявок з використанням Firebase Cloud Messaging для надсилання push-повідомлень на мобільні пристрої клієнтів. Доцільним напрямом розвитку виступає інтеграція аналітичних модулів для збору статистики про типи несправностей пристроїв, терміни виконання ремонтів та рівень задоволеності клієнтів обслуговуванням сервісного центру.

Подальший розвиток системи може включати впровадження модуля управління запасами запчастин з відстеженням наявності компонентів для ремонту та автоматичним формуванням замовлень постачальникам при досягненні мінімального рівня запасів на складі. Інтеграція платіжних систем для онлайн-оплати послуг ремонту та можливість попереднього розрахунку вартості ремонту на основі опису проблеми з використанням алгоритмів машинного навчання становить перспективний напрям розширення функціональності системи для підвищення зручності обслуговування клієнтів та оптимізації фінансових операцій сервісного центру.

Список використаної літератури:

1. Желтобрюхов О. І. Інформаційна система прийому замовлень у сервіс-центрі по обслуговуванню мобільних пристроїв. Миколаїв, 2022. 77 с.
2. Britvin A., Alrawashdeh J., Tkachuk R. Client-Server System for Parsing Data from Web Pages. Advances in Cyber-Physical Systems. 2022. Vol. 7, No. 1. P. 8–13.
3. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd Edition. Print2print, 2016. 208 p.
4. Jason K. Learning Serverless: Design, Develop, and Deploy with Confidence. 1st Edition. O'Reilly Media, 2020. 222 p.
5. Katzer J., Cale J. Developing Client-Side Systems with Modern Frameworks. O'Reilly Media, 2022. 186 p.
6. Lecrerc B., Cale J. Big Data for Service Centers. Tailor&Francis, 2022. 148 p.
7. Savova-Videnova E, Uzunov M. Mongo DB and why should we choose it. Science, Engineering & Education, 4, (1), 2019, P. 19-27.
8. Payne A., Frow P. Strategic Customer Management: Integrating Relationship Marketing and CRM. Cambridge University Press, 2015. 520 p.
9. Stevenson D. What is Firebase? The complete story, abridged. Medium, 2018. URL: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0
10. Tien N. H., Dung N. T. H., Trang T. T. T., Ngoc P. B. Assessing Customer Satisfaction for Service Center Applications. Journal of Software Engineering. 2021. Vol. 18, No. 14. P. 249–268.

Відомості про автора:

Ткаченко Андрій Вікторович – студент ІІ курсу магістратури факультету інформаційних технологій, математики та природничих наук Центральноукраїнського

державного університету імені Володимира Винниченка, тел. +380675957081, tkachenkoandrusha@gmail.com.