https://doi.org/10.30857/2786-5371.2023.2.5

УДК 004.94; 681.3

ПАНАСЮК О. І., ПЛЕСКАЧ В. Л., МОЛЧАНОВ Б. С.
Київський національний університет імені Тараса Шевченка, Україна

РОЗШИРЕННЯ ФУНКЦІОНАЛУ ОНЛАЙН КОМУНІКАЦІЇ МЕДИЧНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМІ HELSI

Мета. *На основі порівняльного аналізу двох популярних СУБД MongoDB та Redis визначити кращу для розширення функціоналу існуючої. Створити новий сервіс для забезпечення комунікації в реальному часі через вебсокет з'єднання для медичної інформаційної системи HELSI. При створенні бази даних врахувати можливість високого навантаження та необхідність додати агрегування даних в наступних ітераціях функціоналу.*

Методика. *При реалізації функціоналу використовувалась мова C# 10.0, фреймворк ASP.NET Core 6.0, і база даних MongoDB.*

Результати*. Проведено аналітичне дослідження і порівняння СУБД MongoDB та Redis за характеристиками продуктивності, безпечності, типом системи зберігання даних і видом внутрішньої мови. Запропоновано використання комбінації бібліотеки SignalR Core для платформи .NET 6 разом з СУБД MongoDB для забезпечення управління підключеннями по протоколу Websocket у високонавантаженій медичній інформаційній системі при проведені онлайн чату між пацієнтом і лікарем. Розроблено новий сервіс для забезпечення комунікації в реальному часі через вебсокет з'єднання для медичної інформаційної системи HELSI.*

Наукова новизна*. Розроблено теоретичні та методологічні підходи до порівняльної оцінки характеристик сучасних NoSQL СУБД і застосуванні гібридних підходів поєднання технологій, які використовують при розроблення прикладного програмного забезпечення при побудові програмних систем.*

Практична значимість. *За допомогою мови програмування C# 10.0 та платформи .NET 6 і бази даних MongoDB розширено функціонал існуючої прикладної медичної інформаційної системи HELSI можливістю чату між лікарем і пацієнтом під час онлайн прийомів. Спроектовано базу даних для зберігання інформації, пов'язаної з даним функціоналом. Підготовлено супутню документацію для розробки інтерфейсу щодо створеного функціоналу.*

Ключові слова: *чат; онлайн прийом; клієнт-серверна архітектура; медична інформаційна система; агрегування даних.*

Вступ. Однією з найважливіших напрямів діяльності держави є охорона здоров'я є. В сучасному житті в сфера охорони здоров'я активно впроваджуються інформаційні технологій.

Інформаційні медичні системи забезпечують високу ефективність обробки і зберігання медичної інформації та ведення документообігу [1–3].

З 2018 року в Україні розпочато реформу системи охорони здоров'я в Україні, створено «Національну службу здоров'я України» (НСЗУ). Вона забезпечує функціонування eHealth – системи охорони здоров'я [3, 4]. Для функціонування державного центрального компоненту eHealth створено медичні інформаційні системи. Серед яких система HELSI є однією з найбільших в Україні [4, 5]. Через систему вже укладено 24 мільйони декларацій між лікарями та пацієнтами. Щодня через Helsi на прийоми до лікарів записуються більше 230 тисяч пацієнтів.

В системі Helsi використовуються наступні програмні рішення [5]:

Reform.helsi.me – система для реєстрації та роботи медичних закладів з мінімально необхідним функціоналом. Не дозволяє працювати з helsi.me.

Helsi.me – система для реєстрації пацієнтів та запису на прийоми. Також дозволяє отримувати ліки, перевіряти рецепти, отримувати страхування та переглядати інформацію щодо сервісу.

Apt.helsi.me – система для роботи аптечних закладів та їх працівників. Дозволяє провізорам перевіряти стан рецептів, погашати їх для роботи з програмою «Доступні ліки».

Helsi.pro – система для роботи медичних закладів та їх працівників з повним функціоналом. Дозволяє проводити прийоми, діагностику, процедури, створювати медичні висновки, перевіряти інформацію щодо минулих прийомів пацієнта, укладати декларації, заповнювати інформацію про заклад для відображення на helsi.me, заповнювати графіки прийому лікарів тощо.

Мобільний додаток Helsi для пацієнтів – дозволяє робити все те ж, що й на веб-платформі, але додає функціонал планування прийому ліків, онлайн-прийомів та в майбутньому – швидких онлайн-консультацій.

Аналіз попередніх досліджень. Ринок програмного забезпечення має велику кількість різнобічних по своїм функціональним можливостям комерційних систем управління БД загального призначення, а також засобами їх оточення практично для всіх моделей машин і для різних операційних систем. Сучасні СУБД володіють засобами забезпечення цілісності даних і надійної безпеки, що дає можливість розробникам гарантувати велику безпеку даних при менших затратах сил на низькорівневе програмування.

Redis – це платформа з відкритим вихідним кодом, ліцензована Berkeley Software Distribution (BSD) [6–8]. Він служить рішенням для зберігання структури даних пам’яті та часто використовується як база даних, кеш-пам’ять і брокер повідомлень. Redis може підтримувати структури даних, включаючи такі елементи, як рядки, хеші, відсортовані набори, запити діапазонів, растрові зображення, гіперлоги, геопросторові індекси, списки, набори та потоки. Redis також підтримує готову реплікацію, сценарії Lua, транзакції та різні рівні збереження диска. Він також може забезпечити високу доступність за допомогою Redis Sentinel, а також автоматичне розділення за допомогою Redis Cluster. Користувачі можуть виконувати атомарні операції над типами Redis, такі як додавання до рядка або збільшення значення в хеші, а також обчислення перетину набору та отримання члена з найвищим загальним рангом у відсортованому наборі. Щоб досягти найвищої продуктивності, Redis працює безпосередньо з набором даних пам’яті. Залежно від конкретного випадку використання користувача, його можна зберегти, періодично створюючи дамп набору даних на головний диск, або додаючи кожну окрему команду до нового журналу. У деяких випадках користувачам просто потрібна багатофункціональна мережева система кешу пам’яті. Коли виникає така ситуація, функцію збереження можна вимкнути. Інші функції Redis включають можливість підтримувати спрощену асинхронну реплікацію, транзакції, pub і sub, сценарії Lua, використання ключів з обмеженим TTL (час життя), LRU (найменше живий), видалення різних ключів, автоматичне відновлення після відмови та багато іншого. Користувачі можуть поєднати Redis з більшістю мов програмування, які використовуються сьогодні.

