# АНОТАЦІЯ

Магістерська робота «Розробка соціальної мережі для бібліотек на основі .NET технологій» Гайдук Роман Степанович, Тернопільський національний технічний університет імені І. Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм–62, Тернопіль, 2019.

Пояснювальна записка містить: 0 с. 0 рис., 0 табл., 0 дод..

Метою роботи є розробка соціальної мережі для бібліотек на основі .Net технологій.

В ході роботи досліджено та проаналізовано предметну область, визначено ключових акторів системи, спроектовано ефективну базу даних, застосовано сучасний підхід до розробки, створено зручний дизайн, виконано тестування.

Розроблена система написана на мовах програмування TypeScript та C#. Використано каркас ASP.NET на основі високопродуктивної платформи .Net Core 2.0, мікросервісну архітектуру у зв'язці з патерном проектування Dependency injection, автентифікація за допомогою JSON Web Token, фреймворк Angular 8, Google books API, карти Mapbox.

Ключові слова: ANGULAR, .NET CORE, ASP NET, WEB API, GOOGLE BOOKS API, MAPBOX, DEPENDENCY INJECTION, МІКРОСЕРВІС, ФРЕЙМВОРК, КНИГА, БІБЛІОТЕКА, УНІФІКОВАНА СИСТЕМА, СОЦІАЛЬНА МЕРЕЖА.

# SUMMARY

Master's Degree «Social Network Development for Libraries Based on .NET Technologies» Roman Gaiduk, I. Pulyu Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPM–62 Group, Ternopil, 2019 .

The explanatory note contains: 0 p. 0 Figure, 0 Table, 0 Add ..

The purpose of the work is to develop a social network for libraries based on .Net technologies.

In the course of the work the subject area was researched and analyzed, key system actors were identified, an efficient database was designed, a modern approach to development was applied, a convenient design was created, testing was performed.

The system is written in TypeScript and C # programming languages. Used ASP.NET framework based on high–performance .Net Core 2.0 platform, microservice architecture in Dependency injection design pattern, JSON Web Token authentication, Angular 8 framework, Google books API, Mapbox maps.

Keywords: ANGULAR, .NET CORE, ASP NET, WEB API, GOOGLE BOOKS API, MAPBOX, DEPENDENCY INJECTION, MICROSERVICE, FRAMEWORK, BOOK, LIBRARY, UNIFIED SYSTEM, SOCIETY.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – Система управління базами даних.

ПЗ – Програмне забезпечення.

БД – База даних.

URL – Uniform Resource Locator (уніфікований локатор ресурсу).

URI – Uniform Resource Identifier (уніфікований ідентифікатор ресурсу).

UML – Unified Modeling Language (уніфікована мова моделювання).

UI – User Interface (інтерфейс користувача).

SQL – Structured Query Language (мова структурованих запитів).

SASS – Syntactically Awesome Stylesheets (метамова на основі CSS).

RUP – Rational Unified Process (раціональний уніфікований процес).

REST – Representational State Transfer (передача репрезентативного стану).

ORM – Object–Relational Mapping (об’єктно–реляційне відображення).

MVVM – Model–View–ViewModel (архітектурний шаблон).

MVC – Model–View–Controller (архітектурний шаблон).

JWT – JSON Web Token (відкритий стандарт створення токенів доступу).

JSON – JavaScript Object Notation (нотація об’єктів JavaScript).

DI – Dependency injection (впровадження залежностей).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

HTML – HyperText Markup Language (мова розмітки гіпертексту).

CSS – Cascading Style Sheets (каскадна таблиця стилів).

CSRF – Cross Site Request Forgery (міжсайтова підробка запиту).

CORS – Cross–Origin Resource Sharing (сумісне використання ресурсів між різними джерелами).

CLI – Command Line Interface (інтерфейс командного рядка).

API – Application Programming Interface (програмний інтерфейс

аплікації).

ЗМІСТ

# ВСТУП

Переходячи від концепції інформаційного суспільства до концепції суспільства знань, роль публічних бібліотек повинна зазнавати подібних пріоритетних змін. Якою має бути роль публічних бібліотек, коли більшість людей мають достатній доступ до безлічі інформації та розваг? Як бібліотекарі можуть визначити свою професійну роль, коли користувачі самі виконують багато функцій, які раніше були передані професіоналам?

Бібліотеки повинні перейти від визначення своєї професійної ролі з точки зору постачальників інформаційної, грамотності до ролі мультимодальних центрів знань, що охоплюють інформацію, а також розваги, пошук та виробництво.

Як відомо, публічні бібліотеки історично розвивалися в тандемі з індустріальним суспільством, і їх формування здебільшого грунтується на просвітницьких ідеалах свободи вираження поглядів та універсальному доступі до інформації та творів уяви. Стаття 19 Загальної декларації прав людини від 1948 р. є красномовним і унікальним вираженням цих ідеалів, вказуючи на те, що «кожен має право на свободу думок і самовираження; це право включає свободу дотримуватися думок без втручання та шукати, отримувати, передавати інформацію, ідеї через будь-які засоби масової інформації та незалежно від кордонів».

Як не менш відомо, в промислових суспільствах інформацію часто важко отримати з класичних джерел, і багато комунікативних технологій розробляються з метою усунення цього дефіциту, прискорення надійності та ефективності віддаленого спілкування. Можна просто подумати про негайний успіх телеграфа та телефону, чий відповідний процес у публічному та приватному житті був центрально залежним від їх майже миттєвого передавання даних, ефективності та безпосередності, що, в свою чергу, спричинило розвиток соціальних мереж та інших ресурсів обміну інформацією.

Бібліотеки слугують для усунення дефіциту інформації та забезпечення загального доступу. Визначення інформаційних ресурсів є чітким, оскільки це поняття має фізичну суттєвість: це книги, журнали, статті та інше. І роль бібліотекаря однаково чітка, хоча і не завжди проста у виконанні: бібліотекарі роблять чіткий вибір серед відомого репертуару артефактів.

Товарність в основному відчувається в країнах Європи, які мають давні традиційні засоби масової інформації, тобто радіо, телебачення та фільми, які служать суспільному благу, а не інтересам інвесторів, рекламодавців чи ЗМІ.

У певному сенсі проблеми, пов'язані з розвитком публічних бібліотек як неформальних центрів знань, як це було описано вище, повертають нас до деяких старих, але повторюваних питань бібліотечного обслуговування. Однак відповіді, які потрібно знайти, належать нашому теперішньому та найближчому майбутньому.

# 1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

## 1.1 Аналіз вимог до програмної системи

### 1.1.1 Аналіз предметної області

Люди скрізь потребують інформації – для покращення здоров'я, пошуку роботи, підвищення рівня життєдіяльності, підтримки навчання та багато іншого. Інтернет змінив те, як ми шукаємо та обмінюємось інформацією, на ринку є безліч онлайн сервісів та соціальних мереж які ми використовуємо в повсякденному житті. Виходячи з цього всього можна задати собі питання «Як не придумувати колесо», тому що конкурувати з великими корпорація дуже важко так як вони вже стали на ринку монополістами, але навіть щось таке просте, як колесо, можна переробити і вдосконалити. Інновації в бібліотеках можуть бути настільки ж простими, як і вдосконалення цього колеса!

Є три ключові причини, чому бібліотеки та спільнота читачів повинні впроваджувати інновації. По-перше, нам потрібно адаптуватися до мінливого середовища. Можна виділити дві сфери змін, які безпосередньо впливають на бібліотеки: перехід відображення інформації у цифровий формат, журнали вже зробили перший крок до цього, книги повинні пройти ще певний шлях; і другий мінливий спосіб це те яким чином користувачі шукають інформацію. Традиційні системи локальні системи стають менш актуальними так як охоплюють менший спектр аудиторії.

По-друге, потрібно вдосконалити існуючі продукти та послуги. Наприклад можна включати додаткове використання потужності, вдосконалення досліджуваних просторів, подальший розвиток програми.

По-третє, потрібно використовувати вже наявні потужності які надають різні транснаціональні компанії та впроваджувати лише інновації, що забезпечить набагато швидший розвиток. Однією з таких компаній є Google,

яка містить безмірну кількість інформації та доступних для використання API.

API Google книги це велика база даних створена для того щоб зробити вміст книг більш відкрити у пошуку. На основі даної платформи можна побудувати додаток який буде більш продуктивним та легким у використанні.

### 1.1.2 Постановка задачі

Завданням даної магістерської роботи є розробка соціальної мережі яка дозволяє створювати власну електронну бібліотек, виконувати обмін та моніторинг книга, поширювати інформацію та організовувати події. Для забезпечення виконання поставлених вимог хорошим варіантом буде використання веб технологій що зумовлено популярністю та доступністю для майбутніх користувачів. Соціальна мережа повинна використовуватись як і великими бібліотеками так і простими користувачами із власною колекцією книг. До основних компонентів можна віднести наступне:

- серверну частину яка буде забезпечувати взаємодію із базою даних, виконання обробки інформації отриманої від клієнта;
- клієнтська частину що забезпечить зручний та простий інтерфейс взаємодії користувача з іншими компонентами системи;
- проміжний шар фільтрування запитів для забезпечення безпеки всередині системи та блокування від несанкціонованого доступу до персональних даних;
- сервіси взаємодії із API Google книги та MapBox.

На основі зібраних даних із предметної області для реалізації даної системи можна виділити наступні фактори:

- можливість публікувати інформацію для інших користувачів та організація подій;
- створення власної електронної бібліотеки із доступом для перегляду інших користувачів;

- онлайн перегляд книги та відображення на карті користувачів з даною книгою;
- серверна частина має бути розроблена на .NET Core фреймворку яка забезпечує можливість розробки для різних платформ та побудови на основі каркасу ASP.NET;
- для написання клієнтської частини необхідно використати каркас Angular версії 8. Даний фреймворк використовується для написання одно сторінкових веб програм з використання мови програмування TypeScript та MVVM паттерном розробки;
- в якості інтегрованого середовища розробки використати Visual Studio 2019 та Visual Studio Code, які займають перше місце по зручності та швидкості розробки через своє автодоповнення та великої кількості розширень;
- більшість частина інформації по книгах для зручного додавання та забезпечення економії ресурсів необхідно отримувати з API Google книг;
- для робота з картами та локаціями потрібно використати безкоштовну бібліотеку MapBox;

Також необхідно врахувати максимально допустимі квоти при використанні зовнішних API.

### 1.1.3 Визначення акторів системи

Для написання ключових варіантів використання для початку необхідно визначити акторів системи.

Актор – суб’єкт у діаграмі UML який представляє тип ролі, коли він взаємодіє із системою та її об'єктами. Важливо зазначити, що актор завжди знаходиться поза сферою системи, яку ми прагнемо моделювати за допомогою діаграми UML.

Актори використовуються, щоб зобразити різні ролі, включаючи користувачів та інші зовнішні суб’єктів взаємодії. Актори представляються в

діаграмі UML, використовуючи позначення особи з палицею. Схематичний вигляд UML актору зображено на рисунку 1.1.


Рисунок 1.1 Схематичне зображення актора UML

Доступ до різних частин системи виконується на основі авторизації та автентифікації. В даному випадку можна виділити двох акторів, а саме гостя та авторизованого користувача. Список акторів та коротких опис їх можливостей взаємодії із системою зображено у таблиці 1.1.

Таблиця 1.1 – Актори системи

| Актор | Короткий опис |
|---|---|
| Гість | Користувач який має доступ до перегляду профілів зареєстрованих користувачів та книг із обмеженим вмістом без можливості створення та моніторингу угод оренди книг. |
| Користувач | Авторизований користувач. Має доступ до всього спектру функцій системи, а саме: редагування персональних даних, редагування власної бібліотеки, моніторинг виданих та взятих книг, публікування інформації |

1.1.4 Опис ключових варіантів використання

Діаграми прицидентів або іншими словами варіантів використання зазвичай називаються діаграмами поведінки, що використовуються для опису

набору дій (випадків використання), які деякі системи повинні або можуть виконуватись у співпраці з одним або декількома зовнішніми користувачами системи (суб'єктами). Кожен випадок використання повинен дати деякий помітний і цінний результат учасникам або іншим зацікавленим сторонам системи.

Можна зауважити, що UML 2.0 – 2.4 специфікації також описують варіанти використання як спеціалізації діаграми класів, а діаграма класів – це структурна схема. Також є структуровані діаграмами – як особливий випадок діаграм класів, де класифікатори мають бути або дійовими особами, або використовують випадки, пов'язані з асоціаціями.

