НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ ТА УПРАВЛІННЯ

**КУРСОВА РОБОТА**

з дисципліни «Сучасні технології програмування – 3. Технології створення програмних продуктів для мобільних платформ»

на тему: «Розробка соціальної мережі з елементами листування у вигляді чату»

Студентки 3 курсу групи ІП-71

спеціальності 121 "Інженерія

програмного забезпечення"

Каспрук А.А.

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ - 2020 рік

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна «Сучасні технології програмування - 3. Технології створення програмних продуктів для мобільних платформ»

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 3 Група ІП-71 Семестр 6

**ЗАВДАННЯ**

**на курсову роботу студентки**

Каспрук Анастасії Андріївни

|  |  |
| --- | --- |
| 1. Тема роботи | Розробка соціальної мережі з елементами листування у вигляді чату |
|  | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи | 15.05.2020 |

|  |  |
| --- | --- |
| 3. Вихідні дані до роботи | Макет програмного продукту, архітектура бази даних, архітектура бек-енду |
|  | |
|  | |

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
| Вибір ідеї, вибір інструментарію, створення інтерактивного прототипу застосунку, |
| архітектура застосунку, імплементація UI застосунку, імплементація бізнес-логіки користувача, |
| імплементація взаємодії з мережею, імплементація зберігання даних локально, |
| імплементація взаємодії з hardware пристрою, імплементація тестів |

|  |  |
| --- | --- |
| 5. Дата видачі завдання | 20.03.2020 |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Вибір ідеї, вибір інструментарію | 21.03.2020 |  |
| 2. | Створення інтерактивного прототипу застосунку | 01.04.2020 |  |
| 3. | Архітектура застосунку | 15.04.2020 |  |
| 4. | Імплементація UI застосунку | 25.04.2020 |  |
| 6. | Імплементація бізнес-логіки користувача | 30.04.2020 |  |
| 5. | Імплементація взаємодії з мережею | 01.05.2020 |  |
| 7. | Імплементація зберігання даних локально | 03.05.2020 |  |
| 8. | Імплементація взаємодії з hardware пристрою | 10.05.2020 |  |
| 9. | Імплементація тестів | 14.05.2020 |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  |  |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | Іванова Л.А. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової роботи: 24 сторінки, 1 рисунок, 6 таблиць, 7 посилань.

Об’єкт дослідження: «Соціальна мережа з елементами листування у вигляді чату».

Мета роботи: розробити мобільний додаток для листування у вигляді чату.

У пояснювальній записці буде описано процес розробки серверної частини програмного продукту.

У даній курсовій роботі було спроектовано та реалізовано трирівневу архітектуру для серверної частини чату, що складається з таких функціональних рівнів: “Data access layer”, “Business layer” та “Presentation layer”. Архітектура додатку витримана з дотриманням принципів організації "Onion Architecture" ("цибулевої" архітектури).

Для розробки було обрано наступні технології:

1. Інтерфейс користувача – Xamarin;
2. Серверна частина – ASP.NET Core 3.0;
3. База даних – SQL Server;
4. База даних на пристрої користувача – SQLite.

Розробка включила в себе тестування програмної системи.

Результатами виконаної роботи буде програмний продукт у вигляді соціальної мережі з можливістю комунікації між користувачами у вигляді он-лайн чату.

ЗМІСТ

[ВСТУП 6](#_Toc40732877)

[1. ОПИС АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ 7](#_Toc40732878)

[1.1. ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РІВНЯ “DATA ACCESS LAYER” 9](#_Toc40732879)

[1.2. ОПИС АРХІТЕКТУРИ РІВНЯ “BUSINESS LAYER” 15](#_Toc40732880)

[1.3. ОПИС АРХІТЕКТУРИ РІВНЯ “PRESENTATION LAYER” 18](#_Toc40732881)

[2. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 21](#_Toc40732882)

[ВИСНОВОК 23](#_Toc40732883)

[ПЕРЕЛІК ПОСИЛАНЬ 24](#_Toc40732884)

# ВСТУП

Обраний вектор розробки – розробка серверної частини застосунку.

Ціль: розробити мобільний додаток для листування у вигляді чату.

Призначення курсової роботи: закріпити, поглибити та узагальнити базові теоретичні знання з розробки мобільних додатків та проектування архітектури проекту.

Загальна постановка завдання: розробити соціальну мережу, з можливістю реєстрації, розміщення особистої інформації та листування між двома довільними користувачами.

Під час виконання даної курсової роботи були досліджені та застосовані найпоширеніші загальноприйняті підходи та патерни для організації архітектури проекту. Зокрема, було використано принцип побудови архітектури програми, який носить назву "Onion Architecture". У процесі реалізації були використані стандартні патерни “Репозиторій” та “Одиниця роботи”. API проекту було побудовано на концепції MVC.