MongoDB – це документо-орієнтована база даних, яка має масштабованість і гнучкість, які користувачі вважатимуть привабливими для запитів та індексування [6–9]. MongoDB має можливість зберігати дані в змінних документах, що означає, що поля можуть відрізнятися від документа до документа, а структуру даних можна змінювати з часом. MongoDB здатна надавати рішення для надзвичайно складних проблем і може керувати великими обсягами інформації. MongoDB зберігає, обробляє та виконує дані у двійковій формі та використовує дуже зручне представлення документів у форматі JSON для введення та виведення даних. Функціонал командного рядка MongoDB також досить простий у використанні. Типи даних цієї платформи досить стандартні та схожі на мови програмування зі строгими типами, наприклад Java або C++. Користувачі можуть переглядати ці типи даних у вигляді рядків у відповідних файлах JSON. MongoDB працює з двійковим представленням, у якому кожен конкретний тип BSON представлено цілим числом та рядковими ідентифікаторами для

кожного типу окремого значення. MongoDB – це, по суті, розподілена база даних. Через це надзвичайно висока доступність, корисне горизонтальне масштабування та географічний розподіл є частиною основної функції платформи, а сама платформа проста у використанні. Модель документа MongoDB проста у вивченні та впровадженні користувачами, але при цьому забезпечує всі різноманітні можливості, необхідні для створення найскладніших рішень складних проблем будь-якого масштабу.

Постановка завдання. Основною задачею роботи є на основі порівняльного аналізу двох популярних СУБД MongoDB та Redis визначити кращу для розширення функціоналу існуючої. Створити новий сервіс для забезпечення комунікації в реальному часі через вебсокет з'єднання для медичної інформаційної системи HELSI. При створенні бази даних врахувати можливість високого навантаження та необхідність додати агрегування даних в наступних ітераціях функціоналу.

Результати дослідження. Порівняльна характеристика баз даних Redis і MongoDB [6–9]

Зберігання даних. Mongodb зберігає дані у формі документів JSON. З іншого боку, redis зберігає дані в пам'яті як пару ключ-значення. Redis працює дуже добре, коли дані швидко змінюються та можуть поміститися в пам'ять. Нижче показана (рис. 1) ілюстрація зберігання даних у Redis з можливими типами даних [7].

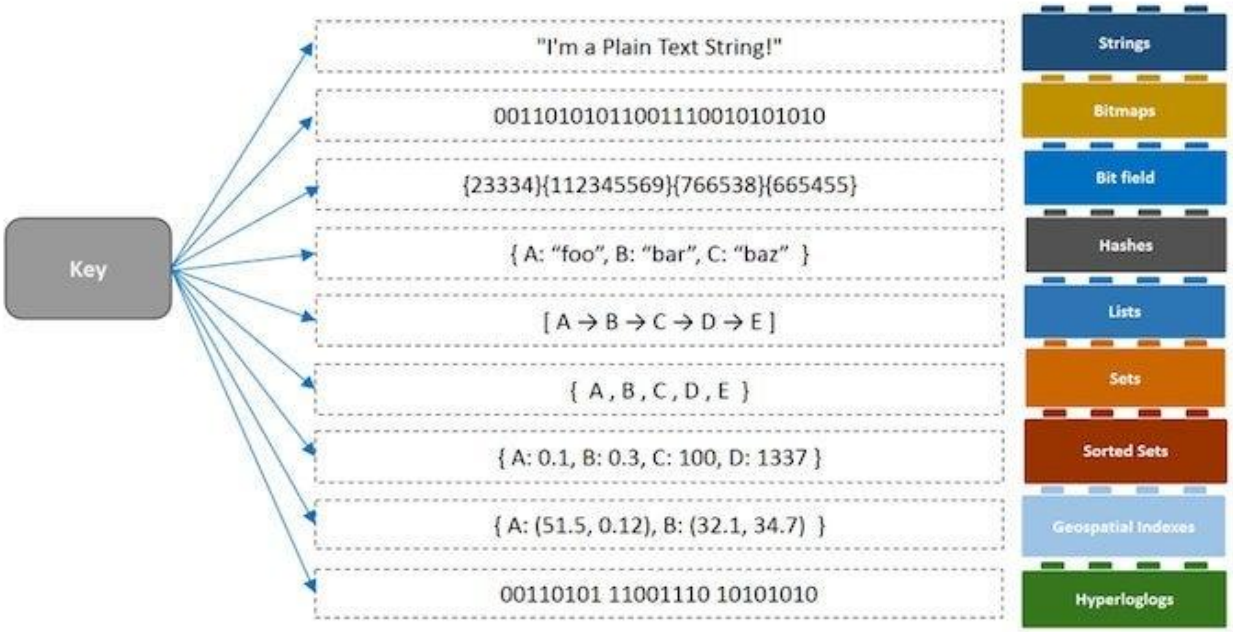

Рис. 1. **Структура зберігання даних у Redis**

Для MongoDB відсутність схеми означає, що два документа в структурі даних NoSQL не повинні мати однакові поля і можуть зберігати дані різних типів. Ось, наприклад, масив об'єктів, набір полів яких не збігається.

{ Model: "BMW", Color: "Red", Manufactured: 2016 },

{ Model: "Mercedes", Type: "Coupe", Color: "Black", Manufactured: "1-1-2017"}

Мова. У випадку кожної NoSQL бази даних мова відрізняється, для Redis це скриптова мова Lua, для MongoDB це модифікована версія Javascript.