На основі визначених акторів в попередньому розділі можна побудувати ключові варіанти використання.

Гість – користувач який має доступ до перегляду певних сторінок, або їх частин. Обмеження проявляється як на серверній так і на клієнтській частині системи. Нижче наведено варіанти використання для актора «Гість»:

- перегляд профілів користувачів та їх книг без можливості підписатись;
- перегляд деталей книги та розташування бібліотек чи користувачів з даною книгою на карті без можливості створення угоди про оренду чи обміну книги;
- загальний пошук книг та зареєстрованих користувачів;
- можливість реєстрації в системі. Для використання всього спектру функцій системи необхідно пройти реєстрацію з внесення персональних даних;
- можливість увійти у систему після реєстрації.

Визначивши всі варіанти використання актора «Гість» можна побудувати наступну UML діаграму (рисунок 1.2).


Рисунок 1.2 – Діаграма варіантів використання для актора «Гість»

Актор «Користувач» – суб’єкт системи який пройшов авторизацію та може входити в системи тому відповідні варіанти використання можна опустити в подальші побудові відповідної діаграми. Ключові варіанти використання наведено нижче:

- додавання, видалення книг до власної електронної бібліотеки книг;
- редагування даних власного аккаунту;
- перегляд постів та подій опублікованих користувачів яких відстежується;
- додавання постів та подій для власних читачів;
- додавання коментарів, позначення вподобання до посту;
- відзначення події як цікава або відвідувана;
- створення угод для обміну та оренди книг між користувачами;
- перегляд активних угод із зворотнім відліком по днях;
- завершення угоди після повернення книги.

Визначивши основні варіанти використання актора «Користувач» можна зобразити наступну UML діаграму (рисунок 1.3).


Рисунок 1.3 – Діаграма варіантів використання для актора «Користувач»

Варіанти використання кожного актора описанні в послідовності збільшення можливостей системи. Тобто актор «Користувач» включає та розширює варіанти використання актора «Гість». Після опису варіантів використання даних акторів можна побудувати загальну діаграму прицидентів, яка зображена на рисунку 1.4.

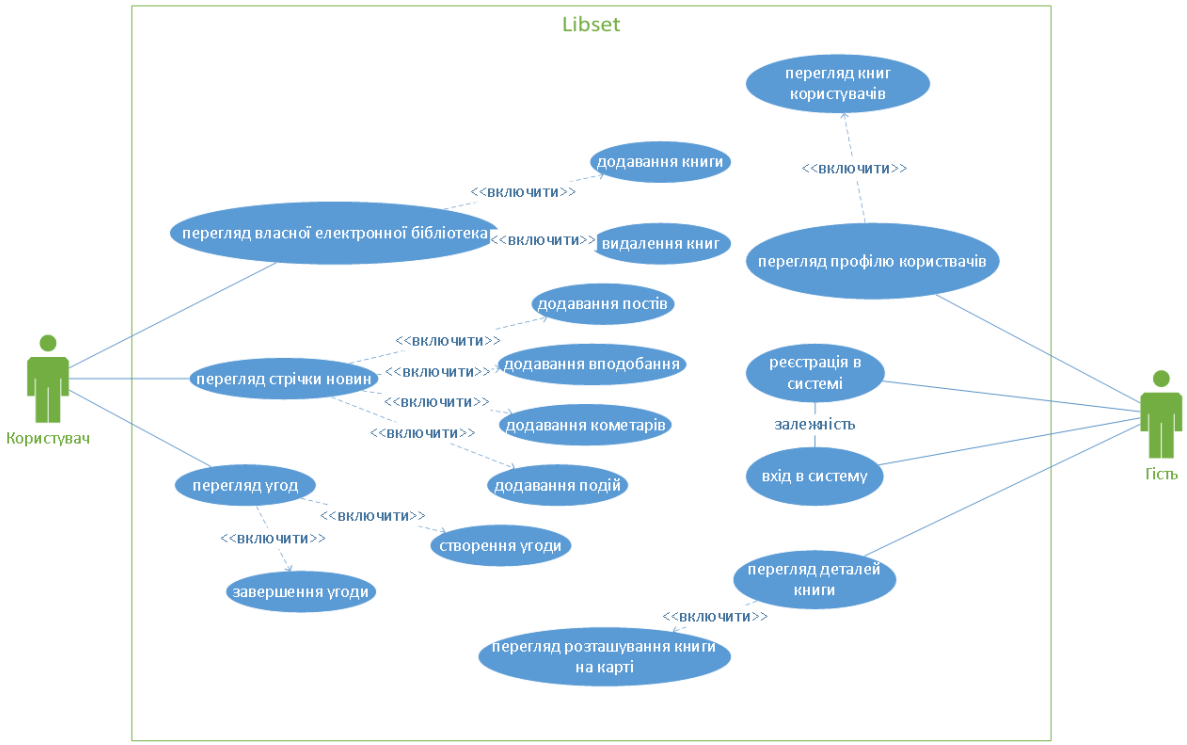


Рисунок 1.4 – Загальна діаграма прицидентів системи

## 1.2 Проектування програмної системи

### 1.2.1 Вибір процесу розробки

Раціональний об'єднаний процес (RUP) – це процес розробки програмного забезпечення. Він забезпечує дисциплінований підхід до розподілу завдань та обов'язків в межах організації. Мета даної методології - забезпечити виробництво високоякісного програмного забезпечення, яке відповідає потребам його кінцевих споживачів, у межах передбачуваного графіку та бюджету. Раціональний об'єднаний процес – це технологічний продукт, розроблений та підтримуваний програмою Rational Software. Команда розробок для Раціонального єдиного процесу тісно співпрацює із замовниками, партнерами, групами продуктів Rational, а також з консультаційною організацією Rational, щоб забезпечити постійний процес оновлення та вдосконалення, щоб відобразити останній досвід та розвинуті, перевірені, найкращі практики [1].

Раціональний об'єднаний процес підвищує продуктивність команди, надаючи кожному члену команди легкий доступ до бази знань з настановами, шаблонами та інструментами для всіх, хто займаються важливою розробкою. Маючи доступ до однієї бази знань, незалежно від того, чи ви працюєте з вимогами, дизайном, тестуванням, управлінням проектами або керуванням конфігурацією, RUP гарантує, що всі члени команди мають спільну мову, процес та уявлення про те, як розробляти програмне забезпечення. Діяльність Раціонального Єдиного Процесу створює та підтримує моделі замість того, щоб зосередитися на виробництві великої кількості паперових документів. Єдиний процес наголошує на розробці та підтримці моделей - семантично багатих представлень програмної системи, що розробляється.

Раціональний об'єднаний процес – це посібник щодо ефективного використання уніфікованої мови моделювання (UML). UML – це галузева мова, яка дозволяє нам чітко спілкуватися з вимогами, архітектурою та дизайном. Спочатку UML був створений Rational Software, а зараз

підтримується організацією стандартів Object Management Group (OMG). Раціональний єдиний процес підтримується інструментами, які автоматизують великі частини процесу. Вони використовуються для створення та підтримки різних артефактів, зокрема моделей процесу інженерії програмного забезпечення: візуального моделювання, програмування, тестування тощо. Вони безцінні для підтримки всієї бухгалтерії, пов'язаної з управлінням змінами, а також управління конфігурацією, які супроводжує кожну ітерацію. Раціональний об'єднаний процес - це налаштований процес. Уніфікований процес підходить як невеликим розробникам, так і великим організаціям з розвитку. Уніфікований процес заснований на простій і зрозумілій архітектурі процесів, яка забезпечує спільність для всієї групи процесів. В собі містить набір для розробки, що забезпечує підтримку налаштування процесу відповідно до потреб кожної організації [1].

Раціональний об'єднаний процес охоплює багато кращих практик сучасної розробки програмного забезпечення у формі, що підходить для широкого кола проектів та організацій. Розгортання цих кращих практик з використанням Rational Unified Process пропонує командам розробників ряд ключових переваг [1].

Раціональний єдиний процес описує, як ефективно застосовувати перевірені на ринку підходи до розробки програмного забезпечення для команд з розробки програмного забезпечення. Вони називаються «найкращими практиками» не стільки тому, що ви можете точно оцінити їхню цінність, а, скоріше, тому, що вони, як спостерігається, успішно використовуються в промисловості. Раціональний єдиний процес надає кожному члену команди керівництво, шаблони та інструменти для інструментів, необхідні для всієї команди, щоб повністю використати серед інших наступні найкращі практики:

- ітераційно розробляти програмне забезпечення;
- керувати вимогами;

- використовувати архітектури на основі компонентів;
- програмне забезпечення для візуальної моделі;
- перевірка якості програмного забезпечення;
- контроль змін програмного забезпечення.

Ітеративні розробки програмного забезпечення. Враховуючи сучасні вдосконалені програмні системи, неможливо спочатку послідовно визначити всю проблему, розробити все рішення, створити програмне забезпечення і потім протестувати продукт наприкінці. Необхідний ітеративний підхід, який дозволяє розширити розуміння проблеми шляхом послідовних вдосконалень, а також поступово виробити ефективне рішення за допомогою декількох ітерацій. Раціональний об'єднаний процес підтримує ітеративний підхід до розробки, який стосується найвищих ризикових стадій на кожному етапі життєвого циклу, значно зменшуючи профіль ризику проекту. Цей ітеративний підхід допомагає ізолювати ризик за рахунок демонстраційного прогресу, часто виконуваних версій, що дозволяють постійно включати кінцевого користувача та зворотній зв'язок. Оскільки кожна ітерація закінчується виконуваним випуском, команда розробників залишається зосередженою на створенні результатів, а часті перевірки стану допомагають забезпечити виконання проекту за графіком. Ітеративний підхід також полегшує пристосування тактичних змін до вимог, особливостей або розкладу [1].

Керування вимогами. Раціональний єдиний процес описує, як отримати, упорядкувати та документувати необхідні функціональні можливості та обмеження; відстежувати та документувати компроміси та рішення; та легко фіксувати та спілкуватись із вимогами бізнесу. Поняття про використання та сценарії використання, прописані в процесі, виявилися відмінним способом збору функціональних вимог та забезпечення їх відповідності розробці, впровадженню та тестуванню програмного забезпечення, роблячи більш імовірним, що кінцева система задовольняє потреби кінцевого користувача. . Вони забезпечують узгоджені та

відстежувані потоки як через розробку, так і за допомогою поставленої системи.

Використання компонентних архітектур. Процес зосереджується на ранній розробці та базовій основі надійної виконуваної архітектури, перш ніж залучити ресурси для повномасштабної розробки. В ньому описано, як створити гнучку архітектуру, яка можна змінювати, зрозуміла, інтуїтивно та сприяє більш ефективному використанню програмного забезпечення. Раціональний єдиний процес підтримує розробка програмного забезпечення на основі компонентів. Компоненти - це нетривіальні модулі, підсистеми, які виконують чітку функцію. Раціональний єдиний процес забезпечує системний підхід до визначення архітектури з використанням нових та існуючих компонентів. Вони зібрані в чітко окресленій архітектурі або в спеціальній інфраструктурі, такі як Інтернет, CORBA та COM, для якої створюється галузь багаторазових компонентів [1].

Програмне забезпечення для візуальної моделі. Процес показує, як візуально моделювати програмне забезпечення для відображення структури та поведінки архітектур та компонентів. Це дозволяє приховати деталі та записати код за допомогою «графічних будівельних блоків». Візуальні абстракції допомагають повідомляти різні аспекти вашого програмного забезпечення; подивіться, як елементи системи поєднуються між собою; переконатись, що будівельні блоки відповідають вашому коду; підтримувати узгодженість між дизайном та його реалізацією; та сприяти однозначному спілкуванню. Індустріальний стандарт уніфікованої мови моделювання (UML), створений Rational Software, є основою для успішного візуального моделювання [1].

Перевірка якості програмного забезпечення. Низька продуктивність програми та низька надійність є загальними факторами, які різко гальмують прийнятність сьогоднішніх програмних процесів. Отже, якість має бути переглянута щодо вимог, що ґрунтуються на надійності, функціональності, продуктивності програми та продуктивності системи. Раціональний єдиний

процес допомагає у плануванні, проектуванні, реалізації, виконанні та оцінці цих типів тестів. Оцінка якості вбудовується в процес, у всіх заходах, в яких беруть участь усі учасники, використовуючи об'єктивні вимірювання та критерії, і не трактується як думка або окрема діяльність, яку виконує окрема група [1].