Для інтерфейсу користувача було створено макет за допомогою стороннього програмного забезпечення під назвою Figma.

Отриманий результат проходив тестування двох видів:

1. Мануальне тестування;
2. Юніт тести з серверної сторони.

# ОПИС АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

Першим етапом у процесі реалізації бекенду курсового проекту була розробка архітектури програмної системи.

У даній курсовій роботі використано принцип побудови архітектури програми, який носить назву "Onion Architecture" ("цибулева" архітектура).

Термін "Onion Architecture" був запропонований Джеффрі Палермо (Jeffrey Palermo) в 2008 році. Ця концепція стала досить популярною і є досить популярним архітектурним патерном.

Onion-архітектура являє собою поділ програми на рівні. При чому є один незалежний рівень, який знаходиться в центрі архітектури. Від цього рівня залежить другий рівень, від другого - третій і так далі. Тобто виходить, що навколо першого незалежного рівня нашаровується другий-залежний. Навколо другого нашаровується третій, який також може залежати і від першого. Образно це може бути виражено у вигляді цибулі, в якій також є серцевина, навколо якої нашаровуються всі інші рівні, аж до лушпиння.

Такий підхід дозволяє зробити програму більш сприйнятливою до розширень за рахунок позбавлення її від сильного зв’язку між функціональними рівнями. В основі даної архітектури лежить фундаментальне правило - будь-який модуль програми може залежати від ближчих до центру “цибулини” модулів, але не може залежати від більш далеких. Іншими словами, будь-яка зв'язаність повинна бути спрямована до центру “цибулини”.

Взаємодія між різними рівнями програми відбувається через інтерфейси. Це досить очевидно, оскільки за допомогою інтерфейсів можна позбутися від сильної зв’язаності, абстрагуючить від конкретних реалізацій взаємодіючих класів.

Кількість рівнів може варіюватися, але в центрі завжди знаходиться модель домену (Domain Model), тобто ті класи моделей, об'єкти яких зберігаються в базі даних.

Зовнішній рівень являє компоненти, які дуже часто змінюються. Зазвичай зовнішній рівень утворюють призначений для користувача інтерфейс, тести, допоміжні класи інфраструктури.

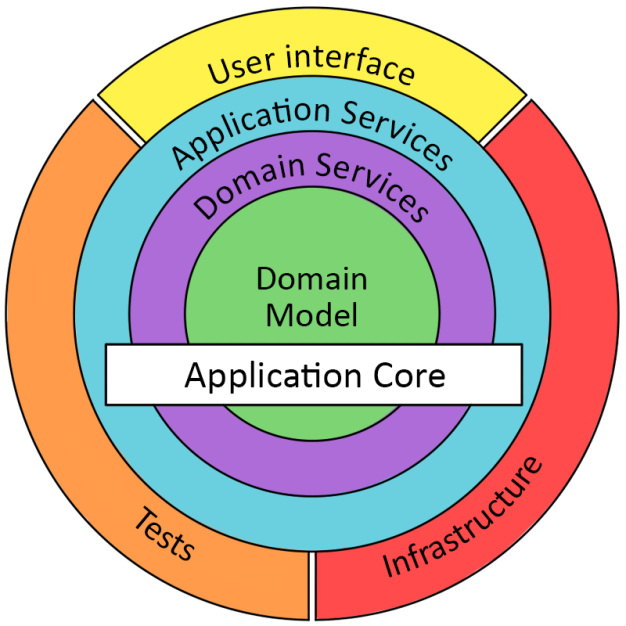


Рисунок 2.1 – Принцип побудови "цибулевої" архітектури

Цибулева архітектура тяжіє до ООП і ставить взаємодію об'єктів понад інші аспекти програми.

Найбільш класичною реалізацією "onion" архітектури є трирівнева система, яка передбачає поділ додатка на три рівні:

1. presentation layer (рівень представлення): це той рівень, з яким безпосередньо взаємодіє користувач; включає компоненти інтерфейсу для користувача, механізм отримання введенних від користувача даних;
2. business layer (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку отриманих від рівня представлення даних; реалізує необхідну логіку додатка, обчислення, взаємодіє з базою даних і передає рівню представлення результати обробки;
3. data access layer (рівень доступу до даних): зберігає моделі, що описують сутності, які використовуються; тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, з якими пізніше буде взаємодіяти рівень бізнес-логіки.