Безпека. У Redis немає вбудованого контролю доступу на основі ролей (RBAC). Користувачі можуть отримати доступ до БД за допомогою простих імен користувачів і паролів, що не є дуже безпечним способом доступу до даних.

У MongoDB є кілька варіантів доступу до оболонки. Ви можете використовувати просте ім'я користувача та пароль, або інші інструменти, такі як RBAC та LDAP, вони легко інтегруються з MongoDB.

Маштабування. Redis пропонує реплікацію master-slave, що означає, що можна оновлювати один або кілька slave-серверів. Використовуючи Redis Enterprise, ви можете мати менеджер кластерів і архітектуру кластера без спільного доступу. Redis також підтримує шардинг і повторний шардинг, пропонуючи високу доступність і реплікацію в пам'яті

Що стосується MongoDB, то вона трохи гнучкіша. MongoDB має реплікацію з одним master і вбудованим авто-вибором. Це означає, що ви можете налаштувати другу базу даних, яку можна вибрати автоматично, якщо початкова база даних недоступна.

Продуктивність. MongoDB є безсхемною, що означає, що можна вставляти дані з різною структурою. Вона не перевіряє схему перед вставленням, що робить його надзвичайно швидкою. Але Redis працює в оперативній пам'яті тому при роботі с не дуже великими значеннями ключів він може бути швидшим за MongoDB.

Продуктивність буде продемонстрована на прикладі «YCSB workloads» [10]. Тестування продуктивності проведено на 10 мільйонах записів. Платформа для тестування – сервер з 32 гігабайтами оперативної пам'яті та 16 ядрами центрального процесору. Перший тест це читання по основному ключу, як видно із графіку на (рис. 2). Redis показав приблизно на 8000 операцій за секунду більше ніж MongoDB.

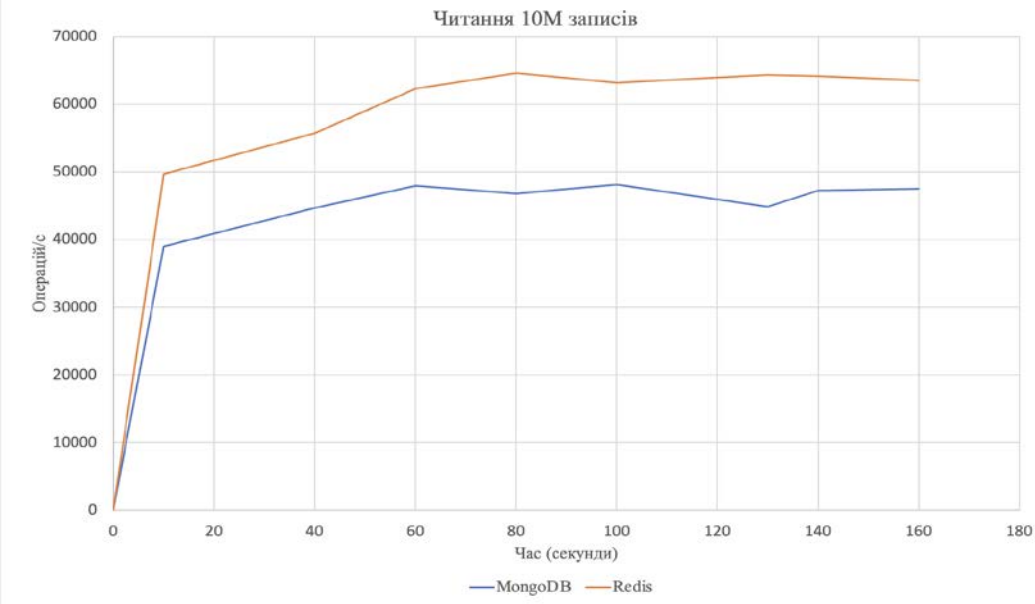

Рис. 2. **Читання 10 мільйонів записів**

У тесті оновлення даних також перемагає Redis (рис. 3). Під час цього тесту виконувались операції пошуку і оновлення. Варто відмітити, що при оновлені в Redis просто відбувається заміна значення ключа, а в MongoDB функція update призводить до досить важкої операції зсуву на диску.

Останній тест це читання останнього доданого запису (рис. 4) і тут вперше MongoDB виявляється швидшою за Redis. Це легко можна пояснити тим що с ранніх версій MongoDB містить вбудовану систему розумного кешування записів, тому для часто повторюваних запитів або для нещодавно доданих документів буде виконуватись вибірка з кешу в оперативній пам'яті, а не з фізичного диску.