Зміни в управлінні програмним забезпеченням. Можливість керувати змінами, що кожна зміна є прийнятною, а можливість відстежувати зміни є важливою умовою, в якій зміни неминучі. Процес описує, як контролювати, відстежувати зміни, щоб забезпечити успішний ітеративний розвиток. Він також вказує вам, як створити захищені робочі простори для кожного розробника, забезпечуючи ізоляцію від змін, внесених в інші робочі простори, і контролюючи зміни всіх артефактів програмного забезпечення (наприклад, моделей, коду, документів тощо). І це об'єднує команду, щоб працювати як єдине ціле, описуючи, як автоматизувати інтеграцію та побудувати управління.

### 1.2.2 Моделювання архітектури системи

Для більш швидкої розробки та на вищому рівні в сучасній IT сфері використовується так званий каркас (Framework).

Програмне каркас (Framework) - це конкретна або концептуальна платформа, де загальний код із загальною функціональністю може бути вибірково спеціалізований. Framework мають форму бібліотек, де чітко визначений інтерфейс програми (API) де можливо повторно використовувати в будь-якому місці в рамках програмного забезпечення, що розробляється.

Деякі функції відрізняють framework від інших форм бібліотеки, включаючи такі:

- поведінка за замовчуванням: перед налаштуванням каркасу поводяться таким чином, що є специфічним для дії користувача.

- інверсія управління: на відміну від інших бібліотек, глобальний потік управління в каркасу використовується не ним самим, а тим хто використовує.
- розширюваність: користувач може розширити каркас, вибірково замінивши код за замовчуванням на код власний;
- код каркасу, який не змінюється: користувач може розширити каркас, але не змінювати код.

Метою програмного каркасу є спрощення середовища розробки, що дозволяє розробникам приділяти свої зусилля проектним вимогам, а не займатися життєвими, повторюваними функціями та бібліотеками. Наприклад, замість того, щоб створити додаток з нуля, розробник, використовуючи підготовлений фреймворк, може сконцентруватися на додаванні зручних для користування кнопок і меню або інтеграції з іншими функціями.

При розробці всіх компонентів програмної системи, а саме клієнтської та серверної частин є доцільним використати відповідні програмні каркаси для забезпечення економії часу та зосередитись більше на безпосередній розробці нестандартного функціоналу. Для серверної частини програмної системи одним із хороших варіантів є застосування .NET Core платформи для розробки та використання каркасу ASP.NET Web API 2.0.

.NET Core – це універсальна платформа розробки з відкритим кодом, яку підтримує корпорація Майкрософт і співтовариство. Вона є кроссплатформенной (підтримує Windows, macOS і Linux) і може використовуватися для створення додатків для пристроїв, хмари і Інтернету речей [3].

.NET Core володіє наступними характеристиками:

- кросплатформеність. Підтримка операційних систем Windows, macOS і Linux.
- узгодженість між архітектурою. Однакове виконання коду в різних архітектурах, включаючи x64, x86 і ARM.

- програми командного рядка. Зручні інструменти для локальної розробки і сценаріїв безперервної інтеграції.
- гнучка розробка. Може включатися в додаток або встановлюватися паралельно (на рівні користувача або системи). Можливість використання з контейнерами Docker.
- сумісність. Платформа .NET Core сумісна з .NET Framework, Xamarin і Mono завдяки .NET Standard.
- відкритий код. Платформа .NET Core має відкритий код і поширюється за ліцензіями MIT і Apache 2. .NET Core є проектом .NET Foundation.
- підтримка від Майкрософт. Корпорація Майкрософт надає підтримку .NET Core.

.NET Core дозволяє створювати додатки і бібліотеки на мовах C #, Visual Basic і F #. Ці мови можна використовувати в вашому улюбленому текстовому редакторі або інтегрованому середовищі розробки (IDE), включаючи такі [3].

- Visual Studio
- Visual Studio Code
- Sublime Text
- Vim

Інтеграція можлива багато в чому завдяки учасникам проектів OmniSharp і Ionide.

.NET Core надає API-інтерфейси для безлічі сценаріїв, в деяких з яких використовуються [3]:

- примітивні типи, такі як bool і int;
- колекції, такі як System.Collections.Generic.List <T> і System.Collections.Generic.Dictionary <TKey, TValue>;
- службові типи, такі як System.Net.Http.HttpClient і System.IO.FileStream;
- типи даних, такі як System.Data.DataSet і DbSet;

- високопродуктивні типи, такі як System.Numerics.Vector і Pipelines.

Завдяки підтримці специфікації .NET Standard платформа .NET Core сумісна з .NET Framework і API-інтерфейсами Mono [3].

Одним із вагомих причин вибору каркасу ASP.NET Core є включення в собі зручного способу управління залежностями всередині системи тобто шаблон проектування Dependency Injection.

Dependency Injection (DI) – шаблон проекетування, що використовується для впровадження IoC. Це дозволяє створювати залежні об'єкти поза класом і надає ці об'єкти класу різними способами. Використовуючи DI, ми переміщуємо створення та зв’язування залежних об'єктів поза класом, який залежить від них [5].

Схема введення залежностей включає три типи класів:

- клас клієнта: Клас клієнта (залежний клас) - це клас, який залежить від класу обслуговування
- клас обслуговування: Клас обслуговування (залежність) - це клас, який надає обслуговування клієнтському класу.
- клас форсунки: Клас інжекторів вводить об'єкт класу обслуговування в клієнтський клас.

Наступний рисунок ілюструє взаємозв'язок між цими класами:

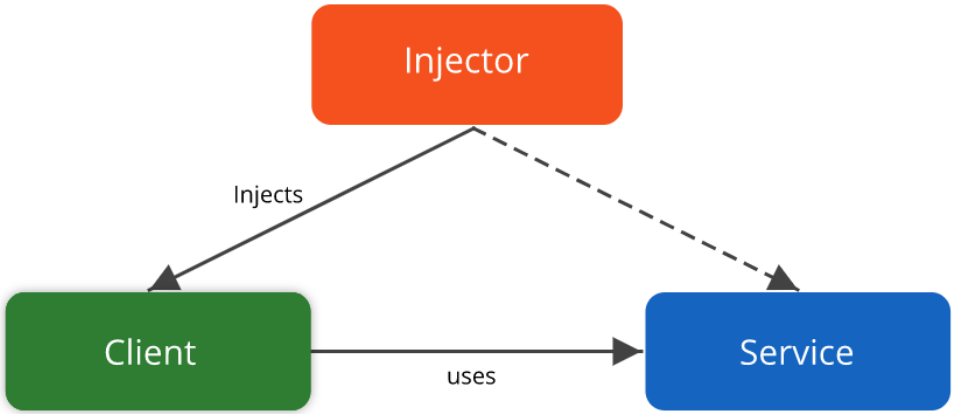


Рисунок 1.5 – Схема взаємодії між класами

Як видно з схеми, клас інжектора створює об'єкт класу обслуговування та вводить цей об'єкт клієнтському об'єкту. Таким чином, шаблон DI відокремлює відповідальність за створення об'єкта класу обслуговування поза класом клієнта. Клас інжекторів вводить клієнту послугу (залежність). Клас інжекторів вводить залежності трьома способами: через конструктор, властивість або через метод [5].

Інжекція в конструктор. В інжектор конструктора інжектор подає послугу (залежність) через конструктор класу клієнтів [5].

Вприскування властивостей. В ін'єкції властивостей (він же інжектор Setter) інжектор подає залежність через публічну властивість класу клієнтів [5].

Метод ін'єкції. У цьому типі ін'єкції клас клієнта реалізує інтерфейс, який оголошує метод (и) для забезпечення залежності, а інжектор використовує цей інтерфейс для подачі залежності до клієнтського класу.

ASP.NET Core містить можливість використання шару проміжної обробки між HTTP запитом клієнтської частини та точкою обробки в контролері на стороні серверу. В обробниках що включаються в цей шар можна додавати проміжні перевірки вхідного запиту для забезпечення несанкціонованого доступу, що може призвести до пошкодження роботи системи. В розробці програмного забезпечення так звані фільтри називають middleware.

Middleware - це програмне забезпечення, яке лежить між клієнтською та серверною частинами системи. По суті, middleware функціонує як прихований шар пропуску. Використання проміжного програмного забезпечення дозволяє користувачам виконувати такі запити, як подання форм у веб-браузері або дозволяє веб-серверу повертати динамічну мережу сторінки на основі профілю користувача. Загальна схема роботи middleware наведено на рисунку 1.6. [4]

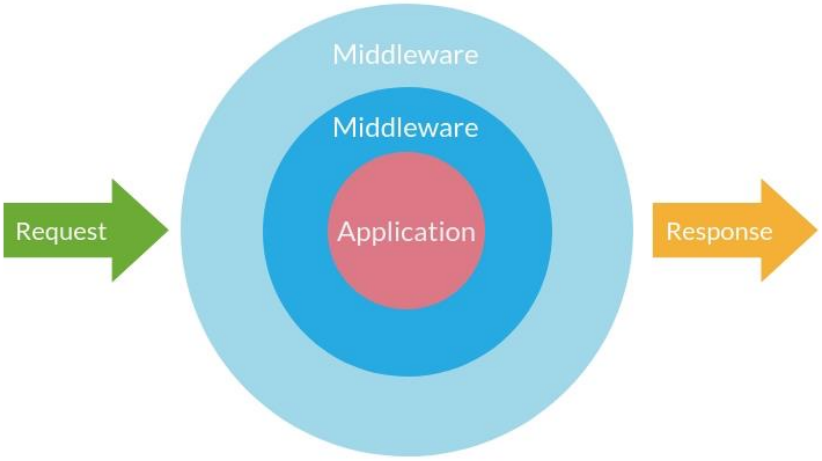


Рисунок 1.5 – Схема роботи проміжного програмного забезпечення

Загальні приклади проміжного програмного забезпечення включають:

- проміжне програмне забезпечення баз даних;
- проміжне програмне забезпечення сервера додатків;
- проміжне програмне забезпечення, орієнтоване на повідомлення,;
- веб-програмне забезпечення та моніторинг обробки транзакцій.

Кожна програма, як правило, надає послуги обміну повідомленнями, щоб різні додатки могли спілкуватися за допомогою фреймворків для обміну повідомленнями, таких як простий протокол доступу до об’єктів (SOAP), веб-служби, передача стану репрезентації (REST) та позначення об'єктів JavaScript (JSON). Незважаючи на те, що все проміжне програмне забезпечення виконує функції зв'язку, тип компанії, який вирішить використовувати, залежатиме від того, яка послуга використовується та який тип інформації потрібно передавати. Серед такої інформації можна віднести:

- автентифікацію безпеки;
- управління транзакціями;
- черги повідомлень;
- сервери додатків;
- веб-сервери та каталоги.

Проміжне програмне забезпечення також може використовуватися для розподіленої обробки з діями, що відбуваються в режимі реального часу, а не надсиланням даних туди і назад.

В системі «Соціальна мережа для бібліотек» буде реалізовуватись один обробник проміжного програмного забезпечення, який буде виконувати перевірку на наявність в запиті JSON Web Token для підвищення безпеки.