Даний курсовий проект було поділено на три функціональні рівні з урахуванням особливостей описаної вище архітектури:

1. рівень представлення (у роботі проект “Chat.API ”);
2. рівень бізнес-логіки (у роботі проект “Chat.Business”);
3. рівень доступу до даних (у роботі проект “Chat.DAL”, скорочено від “Data Access Layer”).

Кожен рівень реалізований у окремому проекті. Таке розділення обґрунтоване як логічно, так і з точки зору організації залежностей між проектами. Кожен рівень пов’язаний посиланням з близьким до нього з точки зору "onion" архітектури. Верхній рівень групує у собі послання на всі рівні, що лежать нижче.

Далі опис розробки архітектури для кожного з функціональних рівнів наведено більш детально.

## ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РІВНЯ “DATA ACCESS LAYER”

Проектування структури рівня збереження та доступу до даних – це перший та найважливіший етап розробки програмного забезпечення. Від того, наскільки грамотно це буде зроблено, залежить як зручність написання коду програми на наступних етапах роботи, так і швидкість її роботи.

База даних для даної курсової роботи створювалася за допомогою технології Entity Framework та підходу Code first взаємодії з базою даних.

Entity Framework - це інструмент, який спрощує зіставлення об'єктів у програмі з таблицями і стовпцями реляційної бази даних; це ORM - фреймворк, який є частиною .NET Framework.

Entity Framework передбачає три можливі способи взаємодії з базою даних:

1. database first: Entity Framework створює набір класів, які відображають модель існуючої бази даних;
2. model first: спочатку розробник створює шаблон, за яким пізніше Entity Framework створює реальну базу на сервері;
3. code first: розробник створює класи моделі даних, які будуть зберігатися в базі, потім Entity Framework за цими моделями генерує базу даних і її таблиці.

Після детального аналізу предметної області та створення класів-моделей, що зберігають необхідні дані, підхід Code first виявився найбільш зручним для створення бази даних.

Для предметної області були виділені наступні моделі рівня “Data access layer”:

1. ChatEntity – об’єкт, що представляє собою чат;
2. ChatUser – суміжний об’єкт, що служить для зв’язки користувач-чат;
3. Message - повідомлення у чаті;
4. User – користувач чату;
5. UserImage – фото користувача, яке зберігатиметься в окремій таблиці задля оптимізації.

Таблиця 2.1.1 – Опис властивостей моделей рівня “Data access layer”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва властивості** | **Призначення властивості** | **Тип властивості** | **Файл** |
| 1 | ChatEntity | ChatID | Унікальний ідентифікатор чату | int | ChatEntity.cs |
| 2 | ChatEntity | SenderID | Унікальний ідендифікатор ініціатора чату | int | ChatEntity.cs |

Продовження таблиці 2.1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва властивості** | **Призначення властивості** | **Тип властивості** | **Файл** |
| 3 | ChatEntity | ReceiverID | Унікальний ідентифікатор співрозмовника ініціатора чату | int | ChatEntity.cs |
| 4 | ChatEntity | Users | Колекція з інформацією про учасників чату | ICollection<ChatUser> | ChatEntity.cs |
| 5 | ChatEntity | Messages | Колекція з інформацією про повідомлення чату | ICollection<Message> | ChatEntity.cs |
| 6 | ChatUser | ChatUserID | Унікальний ідентифікатор пари користувач-чат | int | ChatUser.cs |
| 7 | ChatUser | ChatID | Унікальний ідентифікатор чату | int | ChatUser.cs |
| 8 | ChatUser | Chat | Посилання на об’єкт чату | ChatEntity | ChatUser.cs |
| 9 | ChatUser | UserID | Унікальний ідентифікатор користувача | int | ChatUser.cs |
| 10 | ChatUser | User | Посилання на об’єкт користувача | User | ChatUser.cs |
| 11 | Message | MessageID | Унікальний ідентифікатор повідомлення | int | Message.cs |
| 12 | Message | Text | Текст повідомлення | string | Message.cs |
| 13 | Message | ReceivedDate | Дата отримання повідомлення | DateTime | Message.cs |
| 14 | Message | IsRead | Ознака, що повідомлення було прочитано | bool | Message.cs |
| 15 | Message | SenderID | Унікальний ідентифікатор відправника | int | Message.cs |
| 16 | Message | Sender | Посилання на об’єкт відправника | User | Message.cs |
| 17 | Message | ChatID | Унікальний ідентифікатор чату | int | Message.cs |
| 18 | Message | Chat | Посилання на об’єкт чату | ChatEntity | Message.cs |
| 19 | User | UserID | Унікальний ідентифікатор користувача | int | User.cs |
| 20 | User | Name | Ім’я користувача | string | User.cs |
| 21 | User | PhoneNumber | Номер телефону користувача | string | User.cs |
| 22 | User | Email | Поштова скринька користувача | string | User.cs |
| 23 | User | PasswordHash | Хешована версія паролю користувача | string | User.cs |
| 24 | User | Token | Токен користувача | string | User.cs |
| 25 | User | Bio | Біографія користувача | string | User.cs |
| 26 | User | Image | Фото користувача | UserImage | User.cs |
| 27 | User | Chats | Колекція чатів користувача | ICollection<ChatUser> | User.cs |
| 28 | UserImage | UserImageID | Унікальний ідентифікатор фото користувача | int | UserImage.cs |
| 29 | UserImage | Image | Масив байт фото користувача | byte[] | UserImage.cs |
| 30 | UserImage | UserID | Унікальний ідентифікатор користувача | int | UserImage.cs |
| 31 | UserImage | User | Посилання на об’єкт користувача | User | UserImage.cs |