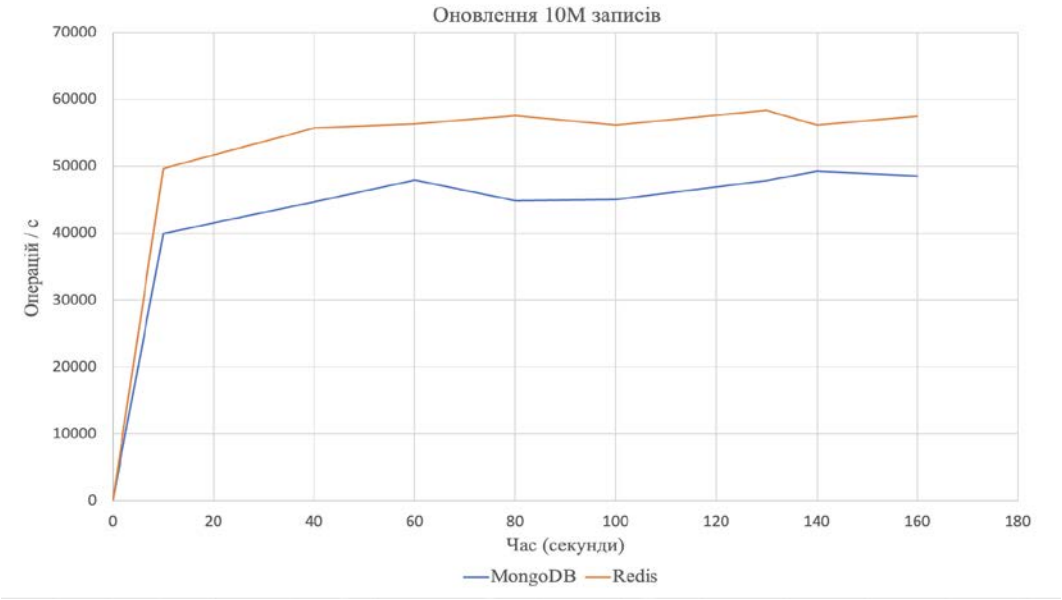

Рис. 3. **Оновлення 10 мільйонів записів**

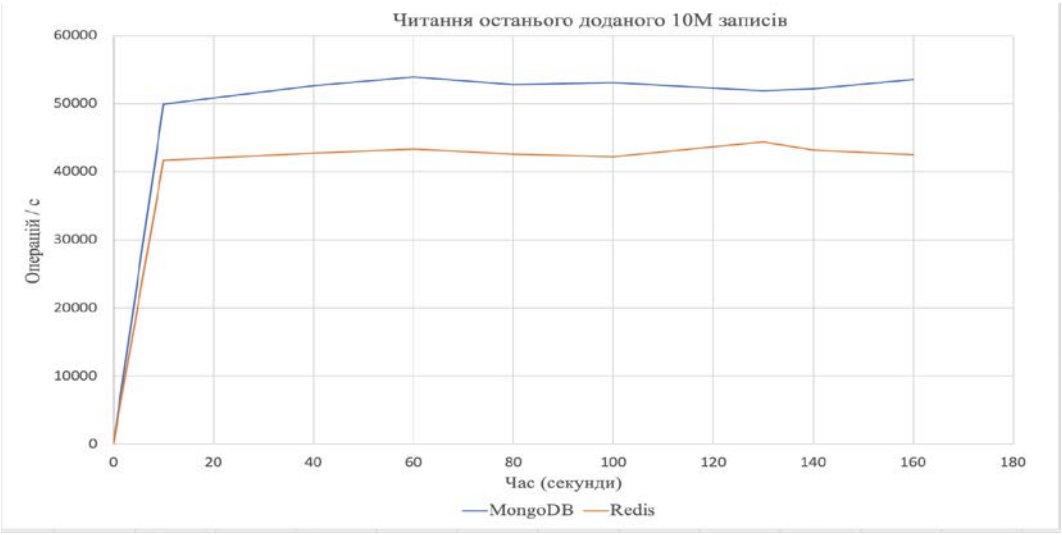

Рис. 4. **Читання останнього доданого запису**

Переваги MongoDB. MongoDB – це більш «традиційна» звичайна база даних, яка має розширені функції обробки даних. Зручність в управлінні: СУБД не потребує окремому адміністратора бази даних. Завдяки достатній зручності у використанні, їй легко можуть користуватися як розробники, так і системні адміністратори. MongoDB має розширену мову запитів. MongoDB пропонує багатодокументні ACID транзакції з версії 4.0. Рівень захищених сокетів (SSL), контроль доступу на основі ролей (RBAC) і масштабування – все це вбудовано в основу платформи.

Динамічна схема: як згадувалося вище, ця СУБД дозволяє гнучко працювати зі схемою даних без необхідності змінювати самі дані. Якщо користувач уже використовує MongoDB як своє основне рішення для баз даних, то його операційні витрати та витрати на розробку стануть досить низькими, оскільки користувачі матимуть лише одну основну базу даних для вивчення та керування, а не кілька баз даних MongoDB.

Переваги Redis: надзвичайно просто налаштувати з «коробки»; дуже простий у використанні та має відносно невелику криву навчання; забезпечує можливість зберігання на

диск, яку користувачі можуть налаштувати, і, як наслідок, легко впоратися з створенням кешу заново у разі збою системи; це одне з кращих рішенн для кешування, яке добре виконує свою роль. Він також має розширені структури даних, які дозволяють використовувати багато потужних і корисних способів збереження та запиту даних, які неможливо зробити за допомогою базового кешу ключ-значення.

Практична реалізація чату. Розробка чату почалась з вибору СУБД, в проекті вже використовуються такій СУБД як: MongoDB, MySQL, PostgreSQL та Redis. Використовувати реляційну СУБД в рамках розробки цього функціоналу не доцільно, тому що бізнес вимоги описані лише для першої версії і надалі можуть сильно змінитись, а тоді потрібно буде міняти структуру БД. Також немає сенсу у використанні реляційної структури тому що за вимогами чат не має зав'язків з медичними даними пацієнтів які зберігаються в PostgreSQL. На вибір залишились дві NoSQL СУБД Redis та MongoDB, хоча в тестах швидкодії Redis трохи обійшов MongoDB, вона компенсує це своєю гнучкістю та функціональністю. Отже для розробки функціоналу було обрано СУБД MongoDB, також застосовувались технології .NET 6.0 та бібліотека SignalR Core для роботи з вебсокет підключеннями [11, 12, 13].

WebSocket – забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. Він може використовуватись будь-яким клієнт-серверним застосунком. WebSocket був стандартизований W3C, протокол WebSocket стандартизований IETF як RFC 6455.

Далі розглянемо сутності зберігання даних і основні методи шару бізнес логіки. Основною сутністю є чат, у нього є зв'язок з прийомом(ContextId) в рамах якого він відбувається. Також в сутності чату знаходиться змінна загальної кількості повідомлень для подальшого розрахунку кількості непрочитаних повідомлень учасника:

```
[BsonIgnoreExtraElements, Table("rtc_chats")]
public class ChatEntity
{
 [BsonId, BsonRepresentation(BsonType.String)]
 public Guid Id { get; set; }

 [BsonElement("cid")]
 public string ContextId { get; set; }

 [BsonElement("ct")]
 public byte ContextType { get; set; }

 [BsonElement("p")]
 public IEnumerable<ExtendedParticipantEntity> Participants { get; set; }

 [BsonElement("mc")]
 public long MessagesCount { get; set; } = 0;

 [BsonElement("cat")]
 public DateTime CreatedAt { get; set; }

 [BsonElement("uat")]
 public DateTime UpdatedAt { get; set; }
}
```