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами як об'єкт JSON. Цю інформацію можна перевірити та довірити, оскільки вона підписана цифровим шляхом. JWT можуть бути підписані за допомогою секретного (за допомогою алгоритму HMAC) або пари відкритих / приватних ключів за допомогою RSA або ECDSA. Загальна схема створення JWT наведено на рисунку 1.6. [2]

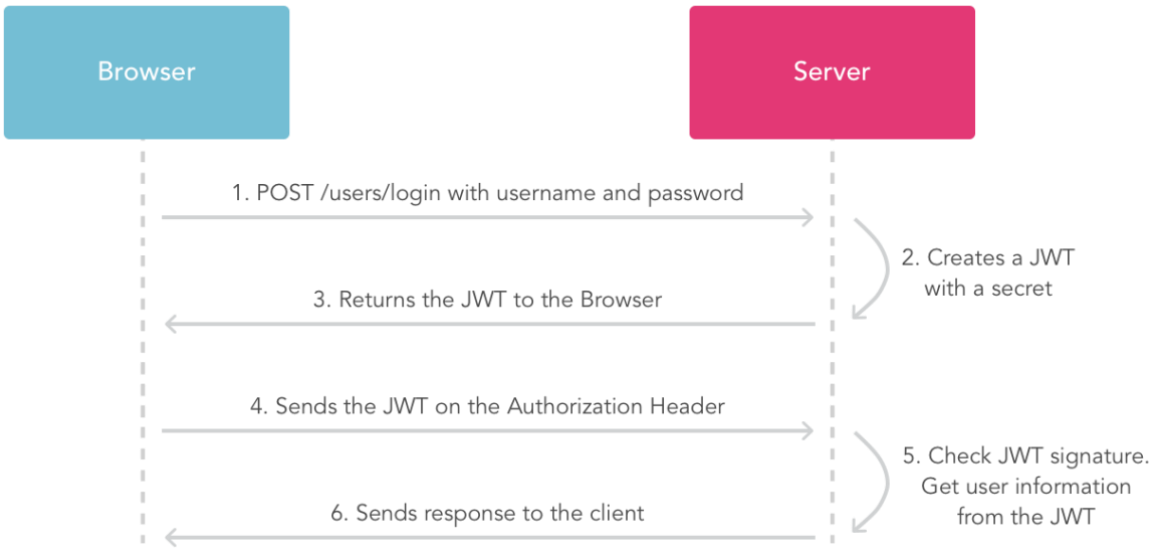


Рисунок 1.6 – Схема створення JSON Web Token

JWT корисно використовувати в наступних випадках:

- Авторизація: Це найпоширеніший сценарій використання JWT. Після входу користувача кожен наступний запит буде включати JWT, дозволяючи користувачеві отримувати доступ до маршрутів, служб та ресурсів, дозволених за допомогою цього маркера. Single

Sign On - функція, яка сьогодні широко використовується JWT, через її невеликі накладні витрати та її можливість легко використовуватись у різних областях.

- Обмін інформацією: веб-токени JSON - це хороший спосіб надійної передачі інформації між сторонами. Оскільки JWT можуть бути підписані (наприклад, за допомогою пар відкритих / приватних ключів), можна бути впевнені, що відправники такі, про які вони кажуть. Крім того, оскільки підпис обчислюється за допомогою заголовка та корисного навантаження, ви також можете перевірити, чи вміст не був підроблений.

У своєму компактному вигляді веб-маркери JSON складаються з трьох частин, розділених крапками (.), а саме:

- Заголовок. Заголовок, як правило, складається з двох частин: типу маркера, який є JWT, та використовуваного алгоритму підпису, такого як HMAC SHA256 або RSA;
- Корисне навантаження. Друга частина маркера - це корисне навантаження, яке містить претензії. Претензії - це заяви про організацію (зазвичай, користувача) та додаткові дані. Існує три типи вимог: іменні, державні та приватні претензії;
- Підпис. Для створення частини підпису вам потрібно взяти кодований заголовок, кодований корисний вантаж, секрет, алгоритм, вказаний у заголовку, і підписати це.

Об'єктно-реляційне відображення (ORM) - це метод програмування, в якому дескриптор метаданих використовується для підключення об'єктного коду до реляційної бази даних. Об'єктний код записується на об'єктно-орієнтованих програмуваннях (ООР), таких як Java або C#. ORM перетворює дані між системами типів, які не можуть співіснувати в реляційних базах даних та мовах ООР.

ORM вирішує невідповідність об'єктного коду та реляційної бази даних трьома підходами: знизу вгору, згори вниз та зустріч посередині. Кожен

підхід має свою частку переваг та недоліків. Вибираючи найкраще програмне рішення, розробники повинні повністю розуміти середовище та вимоги до дизайну.

Окрім техніки доступу до даних, до переваг ORM також належать:

- спрощена розробка, оскільки вона автоматизує перетворення об'єкта на клас мови програмування, що призводить до зниження витрат на розробку та обслуговування;
- менше коду в порівнянні з вбудованими SQL та рукописними збереженими процедурами;
- прозоре кешування об'єктів у рівні програми, покращуючи продуктивність системи;
- оптимізоване рішення, що робить додаток швидшим та легшим у обслуговуванні.

Поява ORM у розробці декількох додатків породила розбіжність серед експертів. Основні занепокоєння полягають у тому, що ORM не працює належним чином і що збережені процедури можуть бути кращим рішенням. Крім того, залежність від ORM може призвести до погано розроблених баз даних за певних обставин.

В .NET Core реалізація ORM відбувається за допомогою Entity Framework Core, який включає в собі зручний спосіб підтримки цілісності зв’язку програми з базою даних на основі міграцій. Такий підхід допоможе при зміні структури моделі бази даних виконувати оновлення її фізичної сторони.

Entity Framework Core – це нова версія Entity Framework після EF 6.x. Це відкритий, легкий, розширюваний і крос-платформний варіант технології доступу до даних Entity Framework [7].

Entity Framework – це структура об'єктно-реляційного картографування (O/RM). Це вдосконалення для ADO.NET, що надає розробникам автоматизований механізм доступу та зберігання даних у базі даних [7].

EF Core призначений для використання з програмами .NET Core. Однак він також може бути використаний у стандартних додатках на основі .NET 4.5+. [7]

EF Core підтримує два підходи до розробки:

- Code-First;
- Database-First.

EF Core в основному орієнтований на підхід, Code-First, і надає незначну підтримку підходу, який базується на Database-First, оскільки візуальний дизайнер або майстер для моделі БД не підтримуються для EF Core 2.0.

У підході до Code-First API EF Core створює базу даних та таблиці за допомогою міграції на основі конвенцій та конфігурацій, наданих у ваших класах домену. Цей підхід корисний у дизайні, керованому доменом (DDD).

У Code-First EF Core API створює доменні та контекстні класи на основі наявної бази даних за допомогою команд EF Core. У EF Core обмежена підтримка, оскільки він не підтримує візуального дизайнера чи майстра.

Так як система соціальної мережі для бібліотек розробляється з нуля тоді доцільніше використати підхід Code-First, за принципом якого свотрюється спочатку класи з моделями таблиць бази даних після чого за допомогою міграції відбувається перенесення у фізичну модель бази даних. Процес виконання підходу Code-First зображено на рисунку 1.7.

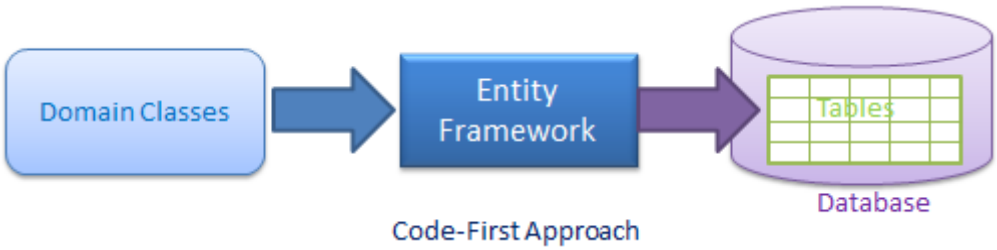


Рисунок 1.7 – Схема Code-First підходу

На стороні клієнта можна використати такий стек технологій: Angular 8, Bootstrap 4, SASS. Також використання зовнішних API, а саме Google книги API із утилтою для перегляду книг та картами MapBox.

Angular – один із найпопулярніших каркасів та платформ JavaScript для розробки мобільних та настільних веб-додатків для мобільних пристроїв та настільних ПК або додатків для однієї сторінки (SPA) на стороні клієнта.

Angular 8 – це сценарій JavaScript з відкритим кодом, що працює на основі побудови односторінкових програм. Він написаний на TypeScript і сумісний з JavaScript. Angular 8 використовується для створення динамічних веб-додатків. Даний каркас схожий на свої попередні версії, за винятком деяких обширних особливостей. Переваги над попередніми версіями є головним чином зосереджений на диференційованих завантаженнях, динамічному імпорті для ледачих маршрутів, веб-службовцях та Angular Ivy в якості допоміжної підтримки. Він також підтримує TypeScript 3.4.

Динамічний веб-додаток - це просто динамічний веб-сайт. тобто www.gmail.com, www.facebook.com, www.yahoo.com тощо, який має тенденцію змінювати дані / інформацію стосовно трьох параметрів:

- час від часу (наприклад, веб-програми для оновлення новин);
- місцеположення (наприклад, веб-програми звіт про погоду);
- користувач-користувач (наприклад, Gmail, програми типу Facebook).

Angular реалізовує паттерн розробки MVVM та модульну структуру додатку.

Model-View-ViewModel (MVVM) - це модель дизайну програмного забезпечення, яка побудована для розділення логіки програми та управління інтерфейсом користувача. MVVM також відомий як в'яжуча модель-модель і була створена архітекторами Microsoft Кен Купер і Джоном Госсманом.

Як і багато інших моделей дизайну, MVVM допомагає впорядковувати коди та розбивати програми на модулі, щоб зробити розробку готовою до оновлення та повторного використання коду, простішою та швидшою.

Шаблон часто використовується в програмному забезпеченні для презентації Windows та веб-графіки.

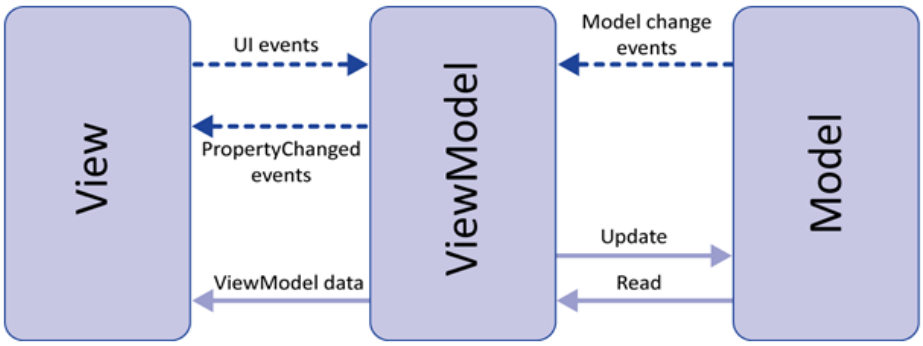


Рисунок 1.8 – Шаблон MVVM

Шаблон MVVM використовується у Фонді презентацій Windows (WPF), який працює на .NET Microsoft. Мультимедійний модуль Silverlight, еквівалентний в Інтернеті Microsoft WPF, також використовує MVVM.

Поділ коду в MVVM поділяється на View, ViewModel та Model:

- перегляд - це сукупність видимих елементів, яка також отримує введення користувача. Сюди входять користувацькі інтерфейси (інтерфейс користувача), анімації та текст. Вміст представлення не взаємодіє безпосередньо з тим, щоб змінити представлене;
- ViewModel розташований між шарами View і Model. Тут розміщуються елементи управління взаємодією з View, в той час як прив'язка використовується для з'єднання елементів інтерфейсу в View з елементами управління в ViewModel;
- модель зберігає логіку програми, яку отримує ViewModel після власного отримання вводу від користувача через View.

У Angular в якості View виступає HTML та SASS, які забезпечують моделювання інтерфейсу, Model реалізовують сервісі в яких відбувається виконання запитів до серверу та обробка результатів. ViewModel в свою чергу виступає мостом між View та Model описуючи логіку поведінки компонентів інтерфейсу із обробленими даними із сервісів.

Для більш ефективного та легко читабельного коду графічних стилей використовується розширення каскадних таблиць стилей (CSS) – SASS.

Для побудови в графічного інтерфейсу в також використано бібліотеку Bootstrap версії 4.

Bootstrap – це бібліотека з великою кількістю компонентів інтерфейсу користувача для веб-сторінок. Основними перевагами даної бібліотеки є простота використання та великого спектру готових компонентів, що забезпечить швидку розробку та дозволить зосередитись на більш глобальних питання стосовно вигляду додатку в цілому.

Для перегляду книг використано Google книги API, що надає зможу не використовувати довгі форми заповнення даних при додаванні та перегляду книг.

Також в системі передбачена перегляд локацій на карті. Для реалізації цієї задачі використовується безкоштовні карти від MapBox.

Mapbox - це платформа даних про місцезнаходження для мобільних та веб-додатків. Mapbox надає будівельні блоки, щоб додати функції розташування, такі як карти, пошук та навігацію.