Для взаємодії з даними використовувалися стандартні патерни “Репозиторій” та “Одиниця роботи”.

“Репозиторій” дозволяє абстрагуватися від конкретних підключень до джерел даних, і є проміжною ланкою між класами, що безпосередньо взаємодіють з даними, і рештою програми.

“Одиниця роботи” використовується для інкапсулювання логіки роботи з джерелами даних та для забезпечення використання одного і того ж контексту даних репозиторіями.

У даній роботі взаємодія з даними здійснюється за допомогою наступних інтерфейсів та їх відповідних реалізацій:

1. IChatDbContext з реалізацією ChatDbContext – контекст бази даних, агрегатор таблиць з функціями доступу та збереження даних;
2. IChatRepository з реалізацією ChatRepository – реалізація патерну “Репозиторій”;
3. IChatUnitOfWork з реалізацією ChatUnitOfWork - реалізація патерну “Одиниця роботи”.

Таблиця 2.1.2 – Опис інтерфейсів і класів рівня “Data access layer”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 1 | IChatDbContext | Users | Доступ до таблиці користувачів | - | DbSet<User> | IChatDbContext.cs |
| 2 | IChatDbContext | Chats | Доступ до таблиці чатів | - | DbSet<ChatEntity> | IChatDbContext.cs |
| 3 | IChatDbContext | Messages | Доступ до таблиці повідомлень | - | DbSet<Message> | IChatDbContext.cs |
| 4 | IChatDbContext | ChatUsers | Доступ до суміжної таблиці користувач-чат | - | DbSet<ChatUser> | IChatDbContext.cs |
| 5 | IChatDbContext | SaveChanges | Збереження змін у БД | - | 1) int – кількість рядків, які було порушено | IChatDbContext.cs |

Продовження таблиці 2.1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 6 | IChatDbContext | Entry | Доступ до сутності в БД | 1) TEntity entity – сутність типу TEntity | 1) EntityEntry  <TEntity> - сутність БД, яка відслідковується ORM | IChatDbContext.cs |
| 7 | IChatRepository | Create | Створення сутності в БД | 1) T entity – сутність, яку треба створити | 1) int - кількість рядків, які було порушено | IChatRepository.cs |
| 8 | IChatRepository | Delete | Видалення сутності з БД | 1) ISpecification<T> deleteSpecification – специфікація, яка визначає умову для видалення | 1) T – сутність типу Т, яку було видалено | IChatRepository.cs |
| 9 | IChatRepository | FindAll | Знаходження всіх сутностей, які задовольняють умові | 1) ISpecification<T> specification – специфікація, яка визначає умову для пошуку  2) Func<IQueryable<T>, IQueryable<T>> includer – функція, яка визначає поля за посиланням, які необхідно включити у вибірку (аналог JOIN в SQL) | 1) List<T> - колекція сутностей типу Т, які задовольняють умовам пошуку | IChatRepository.cs |
| 10 | IChatRepository | FirstOrDefault | Знаходження першої сутності, яка задовольняє умові | 1) ISpecification<T> specification – специфікація, яка визначає умову для пошуку  2) Func<IQueryable<T>, IQueryable<T>> includer - функція, яка визначає поля за посиланням, які необхідно включити у вибірку (аналог JOIN в SQL) | 1) T – сутність типу Т, яка задовольняє умовам пошуку | IChatRepository.cs |
| 11 | IChatRepository | Update | Модифікація сутності, яка задовольняє умові | 1) T entity - сутність, яку треба модифікувати | 1) int - кількість рядків, які було порушено | IChatRepository.cs |