У кожного чату є учасники, в першій ітерації функціоналу учасників може бути три: лікар, пацієнт і пов'язана особа пацієнта яка має доступ до прийому(наприклад мати для неповнолітнього пацієнта). Для опису учасника є класи ExtendedParticipantEntity і ParticipantEntity:

```
[BsonIgnoreExtraElements]
public class ParticipantEntity
{
 [BsonElement("uid")]
 public string UserId { get; set; }

 [BsonElement("rid")]
 public string ResourceId { get; set; }

 [BsonElement("r")]
 public byte Role { get; set; }
}
```

Клас ExtendedParticipantEntity окрім ідентифікаторів учасника містить ще змінні кількості відправлених повідомлень, відправлених вкладень та прочитаних повідомлень:

```
[BsonIgnoreExtraElements]
public class ExtendedParticipantEntity : ParticipantEntity
{
 [BsonElement("smc")]
 public int SentMessagesCount { get; set; }

 [BsonElement("sac")]
 public int SentAttachmentsCount { get; set; }

 [BsonElement("rmc")]
 public int ReadMessagesCount { get; set; }
}
```

Клас MessageEntity описує сутність повідомлення у чаті, він містить посилання на чат, масив вкладень, та інформацію про учасників які прочитали це повідомлення:

```
[BsonIgnoreExtraElements, Table("rtc_messages")]
public class MessageEntity
{
 [BsonId, BsonRepresentation(BsonType.String)]
 public Guid Id { get; set; } = Guid.NewGuid();

 [BsonElement("cid"), BsonRepresentation(BsonType.String)]
 public Guid ChatId { get; set; }

 [BsonElement("t")]
 public string Text { get; set; }

 [BsonElement("a")]
 public IEnumerable<AttachmentEntity> Attachments { get; set; }

 [BsonElement("rby")]
 public HashSet<string> ReadBy { get; set; }

 [BsonElement("cby")]
 public ParticipantEntity CreatedBy { get; set; }

 [BsonElement("cat")]
 public DateTime CreatedAt { get; set; }

 [BsonElement("uat")]
```

```
public DateTime UpdatedAt { get; set; }
}
```

В класі ChatsService є три основні методи. Перший це отримати чат по прийому або створити чати якщо такого чату ще не існує. В середині методу відбуваються перевірки чи має користувач доступ до цього прийому, а також відбуваються запит на інший сервіс для отримання деталей по прийому, якщо чату ще не існує буде створений новий і додано учасників до нього:

```
public async ValueTask<ServiceResponse<ChatModel, ValidationResult>> GetOrCreateChatAsync(string
eventId, ParticipantModel requester)
{
if (string.IsNullOrEmpty(eventId))
{
return new ServiceResponse<ChatModel, ValidationResult>(
new ValidationResult()
{Errors = {new ValidationFailure("eventId", _localizer.Localize(ValidationMessages.Required))}},
ServiceResponseStatuses.ValidationFailed);
}

var chat = await _chatsRepository.GetOneEventChatAsync(eventId);

if (chat is not null)
{
if (!chat.Participants.Select(s => s.RelativeUserId).Contains(requester.RelativeUserId))
{
return new ServiceResponse<ChatModel, ValidationResult>(
new ValidationResult()
{Errors = {new ValidationFailure("", _localizer.Localize(ValidationMessages.AccessDenied))}},
ServiceResponseStatuses.Forbidden);
}

return new ServiceResponse<ChatModel, ValidationResult>(chat);
}

var eventRequest = await _eventsServiceProvider.GetOneAsync(eventId);

if (!eventRequest.IsSuccess)
{
return new ServiceResponse<ChatModel, ValidationResult>(eventRequest.Errors,
eventRequest.Status);
}

if (eventRequest.Result.PatientCreator != requester.RelativeUserId &&
eventRequest.Result.PatientId != requester.RelativeUserId &&
eventRequest.Result.ResourceId != requester.RelativeUserId)
{
return new ServiceResponse<ChatModel, ValidationResult>(
new ValidationResult()
{Errors = {new ValidationFailure("", _localizer.Localize(ValidationMessages.AccessDenied))}},
ServiceResponseStatuses.Forbidden);
}

var participants = new List<ExtendedParticipantModel>()
```

```
{
new()
{
ResourceId = eventRequest.Result.ResourceId,
UserId = eventRequest.Result.ResourceUserId,
Role = Roles.Doctor
},
new()
{
UserId = eventRequest.Result.PatientId,
Role = Roles.Patient
},
};

if (!string.IsNullOrEmpty(eventRequest.Result.PatientCreator) &&
eventRequest.Result.PatientId != eventRequest.Result.PatientCreator)
{
participants.Add(new ExtendedParticipantModel
{
UserId = eventRequest.Result.PatientCreator,
Role = Roles.LinkedPatient
});
}

chat = new ChatModel
{
ContextType = ContextTypes.Event,
ContextId = eventId,
CreatedAt = DateTime.Now,
UpdatedAt = DateTime.Now,
Participants = participants,
MessagesCount = 0
};
await _chatsRepository.CreateChatAsync(chat);

return new ServiceResponse<ChatModel, ValidationResult>(chat);
}
```