### 1.2.3 Побудова схеми бази даних

База даних – це сукупність інформації, яка організована так, що до неї можна легко отримати доступ, керувати і оновлювати. Комп'ютерні бази даних зазвичай містять агрегацію записів даних або файлів, що містять конкретну інформацію про об’єкт.

Бази даних розвиваються з моменту їх створення в 1960-х роках, починаючи з ієрархічних та мережевих баз даних, до 1980-х років з об'єктно-орієнтованими базами даних, а сьогодні – з базами даних SQL і NoSQL та хмарними базами даних.

З одного погляду, бази даних можна класифікувати за типом вмісту: бібліографічний, повний текст, числовий та зображення. При обчислювальній роботі бази даних іноді класифікуються відповідно до їх

організаційного підходу. Існує багато різних типів баз даних, починаючи від найбільш поширеного підходу, реляційної бази даних, до розподіленої бази даних, хмарної бази даних, графічної бази даних або бази даних NoSQL.

Реляційна база даних, винайдена Е. Ф. Коддом в IBM в 1970 році, - це таблична база даних, в якій дані визначаються таким чином, щоб можна було реорганізувати та отримати доступ до них різними способами.

Реляційні бази даних складаються з набору таблиць з даними, що вписуються у заздалегідь задану категорію. Кожна таблиця містить щонайменше одну категорію даних у стовпці, і кожен рядок має певний екземпляр даних для категорій, визначених у стовпцях.

При проектуванні бази даних виникає необхідність в її нормалізації та забезпечення найоптимальнішої структури без дублювання даних.

Нормалізація - це процес організації даних у базі даних. Цей процес включає створення таблиць та встановлення зв'язків між цими таблицями відповідно до правил, призначених як для захисту даних, так і для того, щоб зробити базу даних більш гнучкою, усуваючи надмірні та непослідовні залежності.

Надлишки даних витрачають дисковий простір і створюють проблеми з технічним обслуговуванням. Якщо дані, які існують у більш ніж одному місці, повинні бути змінені, дані повинні бути змінені точно так само у всіх місцях.

Існує кілька правил нормалізації бази даних. Кожне правило називається "нормальною формою".

Перша нормальна форма (1NF) встановлює основні правила нормалізації бази даних і стосується єдиної таблиці в системі реляційних баз даних. Нормалізація проводиться за трьома основними етапами, кожен будуючи останній. Перша з них - перша нормальна форма.

Перша нормальна форма зазначає, що:

- кожен стовпець таблиці повинен бути унікальним;

- для кожного набору пов’язаних даних повинні бути створені окремі таблиці;
- кожна таблиця повинна бути ідентифікована унікальним стовпчиком або об'єднаними стовпцями, які називаються первинним ключем;
- жоден рядок не може бути продубльований;
- жоден стовпчик не може бути продубльований;
- жодні перетини рядків / стовпців не містять нульового значення;
- жоден перетин рядка / стовпця не містить багатозначних полів.

Друга нормальна форма (2NF) - другий крок у нормалізації бази даних. 2NF будується на першій нормальній формі (1NF) і заснована на концепції повної функціональної залежності. Друга нормальна форма стосується відносин із складеними ключами, тобто відносин з первинним ключем, що складається з двох або більше атрибутів.

Третя нормальна форма (3NF) - це третій крок у нормалізації бази даних, і він ґрунтується на першій та другій нормальних формах, 1NF та 2NF.

3NF зазначає, що всі посилання стовпців що посилаються на дані, які не залежать від первинного ключа, повинні бути видалені. Іншим способом цього є те, що для посилання на іншу таблицю слід використовувати лише стовпці із зовнішнім ключем.

Приведення бази даних до наступних нормальних форма є недоцільним так як призведе до збільшення зв’язаності таблиць та подальшого управління даними.

Одним з важливих аспектів є визначення моделі бази даних та вибір найбільш оптимального способу представлення структури.

Модель бази даних – це спосіб визначення структури або логічного проектування бази даних. Він розповідає про спосіб зберігання, доступу та оновлення даних у СУБД (система управління базами даних). Існує багато моделей для представлення даних. Деякі з них:

- Реляційна модель;

- Ієрархічна модель;
- Мережева модель;
- Модель особи-відносини.

При виборі моделі необхідно визначити найбільш підходящу для поставлених задач та системи в цілому. Найбільш популярними моделями виявились ієрархічна та мережева, тому варто виконати аналіз та виявити найбільш підходящу. Переваги мережевої моделі над ієрархічною наведено у таблиці 1.1.

Таблиця 1.1 – Переваги мережевої моделі над ієрархічною

| Мережева модель | Ієрархічна модель |
|---|---|
| Багато до багатьох | Один до багатьох |
| Доступ до них легко через зв’язок між інформацією | Важко орієнтуватися через суворий зв’язок між власником та членом |
| Велика гнучкість між інформаційними файлами, оскільки безліч зв'язків між сутностями | Менша гнучкість у зборі інформації через ієрархічне положення сутностей |

Модель мережевої бази даних – це модель бази даних, яка дозволяє зв'язати кілька записів з одним файлом власника. Модель може розглядатися як перевернуте дерево, де на гілках є інформація про членів, пов'язана з власником, що є дном дерева. Багаторазові зв’язки, які дають цю інформацію, дозволяють моделі мережевої бази даних бути дуже гнучкою. Крім того, взаємозв'язок, яку має інформація в моделі мережевої бази даних, визначається як зв'язок "багато до багатьох", оскільки один власник-файл може бути пов'язаний з багатьма файлами-членами і навпаки. Загальна схема мережевої моделі наведено на рисунку 1.9.

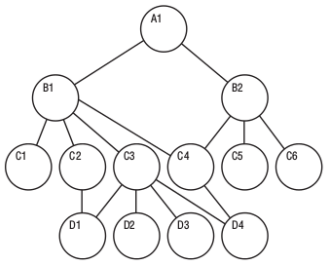


Рисунок 1.9 – Схема мережевої моделі бази даних

Виходячи з вищесказаного в розробці соціальної мережі побудова бази даних буде базуватись на мережеві моделі. Це зумовлено тим що всі дані будуть взаємопов’язані між собою та користувачем.

Визначивши всі архітектурні вимоги щодо моделі бази даних можна перейти до безпосередньої розробки схеми бази даних, або іншими словами ER – діаграми. Для цього скористаємось програмою від компанії Oracle – MySQL Workbench 8.0 CE.

MySQL Workbench – це графічний інструмент для роботи з серверами та базами даних MySQL. MySQL Workbench повністю підтримує версії сервера MySQL 5.6 та новіших версій. Він також сумісний із старими версіями сервера 5.x MySQL, за винятком певних ситуацій (наприклад, відображення списку процесів) через змінені системні таблиці. Він не підтримує версії сервера MySQL 4.x. [10]

Функціонал MySQL Workbench охоплює п’ять основних тем:

- Розробка SQL: дозволяє створювати та керувати з'єднаннями з серверами баз даних. Поряд з можливістю налаштування параметрів з'єднання, MySQL Workbench надає можливість виконувати запити SQL на підключеннях до бази даних за допомогою вбудованого редактора SQL;
- Моделювання даних (дизайн): дозволяє створювати моделі схеми вашої бази даних графічно, реверсувати та переправляти між схемою та реальною базою даних та редагувати всі аспекти вашої бази даних за допомогою всебічного редактора таблиць. Редактор таблиць надає прості у використанні засоби для редагування

таблиць, стовпців, покажчиків, тригерів, розділення, параметрів, вкладок та привілеїв, рутинних програм та представлень;

- Адміністрування сервера: дає змогу керувати екземплярами сервера MySQL шляхом адміністрування користувачів, виконання резервного копіювання та відновлення, перевірки даних аудиту, перегляду стану баз даних та контролю за роботою сервера MySQL;
- Міграція даних: Дозволяє переходити з Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostreSQL та інших RDBMS таблиць, об'єктів і даних до MySQL. Міграція також підтримує перехід від попередніх версій MySQL до останніх версій;
- Підтримка MySQL Enterprise: Підтримка таких корпоративних продуктів, як MySQL Enterprise Backup, MySQL Firewall та MySQL Audit.

Для створення нової моделі бази даних із можливістю проектування її схеми за допомогою будівельних блоків ER – діаграми необхідно вибрати пункт головного меню «Файл » далі «Нова модель » (рис. 1.10).

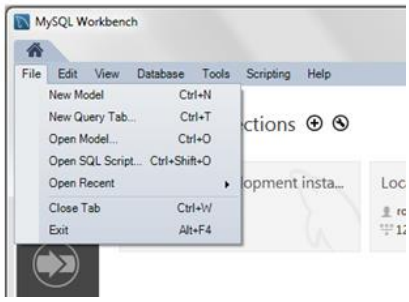


Рисунок 1.10 – Створення нової схеми бази даних

Після чого ввести ім’я бази даних що розробляється в даному випадку це LibraryNetwork та натиснути двічі на пункті вище «ERR Diagram », що виконає створення нової ER – діаграми (рис. 1.11).

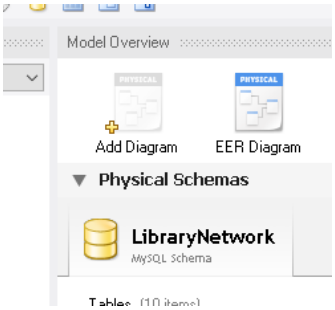


Рисунок 1.11 – Додавання ER – діаграми

Після виконання вищезазначених дій появиться полотно на якому можна буде розміщати елементи схеми бази даних.

Загальна схема побудови бази даних відбувається наступним чином: описуються сутності у вигляді блоків в яких вказується назва таблиці та поля, описуються зв’язки між сутностями.

В базі даних передбачено наступні сутності:

- Користувач (User);
- Книги користувача (UserBook);
- Книга (Book);
- Кгода (Agreement);
- Підписники (SubscriberSet);
- Пост (Post);
- Коментар (Comment)
- Відмітка (Mark);
- Подія (Event);
- Зображення (Image)

Сутність «User» описує інформацію про користувача. Дана сутність є нашої схеми бази даних з якої будуть йти безліч зв’язків до інших сутностей. Це зумовлено тим що в соціальних мережах користувач є ключовим суб’єктом. Іноформаці  буде зберігатись за наступними полями:

- Первинний ключ (Id);
- Логін (UserName);
- Пароль (Password);

- Чи активний (IsActive);
- Чи про аккаунт (IsMaster);
- Електронна пошта (Email);
- Номер телефону (PhoneNumber);
- Локація (Location);
- Додаткова інформація (AdditionalInfo)

Поля UserName та Password необхідні для авторизації в системі. Поле Location використовується для визначення приблизного місця знаходження бібліотеки користувача. Всі інші поля необхідні для майбутньої розширюваності системи. Сутності які виходять із кореня тобто «User» можна умовно розділити на три групи. Перша група відповідатиме за роботу із книгами, угодами, друга група за організацію стрічки новин та третя додаткові таблиці, а саме збереження інформації про завантажені зображення та посилання на підписників. Опис сутності «User» зображено на рисунку 1.12.

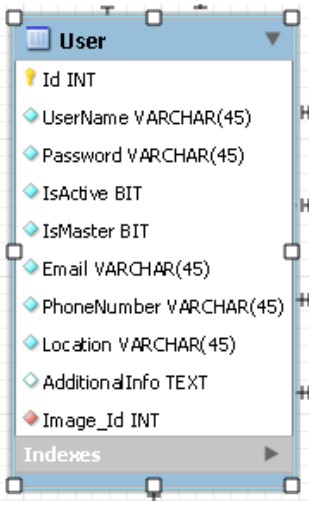


Рисунок 1.12 – Опис сутності «User»

Однією із серйозних проблем при розробці схеми бази даних стосовно даної тематики є можливість дублювання інформації та початкова ініціалізація даних про книги. Частину стосовно наповнення вирішує Google книги API яка містить велику кількість інформації про книги. Це забезпечить

легке та поступове наповнення бази даних лише ключовими полями необхідними при роботі із сутністю «Book» яка міститиме інформацію про книги а саме наступні поля:

- Первинний ключ (Id);
- Назва книги (Name);
- Жанр (Ganre);
- Автор (Author);
- Базове зображення (GoogleBookId);

Інша проблему стосовно дублювання можна вирішити за допомогою створення проміжної таблиці між сутностями «User» та «Book» яка міститиме комбінації ключів. Дану сутність назвемо «UserBooks» яка забеспечуватиме зв'язок під назвою багато до багатьох. Дана сутність також буде мати не тісну прив’язку із іншою сутністю під назвою «Agreement» яка відповідатиме за збереження інформації про угоди стосовно книги яка буде взята в оренду та міститиме наступні поля:

- Первинний ключ (Id);
- Дата та час видачі (IssuedOn);
- На скільки видано днів (Days);
- Статус (Status).
- Зовнішний ключ (User_Id)
- Зовнішний ключ (UserBook_Id)

Опис та побудова зв’язків між даними сутностями наведено на рисунку 1.13.