Продовження таблиці 2.1.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 12 | IChatRepository | TakeOrdered | Відбір фіксованої кількості відсортований сутностей, які задовольняють умові | 1) ISpecification<T> specification – специфікація, яка визначає умову для пошуку  2) Func<IQueryable<T>, IQueryable<T>> includer – функція, яка визначає поля за посиланням, які необхідно включити у вибірку (аналог JOIN в SQL)  3) Func<T, TSelector> orderKeySelector – функція, яка визначає поле, за яким необхідно відсортувати вибірку  4) int? skipCount – кількість записів у вибірці, які необхідно пропустити  5) int? takeCount – кількість записів, які необхідно відібрати  6) bool isDescending – ознака, що визначає необхідність відсортувати у спадаючому порядку | 1) List<T> - колекція сутностей типу Т, які задовольняють умовам пошуку | IChatRepository.cs |
| 13 | IChatUnitOfWork | MessagesRepository | Доступ до репозиторію повідомлень | - | IChatRepository<Message> | IChatUnitOfWork.cs |
| 14 | IChatUnitOfWork | UsersRepository | Доступ до репозиторію користувачів | - | IChatRepository<User> | IChatUnitOfWork.cs |
| 15 | IChatUnitOfWork | ChatsRepository | Доступ до репозиторію чатів | - | IChatRepository<ChatEntity> | IChatUnitOfWork.cs |
| 16 | IChatUnitOfWork | ChatUsersRepository | Доступ до репозиторію з суміжними даними користувач-чат | - | IChatRepository<ChatUser> | IChatUnitOfWork.cs |

## ОПИС АРХІТЕКТУРИ РІВНЯ “BUSINESS LAYER”

Наступним етапом після проектування рівня взаємодії з базою даних стало проектування рівня бізнес-логіки.

Основні задачі, які повинно виконувати дане програмне забезпечення, пов’язані з роботою зі користувачами та чатами як сутностями. Тому на рівні бізнес-логіки було виділено два основних інтерфейси та їх відповідні реалізації:

1. IChatsManager з реалізацією ChatsManager – робота з чатами;
2. IUserManager з реалізацією UserManager – робота з користувачами.

Таблиця 2.2.1 – Опис інтерфейсів і класів рівня “Business layer”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 1 | IChatsManager | GetAllChats | Отримання колекції всіх чатів користувача | 1) int userID – унікальний ідентифікатор користувача | 1) List<ChatShortInfo> - колекція з короткою інформацією про всі чати користувача | IChatsManager.cs |
| 2 | IChatsManager | GetChatMessages | Отримання колекції повідомлень чату | 1) int userID - унікальний ідентифікатор користувача  2) int chatID - унікальний ідентифікатор чату  3) int? skipCount – кількість повідомлень, які треба пропустити  4) int? takeCount – кількість повідомлень, які необхідно включити у вибірку | 1) List<MessageShortInfo> - колекція з даними про повідомлення чату | IChatsManager.cs |
| 3 | IChatsManager | StoreMessage | Збереження надісланого повідомлення | 1) MessageInfo newMessage – дані повідомлення | 1) MessageInfo – збережене повідомлення | IChatsManager.cs |

Продовження таблиці 2.2.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 4 | IUserManager | GetUserInfo | Отримання інформації про користувача | 1) int userId - унікальний ідентифікатор користувача для пошуку | 1) UserInfo – інформація про користувача | IUserManager.cs |
| 5 | IUserManager | GetUserInfo | Отримання інформації про користувача | 1) string userSearchInfo – ім’я, email або номер телефону користувача для пошуку | 1) UserInfo – інформація про користувача | IUserManager.cs |
| 6 | IUserManager | GetUserInfo | Отримання інформації про користувача | 1) UserLogin userLoginData – дані про логін користувача для пошуку | 1) UserInfo – інформація про користувача | IUserManager.cs |
| 7 | IUserManager | RegisterUser | Реєстрація користувача | 1) UserRegister newUser – дані нового користувача для реєстрації | 1) UserInfo – інформація про користувача | IUserManager.cs |
| 8 | IUserManager | UpdateUserInfo | Оновлення інформації користувача | 1) UserInfo newUserInfo – оновлена інформація про користувача | 1) UserInfo – інформація про користувача | IUserManager.cs |

Для реалізації даного рівня були введені наступні моделі даних:

1. ChatShortInfo – коротка інформація про чат;
2. MessageInfo – дані про повідомлення;
3. MessageShortInfo – коротка інформація про повідомлення;
4. UserEdit – інформація для модифікації даних про користувача;
5. UserInfo - інформація про користувача;
6. UserLogin – інформація для авторизації та аутентифікації користувача;
7. UserRegister – інформація для реєстрації користувача.