Метод SaveMessageAsync відповідає за збереження нового повідомлення надісланого до чату. В середині відбуваються перевірки на валідність моделі повідомлення і в разі якщо все добре повідомлення зберігається у базу даних і створюються фонові задачі на відправку сповіщень іншим учасникам чату:

```
public async ValueTask<ServiceResponse<MessageModel, ValidationResult>>
SaveMessageAsync(SaveMessageRequest request)
{
var validationResult = new SaveMessageRequestValidator(_localizer).Validate(request);
if (!validationResult.IsValid)
{
return new ServiceResponse<MessageModel, ValidationResult>(validationResult,
ServiceResponseStatuses.ValidationFailed);
}

var chat = await _chatsRepository.GetOneChatAsync(request.ChatId);
if (chat is null)
```

```
{
return new ServiceResponse<MessageModel, ValidationResult>(
new ValidationResult() {Errors = {new ValidationFailure("chat",
_localizer.Localize(ValidationMessages.EntityNotFound))}},
ServiceResponseStatuses.NotFound);
}

var eventRequest = await _eventsServiceProvider.GetOneAsync(chat.ContextId);
if (!eventRequest.IsSuccess)
{
return new ServiceResponse<MessageModel, ValidationResult>(eventRequest.Errors,
eventRequest.Status);
}

if (DateTime.TryParse(eventRequest.Result.DateBegin, out var eventDate) && eventDate.Date <
DateTime.Today)
{
return new ServiceResponse<MessageModel, ValidationResult>(
new ValidationResult()
{Errors = {new ValidationFailure("", _localizer.Localize(ValidationMessages.AccessDenied))}},
ServiceResponseStatuses.Forbidden);
}

var requestHasAttachments = request.Attachments?.Any() ?? false;
var creatorParticipant =
chat.Participants.FirstOrDefault(x => x.RelativeUserId == request.CreatedBy.RelativeUserId);

if (creatorParticipant is null)
{
return new ServiceResponse<MessageModel, ValidationResult>(
new ValidationResult() {Errors = {new ValidationFailure("",
_localizer.Localize(ValidationMessages.AccessDenied))}},
ServiceResponseStatuses.Forbidden);
}

if (creatorParticipant.SentMessagesCount >= 1000)
{
return new ServiceResponse<MessageModel, ValidationResult>(
new ValidationResult()
{
Errors =
{
new ValidationFailure("",
_localizer.Localize(ValidationMessages.MaxMessagesCountExceeded)
.Replace(ValidationMessages.LocalizationComparisonAttribute, "1000"))
}
},
ServiceResponseStatuses.ValidationFailed);
}

if (requestHasAttachments && creatorParticipant.SentAttachmentsCount >= 15)
{
return new ServiceResponse<MessageModel, ValidationResult>(
```

```
new ValidationResult()
{
Errors =
{
new ValidationFailure("",
_localizer.Localize(ValidationMessages.MaxAttachmentsCountExceeded)
.Replace(ValidationMessages.LocalizationComparisonAttribute, "15"))
}
},
ServiceResponseStatuses.ValidationFailed);
}


var message = new MessageModel()
{
Attachments = request.Attachments,
CreatedAt = DateTime.Now,
UpdatedAt = DateTime.Now,
CreatedBy = request.CreatedBy,
ReadBy = new HashSet<string>() {request.CreatedBy.RelativeUserId},
Text = request.Text,
ChatId = chat.Id
};

await _chatsRepository.SaveMessageAsync(message);
await _chatsRepository.IncrementCountersAsync(chat.Id, creatorParticipant, true, true,
requestHasAttachments);
foreach (var participant in chat.Participants.Where(x => x.RelativeUserId !=
creatorParticipant.RelativeUserId))
{
await _notificationQueue.CreateAsync(new ChatNotificationTask(message.Id, chat.Id,
participant.RelativeUserId,
(byte) participant.Role, chat.ContextId));
}
//TODO transaction

return new ServiceResponse<MessageModel, ValidationResult>(message);
}
```

Метод MarkAsRead відповідає за відмітку повідомлення як прочитаного. В середині відбуваються перевірки на валідність моделі повідомлення і в разі якщо все добре у моделі повідомлення зберігається відмітка що учасник його прочитав:

```
public async ValueTask<ServiceResponse<MessageModel, ValidationResult>>
MarkAsReadAsync(MarkAsReadRequest request)
{
// ReSharper disable once MethodHasAsyncOverload
var validationResult = new MarkAsReadRequestValidator(_localizer).Validate(request);
if (!validationResult.IsValid)
{
return new ServiceResponse<MessageModel, ValidationResult>(validationResult,
ServiceResponseStatuses.ValidationFailed);
}
var message = await _chatsRepository.GetOneMessageAsync(request.MessageId);
if (message is null)
```

```
{
return new ServiceResponse<MessageModel, ValidationResult>(new ValidationResult()
{Errors = {new ValidationFailure("message", _localizer.Localize(ValidationMessages.EntityNotFound))}},
ServiceResponseStatuses.NotFound);
}

if (message.ReadBy.Contains(request.ReadBy.RelativeUserId))
{
await _notificationQueue.CompleteForMessageAsync(message.Id, request.ReadBy.RelativeUserId);
message.IsReadByRequester = true;
return new ServiceResponse<MessageModel, ValidationResult>(message);
}

message.ReadBy.Add(request.ReadBy.RelativeUserId);
await _chatsRepository.UpdateMessageAsync(message);
await _chatsRepository.IncrementCountersAsync(request.ChatId, request.ReadBy, false, true, false);
await _notificationQueue.CompleteForMessageAsync(message.Id, request.ReadBy.RelativeUserId);
message.IsReadByRequester = true;
return new ServiceResponse<MessageModel, ValidationResult>(message);
}
```

Таким чином у результаті розробки було створено новий сервіс, який відповідає за чат між лікарем і пацієнтами. Остаточний результат разом з клієнтською частиною наведено на рис. 5 та 6.