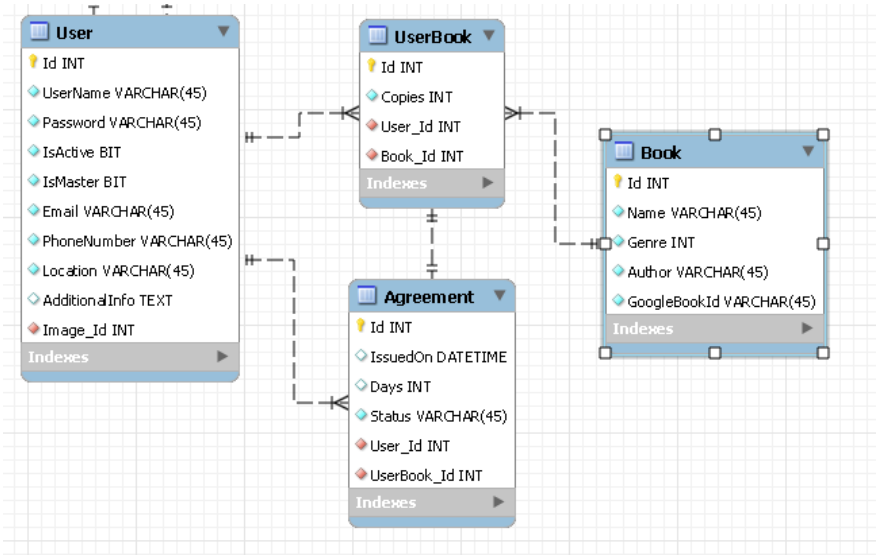


Рисунок 1.13 – Опис сутностей «UserBooks», «Agreement», «Book»

Побудувавши будівельний блок із сутностей першої групи можна переходити до написання наступної групи сутносте а саме: «Post», «Event», «Mark», «Comment».

Сутність «Post» є серцевиною даної групи забезпечуючи збереженням інформації про публікації які будуть формувати стрічку новин. Дана сутність містить наступні поля:

- Первинний ключ (Id);
- Заголовок (Title);
- Опис (Description)
- Дата створення (CreatedOn);
- Зовнішний ключ (User_Id);
- Зовнішний ключ (Image_Id).

Сутність «Event» є розширенням сутності «Post» із зв’язком один до одного та забезпечить інформацією про події а саме за допомогою наступних полів:

- Первинний ключ (Id);
- Адреса проведення події (Address)
- Чи активна подія (IsActive);
- Зовнішний ключ (Post_Id).

Також для реагування на публікації користувачів необхідно додати ще дві сутності, а саме «Mark» та «Comment». Перша сутність виконуватиме маркування за типом подобається, відвідання події, цікавість до події. Дане поле міститиме ціле число яке в системі буде представлене як Enum. Сутність «Comment» міститиме лише поле для тексту та часу створення коментара. Відношення із сутністю «Post» та «User»  буде один до багатьох. Дана група сутносте та зв’язки між ними наведене на рисунку 1.14.

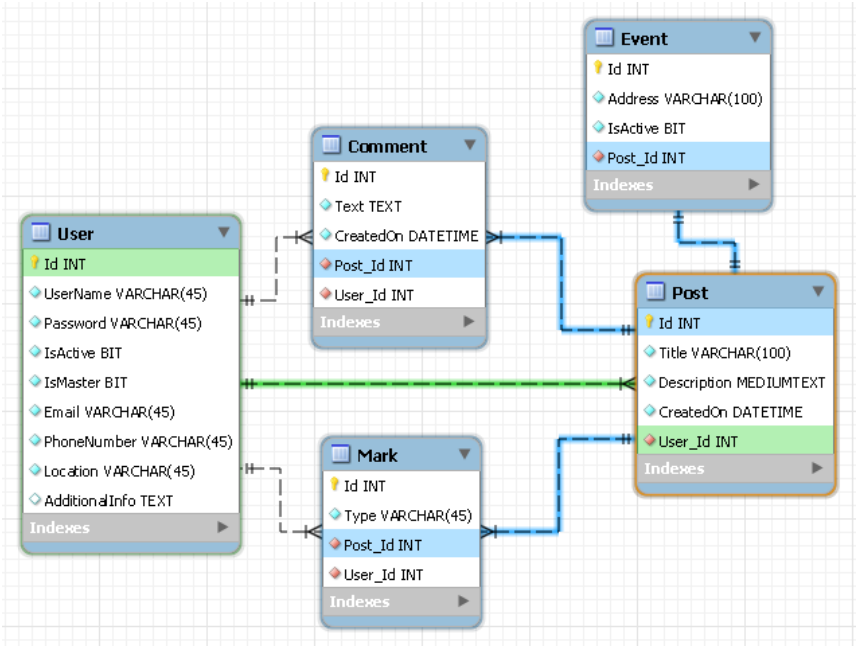


Рисунок 1.14 – Опис групи сутностей що відповідає за стрічку новин

Остання група так звана «обслуговуюча» яка складається з сутностей «Image» та «SubscriberSet». У сутності «Image » зберігатиметься інформація про картинки які будуть завантажуватися в якості фотографій профілю, як колажі до постів. «SubscriberSet» міститиме список підписників та при реверсії ключів при вибірці можна отримати комбінації також і для підписок. Опис даної групи сутностей наведено на рисунку 1.15.

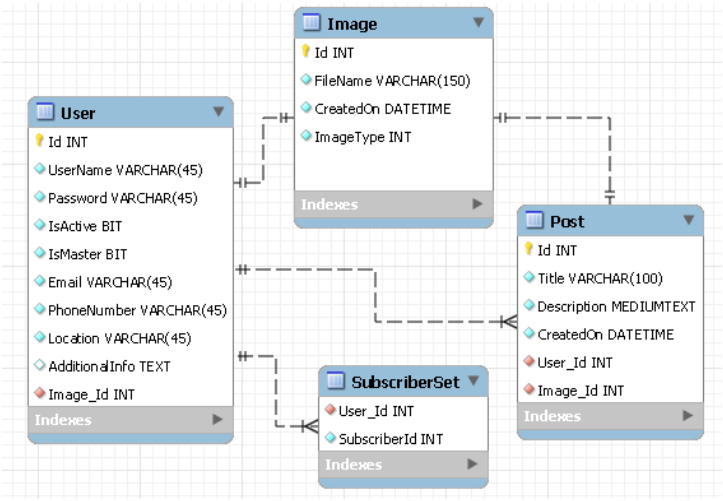


Рисунок 1.15 – Опис групи «обслуговуючих» сутностей

Кінцева ER – діаграма бази даних наведено у додатку Д.

## 1.3 Конструювання програмної системи

### 1.3.1 Вибір мови та середовища розробки

В якості мови програмування для серверної частини було вибрано C# та каркас ASP.NET Web API на платформі .NET Core 2 версії.

C# – об'єктно-орієнтована мова програмування від Microsoft, яка має на меті поєднати обчислювальну потужність C++ з простотою програмування Visual Basic. C # заснований на C ++ і містить функції, подібні до функцій Java [9].

C # призначений для роботи з платформою .Net Microsoft. Метою Microsoft є сприяння обміну інформацією та послугами через Інтернет та надання можливості розробникам створювати високо портативні програми. C # спрощує програмування за допомогою використання розширюваної мови розмітки (XML) та простого протоколу доступу до об'єктів (SOAP), які дозволяють отримати доступ до об'єкта чи методу програмування, не вимагаючи від програміста запису додаткового коду для кожного кроку. Оскільки програмісти можуть базуватися на існуючому коді, а не повторно його дублювати, очікується, що C# зможе швидше і дешевше вивести на ринок нові продукти та послуги [9].

Microsoft співпрацює з ECMA, органом міжнародних стандартів, щоб створити стандарт для C#. Визнання міжнародної організації стандартів (ISO) для C# заохочувало б інші компанії розробляти власні версії мови. Компанії, які вже використовують C#, включають Apex Software, Bunka Orient, Component Source, devSoft, FarPoint Technologies, LEAD Technologies, ProtoView та Seagate Software.

Для розробки на мові C# було вибрано Visual Studio 2019, що є найбільш зручним та легким у використанні.

Visual Studio – це інтегрованого середовище розробки (IDE) від компанії Microsoft, яке можна використовувати для розробки консольних програм, графічних інтерфейсів користувача (GUI), Windows Forms, веб-служб та веб-додатків.

Visual Studio використовується для запису нативного коду та керованого коду, що підтримується Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Редактор коду Visual Studio підтриму IntelliSense та рефакторинг коду, тоді як інтегрований налагоджувач Visual Studio підтримує налагодження як вихідного, так і машинного рівня. Visual Studio включає інші вбудовані інструменти, наприклад дизайнер форм, який корисний при створенні програм GUI; веб-дизайнер, який створює динамічні веб-сторінки; дизайнер класів, який використовується для створення користувацьких бібліотек, і конструктор схем для підтримки бази даних.

Розробка клієнтської частини системи відбувається на основі каркасу Angular 8 версії із використання мови програмування TypeScript.

TypeScript (TS) – мова програмування з відкритим кодом, створена Microsoft. Він пов'язаний з JavaScript (JS) завдяки тому, що він компілюється в нього. Це означає, що код, написаний у TypeScript, автоматично переводиться на JavaScript. Простіше кажучи – TS - це «накладення» JS. Найважливішою його особливістю є можливість статичного набору тексту та об'єктно-орієнтованого програмування на основі класів. У контексті

JavaScript часто використовується формулювання, яке називається набором JavaScript TypeScript JavaScript.

Для написання коду мовою програмування TypeScript, JavaScript найбільш зручним редактором виявився Visual Studio Code із безліччю плагінів та легким налаштуванням.

Visual Studio Code – це простий, але потужний редактор вихідного коду, який працює на десктоп комп'ютерах та доступний для Windows, macOS та Linux. Він оснащений вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов (таких як C ++, C #, Java, Python, PHP, Go) та режимів виконання (таких як .NET і Unity).

1.3.2 Вибір СУБД та опис її фізичної моделі

Взаємодія з базою даних при розробці на основі ASP.NET каркасу як згадувалось раніше відбувається за допомогою ORM Entity framework (EF). Даний фреймворк підтримую велику кількість джерел даних, а саме: MySQL, MS SQL Server, MongoDB, PostgreSQL. При розробці даної системи використовуватиметься MS SQL Server через хорошу інтеграцію із стеком технологій від Microsoft.

Як було раніше сказано при формуванні фізичної моделі бази даних буде задіяний Code-First підхід. Загальний принцип побудови за даною практикою полягає у наступних кроках:

- створенні окремого класу до кожної із сутностей;
- створення класів конфігурацій до кожного класу сутностей;
- створення та налаштування контексту із сервером бази даних;
- формування фабрики по створенню контексту;
- генерування файлів міграцій;
- виконання міграції.

Приступимо до першого кроку перенесення сутностей у класи C#. Для цього необхідно створити папку Data та у ній підкаталог Entities. Дана папка

міститиме файли класів що описують сутності. Розглянемо лістинг ключового класу UserEntity який наведено у лістингу 1.1.

Лістинг 1.1 – Клас UserEntity

```
public class UserEntity
{
  [Key]
  public Guid Id { get; set; }
  public string UserName { get; set; }
  public string Password { get; set; }
  public bool IsActive { get; set; }
  public bool IsOnline { get; set; }
  public bool IsMaster { get; set; }
  public DateTime LastActive { get; set; }
  public string Email { get; set; }
  public string PhoneNumber { get; set; }
  public string Location { get; set; }
  public string AdditionalInfo { get; set; }

  public ImageEntity Image { get; set; }
  public List<SubscriberSetEntity> SubscriberSets { get; set; }
  public List<PostEntity> Posts { get; set; }
  public List<UserBookEntity> Books { get; set; }
  public List<AgreementEntity> Agreements { get; set; }
}
```

З лістингу 1.1 видно що кожний атрибут сутності описується як проста властивість класу із типами мови програмування. Для опису зв’язків між сутностями використовується колекція List<T> за допомогою T вказується інший клас як тип що міститиметься у колекції при вибірці. У класі нащадкові згідно схеми бази даних буде розташоване поле з типом класу батька тобто UserEntity. Лістинг прикладу класа нащадка PostEntity відносно UserEntity наведено у лістингу 1.2.