Таблиця 2.2.2 – Опис властивостей моделей рівня “ Business layer ”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва властивості** | **Призначення властивості** | **Тип властивості** | **Файл** |
| 1 | ChatShortInfo | ChatID | Унікальний ідентифікатор чату | int | ChatShortInfo.cs |
| 2 | ChatShortInfo | PartnerID | Унікальний ідентифікатор співрозмовника | int | ChatShortInfo.cs |
| 3 | ChatShortInfo | PartnerName | Ім’я співрозмовника | string | ChatShortInfo.cs |
| 4 | ChatShortInfo | PartnerImage | Фото співрозмовника | byte[] | ChatShortInfo.cs |
| 5 | ChatShortInfo | LastMessage | Останнє повідомлення | MessageShortInfo | ChatShortInfo.cs |
| 6 | MessageInfo : MessageShortInfo \* | ReceiverID | Унікальний ідентифікатор отримувача | int | MessageInfo.cs |
| 7 | MessageInfo : MessageShortInfo \* | ChatID | Унікальний ідентифікатор чату | int? | MessageInfo.cs |
| 8 | MessageShortInfo | Text | Текст повідомлення | string | MessageShortInfo.cs |
| 9 | MessageShortInfo | ReceivedDate | Дата отримання повідомлення | DateTime | MessageShortInfo.cs |
| 10 | MessageShortInfo | SenderID | Унікальний ідентифікатор відправника | int | MessageShortInfo.cs |
| 11 | MessageShortInfo | IsRead | Ознака, що повідомлення було прочитане | bool | MessageShortInfo.cs |
| 12 | UserEdit | Name | Ім’я користувача | string | UserEdit.cs |
| 13 | UserEdit | PhoneNumber | Номер телефону користувача | string | UserEdit.cs |
| 14 | UserEdit | Email | Електронна пошта користувача | string | UserEdit.cs |
| 15 | UserEdit | Bio | Біографія користувача | string | UserEdit.cs |
| 16 | UserEdit | Image | Фото користувача | byte[] | UserEdit.cs |
| 17 | UserInfo : UserEdit \* | UserID | Унікальний ідентифікатор користувача | int? | UserInfo.cs |
| 18 | UserInfo : UserEdit \* | AccessToken | Токен доступу користувача | string | UserInfo.cs |
| 19 | UserLogin | Login | Логін користувача | string | UserLogin.cs |
| 20 | UserLogin | LoginType | Тип логіну користувача | LoginType (PhoneNumber = 1, Email = 2) | UserLogin.cs |
| 21 | UserLogin | PasswordHash | Хешований пароль користувача | string | UserLogin.cs |
| 22 | UserRegister : UserLogin \* | Name | Ім’я користувача |  | UserRegister.cs |
| 23 | UserRegister : UserLogin \* | Image | Фото користувача |  | UserRegister.cs |

\* - позначка, що показує наслідування класів; це означає, що всі поля, властивості та методи базового класу стали частиною класу нащадка.

## ОПИС АРХІТЕКТУРИ РІВНЯ “PRESENTATION LAYER”

Для забезпечення повноцінної роботи мобільного додатку на рівні “Presentation layer” курсового проекту необхідно було надати можливість маніпулювання даними користувачів та чатів. Для цього було створено RESTful API з двома контролерами. Для передачі повідомлень у режимі реального часу була використана технологія SignalR Core.

SignalR Core представляє бібліотеку від компанії Microsoft, яка призначена для створення додатків, що працюють в режимі реального часу. SignalR використовує двонаправлений зв'язок для обміну повідомленнями між клієнтом і сервером, завдяки чому сервер може відправляти в режимі реального часу дані клієнтам.

API проекту було побудовано на концепції MVC.

Model-View-Controller (MVC, «Модель-Представлення-Контролер», «Модель-Вид-Контролер») - схема поділу даних програми, інтерфейсу користувача і керуючої логіки на три окремих компоненти: модель, представлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно.

Модель (Model) надає доступ до даних і реагує на команди контролера, змінюючи свій стан.

Представлення (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.

Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

Як вже зазначалося, на даному рівні було реалізовано два контролера:

1. ChatController – кінцева точка роботи з чатами;
2. UserController - кінцева точка роботи з користувачами.

Для передачі повідомлень у режимі реального часу був реалізований SignalR Core Hub ChatHub.