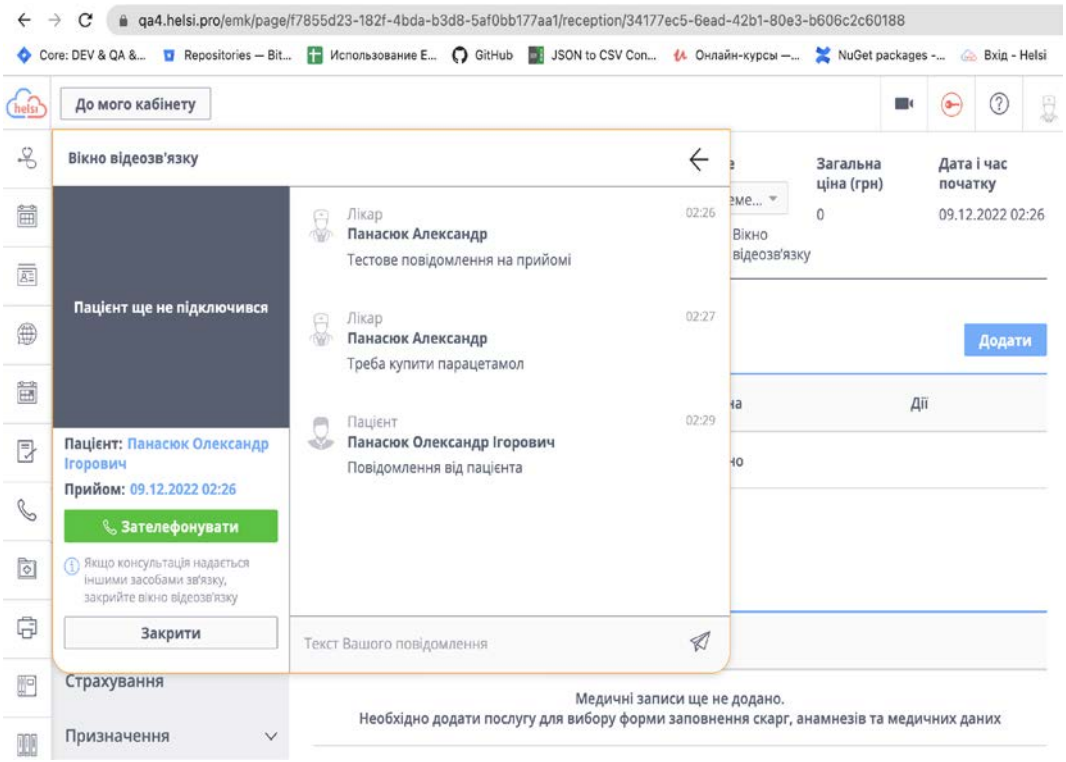

Рис. 5. **Інтерфейс чату у системі helsi.pro**

За допомогою використання СУБД MongoDB і її динамічної структури даних для накопичення і агрегації вдалося мінімізувати кількість логіки по формуванню великих зв’язаних сутностей у коді.

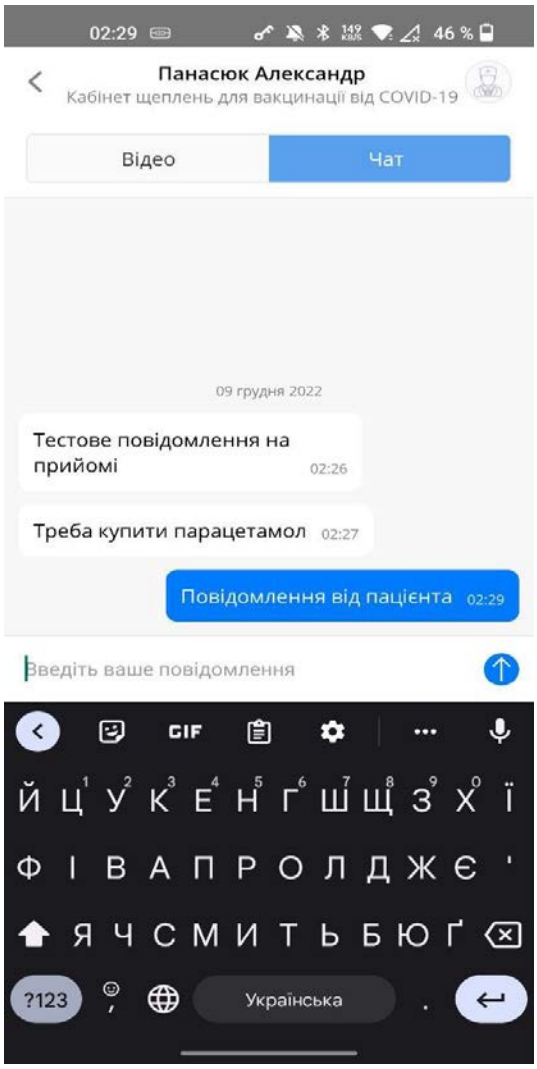

Рис. 6. **Інтерфейс чату у мобільному додатку helsi.me**

Висновки. Проведено аналітичне дослідження і порівняння СУБД MongoDB та Redis за характеристиками продуктивності, безпечності, типом системи зберігання даних і видом внутрішньої мови. Запропоновано використання комбінації бібліотеки SignalR Core для платформи .NET 6 разом з СУБД MongoDB для забезпечення управління підключеннями по протоколу Websocket у високонавантаженій медичній інформаційній системі при проведені онлайн чату між пацієнтом і лікарем. Розроблено новий сервіс для забезпечення комунікації в реальному часі через вебсокет з'єднання для медичної інформаційної системи HELSI..

Розроблено теоретичні та методологічні підходи до порівняльної оцінки характеристик сучасних NoSQL СУБД і застосуванні гібридних підходів поєднання технологій, які використовують при розроблення прикладного програмного забезпечення при побудові програмних систем.

За допомогою мови програмування C# 10.0 та платформи .NET 6 і бази даних MongoDB розширено функціонал існуючої прикладної медичної інформаційної системи HELSI можливістю чату між лікарем і пацієнтом пі час онлайн прийомів. Сспроектовано базу даних для зберігання інформації пов'язаної з даним функціоналом. Підготовлено супутню документацію для розробки інтерфейсу щодо створеного функціоналу.