Лістинг 1.2 – Клас PostEntity

```
public class PostEntity
{
  [Key]
  public Guid Id { get; set; }
  public string Title { get; set; }
  public string Description { get; set; }
  public DateTime CreatedOn { get; set; }
  public Guid UserId { get; set; }

  public ImageEntity Image { get; set; }
  public UserEntity User { get; set; }
  public List<CommentEntity> Comments { get; set; }
```

```
  public List<MarkEntity> Marks { get; set; }
  public EventEntity Event { get; set; }
}
```

Всі інші класи що описують сутності бази даних та зв’язки між ними будуються за аналогічним алгоритмом то детальний розгляд їх недоцільний. Описавши всі класи сутностей можна виконати їхню конфігурацію. Розглянимо на прикладі класу конфігурації UserBookEntityConfiguration для сутності UserBookEntity яка наведена у лістингу 1.3.

Лістинг 1.3 – Клас UserBookEntityConfiguration

```
public class UserBookEntityConfiguration:
IEntityTypeConfiguration<UserBookEntity>
{
  public void Configure(EntityTypeBuilder<UserBookEntity> builder)
  {
    builder.HasKey(bs => bs.Id);
    builder.Property(bs => bs.Id).IsRequired().ValueGeneratedOnAdd();
    builder.Property(bs => bs.Copies).IsRequired().HasDefaultValue(1);
    builder.Property(bs => bs.UserId).IsRequired();
    builder.Property(bs => bs.BookId).IsRequired();

    builder.ToTable("UserBook");

    builder.HasOne(r => r.User)
        .WithMany(p => p.Books)
        .HasForeignKey(b => b.UserId)
        .OnDelete(DeleteBehavior.Restrict);

    builder.HasOne(r => r.Book)
        .WithMany(p => p.UserBooks)
        .HasForeignKey(p => p.BookId)
        .OnDelete(DeleteBehavior.Restrict);

    builder.HasMany(r => r.Agreements)
        .WithOne(p => p.UserBook)
        .HasForeignKey(p => p.UserBookId)
        .OnDelete(DeleteBehavior.Restrict);
  }
}
```

Як видно з лістингу 1.3 конфігурації EF сутності виконується у класі що реалізовує інтерфейс IEntityTypeConfiguration<T>, де в ролі T виступає клас сутності який конфігурується. В методі Configure передається EntityTypeBuilder<UserBookEntity> тип який виступає мостом між сутністю та конфігурацією. За допомогою даного вхідного параметру відбувається доступ до полів через лямда конструкції. Кожне поле конфігурується на

зазначення розміру поля, чи вимагається обов’язкова ініціалізація при внесенні, генарція даних по замовчуванню і так далі. Після опису полів відбувається налаштування зв’язків. В даних конструкціях виконується явний вибір полів в які будуть вставлятись дані за потреби, а також зовнішні ключі. Всі інші конфігурування сутностей відбувається аналогічно тому на них зупинятись не будемо.

Описавши конфігурації наступним кроком є створення та конфігурування контексту підключення до MS SQL Server та фабрики створення контексту для подальшого використання в сервісах. Клас контексту та фабрики по створенню контексту наведено у лістингу 1.11 та 1.12.

Лістинг 1.4 – Клас UserBookEntityConfiguration

```
public class LibsetContext : DbContext, ILibsetContext
{
  public LibsetContext(DbContextOptions dbContextOptions) :
base(dbContextOptions) {}

  public DbSet<UserEntity> Users { get; set; }
  public DbSet<AgreementEntity> Agreements { get; set; }
  public DbSet<BookEntity> Books { get; set; }
  public DbSet<UserBookEntity> UserBooks { get; set; }
  public DbSet<CommentEntity> Comments { get; set; }
  public DbSet<EventEntity> Events { get; set; }
  public DbSet<ImageEntity> Images { get; set; }
  public DbSet<MarkEntity> Marks { get; set; }
  public DbSet<PostEntity> Posts { get; set; }
  public DbSet<SubscriberSetEntity> SubscriberSets { get; set; }

  protected override void OnModelCreating(ModelBuilder modelBuilder)
  {
      modelBuilder.ApplyConfiguration(new UserEntityConfiguration());
      modelBuilder.ApplyConfiguration(new
      AgreementEntityConfiguration());
      modelBuilder.ApplyConfiguration(new BookEntityConfiguration());
      modelBuilder.ApplyConfiguration(new
      UserBookEntityConfiguration());
      modelBuilder.ApplyConfiguration(new CommentEntityConfiguration());
      modelBuilder.ApplyConfiguration(new EventEntityConfiguration());
      modelBuilder.ApplyConfiguration(new ImageEntityConfiguration());
      modelBuilder.ApplyConfiguration(new MarkEntityConfiguration());
      modelBuilder.ApplyConfiguration(new PostEntityConfiguration());
      modelBuilder.ApplyConfiguration(new
      SubscriberSetEntityConfiguration());

      base.OnModelCreating(modelBuilder);
  }
```

```
}
```

З лістингу 1.4 видно що клас контексту повинен обов’язково наслідувати базовий клас EF DbContext з якого необхідно переоприділити метод OnModelCreating в якому вказуються конфігурації сутностей які описані вище.

Для зручного використання та додавання контексту у контейнер DI створимо фабрику яка буде цим займатись. Для цього необхідно свторити клас що буде реалізовувати інтерфейс IDesignTimeDbContextFactory<T>, де в якості T виступає клас контексту в нашому випадку це LibsetContext. Далі в методі CreateDbContext що був реалізований з вище вказаного інтерфейсу додається стрічка підключення та у середні об’єкта опцій передається при створенні контексту як параметр. Клас даної фабрики наведено у лістингу 1.5.

Лістинг 1.5 – Клас LibsetContextFactory

```
public class LibsetContextFactory :
IDesignTimeDbContextFactory<LibsetContext>
{
  public LibsetContext CreateDbContext(string[] args)
  {
      var optionsBuilder = new DbContextOptionsBuilder<LibsetContext>();
      optionsBuilder.UseSqlServer("connectionString");
      return new LibsetContext(optionsBuilder.Options);
  }
}
```

На основі створених класів сутностей, конфігурацій та налаштувань для підключення до бази даних можна виконувати генерування міграції. Для автогенеруання міграції необхідно відкрити  Package Manager Console або PowerShell та виконати одну з наступних відповідно команд з кореня папки де знаходиться проект:

`Add-Migration InitialCreate` – для Package Manager Console

`dotnet ef migrations add InitialCreate` – для PowerShell

### 1.3.3 Реалізація основних класів та методів

#### 1.3.3.1 Серверна частина

Додаток на основі ASP.NET Core Web API базується на архітектурі MVC як і більшість веб-фреймворків. В попередньому розділі було описано складову Models яку реалізовують контекст та сутності бази даних. Наступним у етапом розробки є написання сервісів які взаємодіятимуть з шаром бази даних та міститимуть в собі бізнес логіку. Також необхідно організувати взаємодію між сервісами за допомогою Dependency injection.

Розпочнемо з сервісу для ідентифікації користувача та видачі JSON Web Token для зареєстрованих користувачів. Даний сервіс реалізовуватиме інтерфейс IIdentityService. Лістинг даного інтерфейсу наведено у лістингу 1.6.

Лістинг 1.6 – Інтерфейс IIdentityService

```
public interface IIdentityService
{
  Task<string> GetTokenAsync(UserEntity userEntity);
  Task<UserEntity> GetCurrentUserByTokenAsync(string token);
  Task<UserEntity> AddUserAsync(UserEntity userEntity);
}
```

Сервіс що реалізовуватиме даний інтерфейс виконуватиме видачу JWT за іменем та паролем користувача. Приклад формування JWT наведено у лістингу 1.7.

Лістинг 1.7 – Формування JWT

```
var claims = new Claim[]
{
  new Claim(JwtRegisteredClaimNames.Sub, User.UserName),
  new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
  new Claim(JwtRegisteredClaimNames.Iat,
now.ToUniversalTime().ToString(),
  ClaimValueTypes.Integer64)
};
```

```
var signingKey = new SymmetricSecurityKey(Encoding.ASCII
  .GetBytes("Y2F0Y2hlciUyMHdvbmclMjBsb3ZlJTIwLm5ldA=="));

var jwt = new JwtSecurityToken(
issuer: "http://libset.com",
audience: "Haiduk Roman",
claims: claims,
notBefore: now,
expires: now.Add(TimeSpan.FromDays(30)),
);
var encodedJwt = new JwtSecurityTokenHandler()
  .WriteToken(jwt);
```

Спочатку відбувається формування Claim на основі ключа, імені користувача та поточного часу. Далі додається ключ для шифрування та деякі додаткові дані такі як посилання на сайт. Після чого формується JWT та повертається клієнту.

Наступний сервіс реалізовує інтерфейс IUserService, який відповідатиме за інформацію про користувачів, підписників та підписок. Даний інтерфейс описано в лістингу 1.8.

Лістинг 1.8 – Інтерфейс IUserService

```
public interface IUserService
{
  Task<UserEntity> GetUserAsync(string userName);
  Task<List<SubscriberSetEntity>> GetUserSubscribersAsync(string
userName);
  Task<List<SubscriberSetEntity>> GetUserIdolsAsync(string userName);
  Task<SubscriberSetEntity> SubscribeAsync(Guid userId, Guid
subscribeId);
  Task UnsubscribeAsync(Guid userId, Guid subscribeId);
}
```

Наступний сервіс реалізовує інтерфейс IFeedService, що слугує для роботи із стрічкою новин, а саме:

- свторення нвого поста;
- коментування поста;
- маркування поста;
- перегляд постів від користувачів з ваших підписок;
- перегляд найближчих подій.

Даний інтерфейс описано у лістингу 1.10.

Лістинг 1.10 – Інтерфейс IUserService

```
public interface IFeedService
{
  Task<List<PostEntity>> GetUserFeedAsync(Guid userId, int start, int
  number);
  Task<PostEntity> AddPostAsync(PostEntity postEntity);
  Task DeletePostAsync(Guid postId);
  Task<PostEntity> GetPostAsync(Guid postId);
  Task<CommentEntity> AddCommentAsync(CommentEntity commentEntity);
  Task<MarkEntity> AddMarckAsync(MarkEntity markEntity);
}
```

Наступний сервіс реалізовуватиме інтерфейс IBookService і виконуватиме роботу з книгами, а саме:

- пошук книг в базі даних;
- повернення базової інформації про книгу;
- додавання та видалення книги користувача.

Опис даного інтерфейсу наведено у лістингу 1.11.

Лістинг 1.11 – Інтерфейс IBookService

```
public interface IBookService
{
  Task<BookEntity> AddBook(Guid userId, BookEntity bookEntity);
  Task<List<UserBookEntity>> GetUserBooks(string userName);
  Task DeleteUserBookAsync(Guid userBookId);
  Task<List<BookEntity>> FindBooksAsync(string searchValue);
  Task<BookEntity> GetBook(Guid bookId);
}
```

Останні із сервісів реалізовуватиме інтерфейс IAgreementService та відповідатиме за роботу з угодами між користувачами по обміну чи оренді книги в бібліотеці. Опис даного інтерфейсу наведено у лістингу 1.12.

Лістинг 1.12 – Інтерфейс IAgreementService

```
public interface IAgreementService
{
  Task<AgreementEntity> AddAgreementAsync(AgreementEntity
  agreementEntity);
  Task<bool> ChangeAgreementStatusAsync(Guid id, AgreementStatus
      agreementStatus, int days = 0);
```

```
    Task<List<AgreementEntity>> GetAgreementsByUser(Guid userId);
    Task<List<AgreementEntity>> GetAgreementsByUserBook(Guid userBookId);
}
```

Реалізувавши всі сервіси та маючи вже описаний контекст бази даних необхідно їх додати до контейнера DI. Цей процес відбувається у файлі Startap.cs в методі ConfigureServices. Блок коду який це реалізовує міститься в лістингу 1.13.