Таблиця 2.3.1 – Опис інтерфейсів і класів рівня “Presentation layer”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Назва інтерфейсу / класу** | **Назва функції / властивості** | **Призначення функції / властивості** | **Опис вхідних параметрів** | **Опис вихідних параметрів / тип властивості** | **Файл** |
| 1 | ChatController | GetAllChats | Отримання всіх чатів користувача | - | 1) ObjectResult – HTTP результат зі всіма чатами користувача | ChatController.cs |
| 2 | ChatController | GetAllChatMessages | Отримання всіх повідомлень чату | 1) int chatID – унікальний ідентифікатор чату  2) int? skipCount – кількість повідомлень, які треба пропустити  3) int? takeCount – кількість повідомлень, які необхідно включити у вибірку | 1) ObjectResult - HTTP результат зі всіма повідомленнями чату | ChatController.cs |
| 3 | UserController | Info | Отримання інформації про користувача | 1) int userId - унікальний ідентифікатор користувача | 1) ObjectResult - HTTP результат з інформацією про користувача | UserController.cs |
| 4 | UserController | Info | Отримання інформації про користувача | 1) string userSearchInfo - ім’я, email або номер телефону користувача для пошуку | 1) ObjectResult - HTTP результат з інформацією про користувача | UserController.cs |
| 5 | UserController | Register | Реєстрація користувача | 1) UserRegister newUser - дані нового користувача для реєстрації | 1) ObjectResult - HTTP результат з інформацією про користувача | UserController.cs |
| 6 | UserController | Login | Авторизація користувача | 1) UserLogin userLoginData – авторизаційна інформація користувача | 1) ObjectResult - HTTP результат з інформацією про користувача | UserController.cs |
| 7 | UserController | Edit | Модифікація інформації про користувача | 1) UserEdit userEditInfo - оновлена інформація про користувача | 1) ObjectResult - HTTP результат з інформацією про користувача | UserController.cs |
| 8 | ChatHub | Send | Пересилання повідомлення | 1) MessageInput message – повідомлення, яке необхідно переслати отримувачу | - | ChatHub.cs |

Користуватися всіма методами (окрім Login та Register) API та Hub-у може тільки авторизований користувач. Для цього був реалізований клас ChatAuthenticationHandler, успадкований від стандартного класу AuthenticationHandler, з методом HandleAuthenticateAsync. Там виконується авторизація з використанням токену доступу, який надається користувачу під час реєстрації. Даний токен передається у хедері Authentication і звіряється з наявними у базі даних на кожен запит.

# ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Для класів API та бізнес-логіки були написані модульні (unit) тести за допомогою середовища Nunit.

NUnit - відкрите середовище юніт-тестування додатків для .NET. Воно було перенесено з мови Java (бібліотека JUnit). Перші версії NUnit були написані на J #, але потім весь код був переписаний на C # з використанням атрибутів .NET.

Всього у проектів є чотири класи з тестами:

1. ChatControllerTest - тестування методів класу ChatController;
2. UserControllerTest – тестування методів класу UserController;
3. ChatsManagerTest – тестування методів класу ChatsManager;
4. UserManagerTest - тестування методів класу UserManager.

Таблиця 3.1 – Опис тестів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Назва класу** | **Назва тесту** | **Призначення тесту** | **Файл** |
| 1 | ChatControllerTest | GetAllChatsTest | Перевірка методу GetAllChats за позитивним сценарієм | ChatControllerTest.cs |
| 2 | ChatControllerTest | GetAllChatsTestUnauthorized | Перевірка методу GetAllChats без передачі токену авторизації (негативний сценарій) | ChatControllerTest.cs |
| 3 | ChatControllerTest | GetAllChatMesagesTest | Перевірка методу GetAllChatMesages за позитивним сценарієм | ChatControllerTest.cs |
| 4 | ChatControllerTest | GetAllChatMesagesTestUnauthorized | Перевірка методу GetAllChatMesages без передачі токену авторизації (негативний сценарій) | ChatControllerTest.cs |
| 5 | UserControllerTest | InfoByUserIdTest | Перевірка методу Info з пошуком інформації про користувача за унікальним ідентифікатором | UserControllerTest.cs |
| 6 | UserControllerTest | InfoByEmailTest | Перевірка методу Info з пошуком інформації про користувача за поштовою скринькою | UserControllerTest.cs |
| 7 | UserControllerTest | InfoByPhoneTest | Перевірка методу Info з пошуком інформації про користувача за номером телефону | UserControllerTest.cs |
| 8 | UserControllerTest | RegisterTest | Перевірка методу Register за позитивним сценарієм | UserControllerTest.cs |
| 9 | UserControllerTest | LoginTest | Перевірка методу Login за позитивним сценарієм | UserControllerTest.cs |