References

1. Medychni informatsiini systemy – vprovadzhuiemo u vashomu medychnomu zakladi [Medical information systems – we implement in your medical institution]. URL: https://www.medsprava.com.ua/article/544-qqq-17-m4-10-04-2017--medichn-nformatsyn-sistemi-vprovadjumo-u-vashomu-medichnomu-zaklad.
2. Zahorodnii, H. M., Zynov'iev, Ye. S., Martynov, V. M., Shadura, V. M. (2005). HRID – nova obchysliuvalna tekhnolohiia dlia nauky [GRID – a new computing technology for science]. *Visnyk NAN Ukrainy = Bulletin of the NAS of Ukraine*, № 6, P. 17–19 [in Ukrainian].
3. Panasiuk, O. I., Pleskach, V. L., Statsenko, V. V., Khomaziuk, V. A. (2021). Rozrobka medychnoi informatsiinoi systemy dlia medychnykh zakladiv pervynnoi lanky [Development of Medical Information System for Primary Medical Institutions]. *Tekhnolohii ta inzhynirynh = Technologies and Engineering*. № 6, P. 9–18 [in Ukrainian].
4. eHealth: website. URL: https://ehealth.gov.ua/.
5. Helsi.me. Helsi: website. URL: https://helsi.me/.
6. Redis vs MongoDB: 10 Critical Differences. URL: https://hevodata.com/learn/redis-vs-mongodb/#perf.
7. Redis vs Mongodb: which one should you choose in 2022? URL: https://www.sfappworks.com/blogs/redis-vs-mongodb-which-one-should-you-choose.
8. MongoDB vs Redis: Complete Comparision in 2022. URL: https://naiveskill.com/mongodb-vs-redis/
9. Bradshaw, S., Brazil, E., Chodorow, K. (2019). MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd Edition. O'Reilly Media. 511 p.
10. YCSB Workloads. URL: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads.
11. Price, M. J. (2019). C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development. Packt Publishing. 818 p.
12. Albahari, J., Albahari, B. (2008). LINQ Pocket Reference: Learn and Implement LINQ for .NET Applications. O'Reilly Media. 174 p.
13. Nagel, C. (2021). Professional C# and .NET, 2021st Edition. O'Reilly Media. 31 p.

Література

1. Медичні інформаційні системи – впроваджуємо у вашому медичному закладі. URL: https://www.medsprava.com.ua/article/544-qqq-17-m4-10-04-2017--medichn-nformatsyn-sistemi-vprovadjumo-u-vashomu-medichnomu-zaklad.
2. Загородній Г. М., Зинов'єв Є. С., Мартинов В. М., Шадура В. М. ГРІД – нова обчислювальна технологія для науки. *Вісник НАН України*. 2005. № 6. С. 17–19.
3. Панасюк О. І., Плескач В. Л., Стаценко В. В., Хомазюк В. А. Розробка медичної інформаційної системи для медичних закладів первинної ланки. *Технології та інжиніринг*. 2021. № 6(152). С. 9–18.
4. eHealth: website. URL: https://ehealth.gov.ua/.
5. Helsi.me. Helsi: website. URL: https://helsi.me/.
6. Redis vs MongoDB: 10 Critical Differences. URL: https://hevodata.com/learn/redis-vs-mongodb/#perf.
7. Redis vs Mongodb: which one should you choose in 2022? URL: https://www.sfappworks.com/blogs/redis-vs-mongodb-which-one-should-you-choose.
8. MongoDB vs Redis: Complete Comparision in 2022. URL: https://naiveskill.com/mongodb-vs-redis/
9. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd Edition. O'Reilly Media, 2019. 511 p.
10. YCSB Workloads. URL: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads.
11. Price M. J. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development. Packt Publishing, 2019. 818 p.
12. Albahari J., Albahari B. LINQ Pocket Reference: Learn and Implement LINQ for .NET Applications. O'Reilly Media, 2008. 174 p.
13. Nagel C. Professional C# and .NET, 2021st Edition. O'Reilly Media, 2021. 31 p.

PANASIUK OLEKSANDR
Department of Applied Information Systems
Taras Shevchenko National University of Kyiv, Ukraine
https://orcid.org/0000-0003-3591-3689
Scopus Author ID: 57289566900
E-mail: vohigi@gmail.com

PLESKACH VALENTYNA
Doctor of Economic Sciences, Professor
Department of Applied Information Systems
Taras Shevchenko National University of Kyiv, Ukraine
https://orcid.org/0000-0003-0552-0972
Scopus Author ID: 12039392900
ResearcherID: HNS-5015-2023
E-mail: v_pleskach@ukr.net

MOLCHANOV BOHDAN
Department of Applied Information Systems
Taras Shevchenko National University of Kyiv, Ukraine
https://orcid.org/0000-0002-4633-9654
E-mail: molchanovbohdan@gmail.com

PANASIUK O. I., PLESKACH V. L., MOLCHANOV B. S.
Taras Shevchenko National University of Kyiv, Ukraine

EXPANDING THE ONLINE COMMUNICATION FUNCTIONALITY OF THE HELSI MEDICAL INFORMATION SYSTEM

Purpose. *Based on a comparative analysis of the two popular DBMS MongoDB and Redis, determine the best one for expanding the functionality of the existing applied information system. Create a new service to provide real-time communication via a websocket connection for the HELSI medical information system. When creating a database, take into account the possibility of high load and the need to add data aggregation in subsequent iterations of the functionality.*

Methodology. *The functionality was implemented using the C# 10.0 language, the ASP.NET Core 6.0 framework, and the MongoDB database.*

Findings. *An analytical study and comparison of the MongoDB and Redis DBMSs was carried out in terms of performance, security, type of data storage system and type of internal language. It is proposed to use a combination of the SignalR Core library for the .NET 6 platform together with the MongoDB DBMS to provide management of connections using the Websocket protocol in a highly loaded medical information system during an online chat between a patient and a doctor. A new service has been developed to provide real-time communication through a websocket connection for the medical information system HELSI.*

Originality. *Theoretical and methodological approaches to the comparative assessment of the characteristics of modern NoSQL DBMS and the application of hybrid approaches to the combination of technologies, which are used in the development of application software in the construction of software systems, have been developed.*

Practical value. *Using the C# 10.0 programming language and the .NET 6 platform and the MongoDB database, the functionality of the existing applied medical information system HELSI has been expanded with the possibility of a chat between the doctor and the patient during online appointments. A database has been designed to store information related to this functionality. Accompanying documentation for the development of the interface for the created functionality has been prepared.*

Keywords*: chat; online reception; client-server architecture; medical information system; data aggregation.*