Продовження таблиці 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ п/п** | **Назва класу** | **Назва тесту** | **Призначення тесту** | **Файл** |
| 10 | UserControllerTest | LoginTestNotFound | Перевірка методу Login для неіснуючого користувача (негативий сценарій) | UserControllerTest.cs |
| 11 | UserControllerTest | EditTest | Перевірка методу Edit за позитивним сценарієм | UserControllerTest.cs |
| 12 | UserControllerTest | EditTestUnauthorized | Перевірка методу Edit без передачі токену авторизації (негативний сценарій) | UserControllerTest.cs |
| 13 | UserControllerTest | EditTestInternalError | Перевірка методу Edit з емуляцією проблем з базою даних (негативний сценарій) | UserControllerTest.cs |
| 14 | ChatsManagerTest | GetAllChatsTest | Перевірка методу GetAllChats за позитивним сценарієм | ChatsManagerTest.cs |
| 15 | ChatsManagerTest | GetChatMessagesTest | Перевірка методу GetAllChatMesages за позитивним сценарієм | ChatsManagerTest.cs |
| 16 | ChatsManagerTest | StoreMessageTest | Перевірка методу StoreMessage за позитивним сценарієм | ChatsManagerTest.cs |
| 17 | UserManagerTest | GetUserInfoByIdTest | Перевірка методу GetUserInfo з пошуком інформації про користувача за унікальним ідентифікатором | UserManagerTest.cs |
| 18 | UserManagerTest | GetUserInfoByEmailTest | Перевірка методу GetUserInfo з пошуком інформації про користувача за поштовою скринькою | UserManagerTest.cs |
| 19 | UserManagerTest | GetUserInfoByPhoneTest | Перевірка методу GetUserInfo з пошуком інформації про користувача за номером телефону | UserManagerTest.cs |
| 20 | UserManagerTest | GetUserInfoByPhoneAndPasswordTest | Перевірка методу GetUserInfo з пошуком інформації про користувача за авторизаційною інформацією | UserManagerTest.cs |
| 21 | UserManagerTest | GetUserInfoByEmailAndPasswordTest | Перевірка методу GetUserInfo з пошуком інформації про користувача за авторизаційною інформацією | UserManagerTest.cs |
| 22 | UserManagerTest | RegisterUserTest | Перевірка методу RegisterUser за позитивним сценарієм | UserManagerTest.cs |
| 23 | UserManagerTest | UpdateUserInfoTest | Перевірка методу UpdateUserInfo за позитивним сценарієм | UserManagerTest.cs |

# ВИСНОВОК

Першим кроком при розробці чату було створення макету графічного інтерфейсу користувача з використанням новітніх технік та рекомендацій UI/UX для створення максимально комфортного для користування інтерфейсу. Це дозволило виділити основні сутності, які надалі були використані при проектуванні архітектури бази даних.

Наступним кроком була розробка WEB API серверної частини, яке відповідало за логіку програми. Основні задачі, які повинно виконувати дане програмне забезпечення, пов’язані з роботою зі користувачами та чатами як сутностями. Розроблене АРІ надає всі необіхні для цього механізми.

Паралельно з розробкою серверної частини проекту велася розробка графічного інтерфейсу користувача. Для аутентифікації та авторизації був обраний стандартний підхід з сторінкою для введення логіну та паролю, якщо такі існують, інакше ж необхідно перейти на сторінку реєстрації нового користувача. Пароль відразу ж шифрується за допомогою обраного алгоритму шифрування та передається на серверну частину у зашифрованому вигляді, що додає програмному продукту більшої безпеки для особистих даних користувача.

Режим передачі повідомлень у реальному часі забезпечується за допомогою технології SignalR Core.

В результаті виконання курсової роботи ми отримали програмний продукт, який повністю відповідає заданим вимогам та очікуваним результатам.

# ПЕРЕЛІК ПОСИЛАНЬ

1. SignalR Core. Первое приложение [Електронний ресурс] – Режим доступу <https://metanit.com/sharp/aspnet5/30.1.php> .
2. Введение в ASP.NET Core [Електронний ресурс] – Режим доступу <https://metanit.com/sharp/aspnet5/1.1.php>
3. Мова програмування C# та платформи .Net та .NET Core [Електронний ресурс] – Режим доступу https://codernet.ru/books/c\_sharp
4. Енциклопедія C# та платформи .Net [Електронний ресурс] – Режим доступу https://metanit.com/sharp/tutorial/
5. MS SQL Server 2017 [Електронний ресурс] – Режим доступу https://metanit.com/sql/sqlserver/
6. Керівництво ASP.NET Core 3 [Електронний ресурс] – Режим доступу https://metanit.com/sharp/aspnet5/
7. Програмування на Xamarin Forms [Електронний ресурс] – Режим доступу https://metanit.com/sharp/xamarin/