Лістинг 1.13 – Додавання сервісів у DI контейнер

```
services.AddScoped<IDesignTimeDbContextFactory<LibsetContext>,
LibsetContextFactory>();
services.AddScoped<IIdentityService, IdentityService>();
services.AddScoped<IBookService, BookService>();
services.AddScoped<IFeedService, FeedService>();
services.AddScoped<IUserService, UserService>();
services.AddScoped<IAgreementService, AgreementService>();
```

Після обробки інформації у сервісах необхідно створити точки для повернення даних клієнту у форматі JSON. Цю роботу виконують контролери.

Контролер - це клас як часто буває з атрибутом Produces, що містить формат повернення об’єктів та методи з атрибутом одного з наступних типів запиту:

- HttpDelete;
- HttpGet;
- HttpPost;
- HttpPut.

Для використання сервісу у контролері чи в інших класах необхідно його вказати у контролері та присвоїти відповідному закритому полю класа яке отримуватиме об’єкт сервісу для подальшого виклику відповідних методів сервісу. Опис структури класу одного із контролерів, а саме IdentityController наведено у лістингу 1.14.

Лістинг 1.14 – Клас контролера IdentityController

```
[Route("api/identity"), Produces("application/json"),
DisableRequestSizeLimit]
[ApiController]
public class IdentityController : Controller
{
  private readonly IIdentityService _identityService;
  private readonly IMapper _mapper;
  public IdentityController(IIdentityService identityService, IMapper
mapper)
  {
      _identityService = identityService;
      _mapper = mapper;
  }
  [HttpGet("~/api/token")]
  public async Task<IActionResult> GetToken(string userName, string
password)
  {
    var user = new UserEntity()
    {
        UserName = userName,
        Password = password
    };
    var token = await _identityService.GetTokenAsync(user);
    if (token == null)
    {
        return Unauthorized();
    }
    else
    {
        return Ok(token);
    }
  }
}
```

В якості захисту від несанкціонованого доступу до API використовується програмне забезпечення проміжного рівня (Middleware).

Для цього необхідно створити клас який в собі буде містити обовязковий методо Invoke який приймає в якості параметру HttpContext запитів, а також приватне полу класу типу RequestDelegate, що забеспешить перехід до наступного Middleware класу або контролера. Опис даного класу наведено у лістингу 1.15.

Лістинг 1.20 – Клас TokenMiddleware

```
public class TokenMiddleware
{
  public List<RouteModel> Routes { get; set; }

  private readonly IdentityService _identityService;
  private readonly RequestDelegate _next;

  public TokenMiddleware(RequestDelegate next, IdentityService
identityService)
```

```
    {
      _next = next;
      _identityService = identityService
    }

    public async Task InvokeAsync(HttpContext context)
    {
      if (context.Request.Path == "/api/tokens")
      {
        await _next.Invoke(context);
      }
      else
      {
            var authorizationHeader =
        context.Request.Headers["Authorization"].ToString();
        var authorizationValues = authorizationHeader.Split(" ");
        StringValues token;

        if (authorizationValues.Count() > 1 && authorizationValues[0] ==
        "Bearer" || context.Request.Query.TryGetValue("token", out token))
        {
              var token = String.IsNullOrEmpty(token) ?
        authorizationValues[1] : $"{token}";

          if (_identityService.ValidateToken(token))
              await _next.Invoke(context);

          context.Response.StatusCode = (int)HttpStatusCode.Unauthorized;
        }
      }
    }
}
```

1.3.3.2 Клієнтська частина

Для створення клієнтської частини як було зазначено в попередніх розділах в якості каркасу буде використовуватись Angular 8 версії. Щоб розпочати створення додатку необхідно для початку згенерувати початковий скелет який буде в подальшому розширюватись. Для цього можна скористатись інтерфейсом командної стрічки (CLI) Angular та ввести наступну команду:

```
ng new client-app
```

Після виконання даної команди створиться початковий код додатку з корінним модулем AppModule. Називається корінним він тому що слугує точкою входу і є батьківським для всіх інших дочірніх модулей.

Далі можна перейти до розробки власних модулей які відповідатимуть за певну частину логіки системи. Основа кожного модуля складають файл самого модуля, внутрішня маршрутизація, компоненти.

Враховуючи предметну область та уже спроектовану базу даних створимо наступні модулі:

- Profile module;
- Feed module;
- Book module;
- Agreement module.

Також крім глобальних модулей створимо наступні компоненти для забезпечення входу та реєстрації в системі:

- Login component;
- Registration component.

Для створення відповідних модулей та компонент необхідно скористатись рядом наступних CLI команд:

`ng generate module` – генерування модуля

`ng generate component` – генерування компоненти

Після виконання даних команд створено відповідні файли із кодом по замовчуванню. Далі необхідно приступати до наповнення модулей бізнес логікою, верстkою компонент сторінки та взаємодією із серверною частиною програми.

Розпочнем з модуля Profile, який відображатиме сторінку профілю користувача. Дана сторінка буде складатись з фотографії користувача, даних про підписників та підписки, блок з переліком книг що є у власній бібліотеці, панель з інструментами. Опис даного модуля зображено у лістингу 1.19.

Лістинг 1.19 – Клас модуля Profile

```
@NgModule({
    imports: [
```

```
        CommonModule,
        RouterModule.forChild(UserRoutes),
        FormsModule,
        MatDialogModule,
        MaterialModule,
    ],
    declarations: [
        UserComponent,
        UserChangePasswordDialogComponent
    ],
    exports: [
        UserChangePasswordDialogComponent
    ],
})
export class ProfileModule {}
```

Як видно з лістингу 1.19 кожен модуль розпочинається з декоратора @NgModule. Декоратор – це можливість додання до класів чи методів метаданих для зміни їх поведінки без зміни їх коду. В даному випадку в якості параметру передається об’єкт з модулями що імпортуються, компоненти що задекларовані в даному модулі, та компоненти що експортуються для використання в інших модулях. В кінці експортується поточний клас модуль.

Модуль Feed  відповідає за стрічку новин користувача. На сторінці даного модуля розміщено пости користувачів за якими ведеться стеження, найближчі події, позначення публікацій певним маркером. Також є можливість створення власного поста який відображатиметься підписникам.

Модуль Book міститиме сторінку для перегляду книги через Embed читанку від Google Book API. На даній сторінці відображатиметься також і користувачі з даною книго та перегляд їх на карті для обрання найбільш оптимального для обміну.

Модуль Agreement відповідає за моніторинг виданих чи орендованих книг користувачем. Це відбувається за допомогою різниці між поточною та виданою датах і відображається у вигляд кількості днів що залишились.

Структура та принцип побудови даних модулі є ідентичною до Profile тому у виведення лістингу немає потреби.

Angular реалізовує впровадження залежностей (DI) для зручного використання сервісів системи. Сервіс – це клас з декоратором @Injectable за

допомогою якого реєструється даний клас за принципом патерну Singleton. В сервісах велика частина методів виконують роль запитів до сервера обробляючи за потреби інформацію що повертається у зручні структурі для відображення у користувацькому інтерфейсі. В додатку передбачено наступні сервіси:

- book service;
- feed service;
- identity service;
- user service;
- agreement.

Identity сервіс виконує роботу отримання JWT від сервера по імені та паролеві користувача після чого зберігає його в локальному сховищі. Об’єкт з інформацією про користувача заповнюється одноразово при вході після чого зберігається в сервісі і може бути отриманий при звернення ді відповідного поля класу по гетеру. Опис даного класу наведено у лістингу 1.20.

Лістинг 1.19 – Клас IdentityService

```
@Injectable({
  providedIn: 'root'
})
export class IdentityService {

    private baseUrl = 'https://localhost:44349/';
    private _user: any;

    get user() {
        return this._user;
    }

    constructor (private http: HttpClient) { }

    private getCurrentUser(): Promise<any> {
        return
this.http.get<any>(`${this.baseUrl}api/users/current?token=${localStorage.getI
tem('token')}`).toPromise();
    }

    async loadCurrentUser(): Promise<void> {
        this._user = await this.getCurrentUser();
    }
```

```
    async login(userName: string, password: string): Promise<void> {
        const token = await
this.http.get<any>(`${this.baseUrl}api/token?userName=${userName}&password=${p
assword}`).toPromise();
        localStorage.setItem('token', token);
        await this.loadCurrentUser();
    }
}
```

Всі інші сервісі мають подібну структуру і містять в собі відповідні до вимог модуля методи HTTP запитів до сервера.

Одним з важливих аспектів розробки клієнтського додатку є організація внутрішньої безпеки так як через доступ до певних її частин можливе витікання приватної інформації. Аутинтифікація всередені додатку відбувається за допомогою «захисника» (Guard). При переході на кожний новий маршрут всередині додатку відбувається перевірка на наявність JWT та користувача, якщо вони присутні тоді пропускають перехід на даний маршрут за відсутності відбувається перенаправлення на сторінку входу. Опис AuthGuard наведено у лістингу 1.21.

Лістинг 1.21 – Клас AuthGuard

```
@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router
    ) { }
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        if (localStorage.getItem('token')) {
            return true;
        }
        this.router.navigate(['/login']);
        return false;
    }
}
```

Для розробки перегляду локацій на картах використаємо безкоштовне API від компанії MapBox. Для роботи з даною картою необхідне лише підключення додаткових скриптів, які забезпечуватимуть стилізацію та взаємозв’язок із сервером. Приклад створення полотна із картою наведено у лістингу 1.22.

Лістинг 1.21 – Функція створення полотна карти MapBox

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>View a fullscreen map</title>
<meta name="viewport" content="initial-scale=1,maximum-scale=1,user-
scalable=no" />
<script src="https://api.tiles.mapbox.com/mapbox-gl-js/v1.6.0/mapbox-
gl.js"></script>
<link href="https://api.tiles.mapbox.com/mapbox-gl-js/v1.6.0/mapbox-
gl.css" rel="stylesheet" />
<style>
      body { margin: 0; padding: 0; }
      #map { position: absolute; top: 0; bottom: 0; width: 100%; };
</style>
</head>
<body>
<div id="map"></div>

<script>
mapboxgl.accessToken = '<your access token here>';
var map = new mapboxgl.Map({
container: 'map', // container id
style: 'mapbox://styles/mapbox/outdoors-v11', //stylesheet location
center: [11.255, 43.77], // starting position
zoom: 13 // starting zoom
});

map.addControl(new mapboxgl.FullscreenControl());
</script>

</body>
</html>
```

# 2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

## 2.1 Модульне тестування

Модульне тестування (Unit Testing) – це методика тестування програмного забезпечення, за допомогою якої окремі одиниці програмного забезпечення, тобто група модулів комп'ютерної програми, процедури використання та операційні процедури тестуються для визначення того, чи підходять вони для використання чи ні. Це метод тестування, за допомогою якого кожен незалежний модуль тестується, щоб визначити, чи є якісь проблеми безпосередньо самим розробником. Це співвідноситься з функціональною коректністю незалежних модулів [11].

Модульне тестування визначається як тип тестування програмного забезпечення, де тестуються окремі компоненти програмного забезпечення.

Написання тестів програмного продукту проводиться під час розробки програми. Окремий компонент може бути або окремою функцією, або процедурою. Модульне тестування, як правило, проводиться розробником.

У моделі SDLC яка наведена на рисунку 2.1 – це перший рівень тестування, який проводиться перед інтеграційним тестуванням.

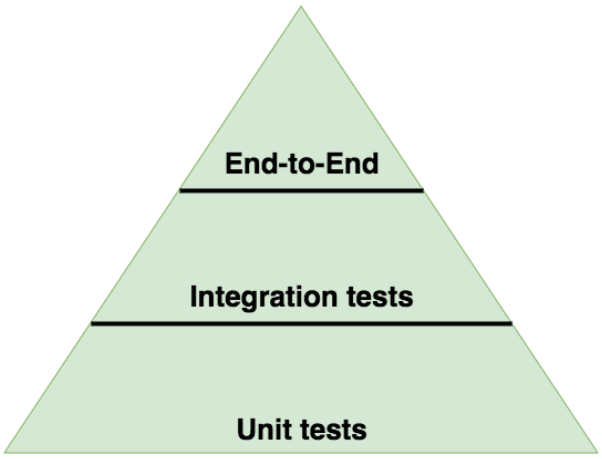


Рисунок 2.1 – Схема тестування програмного забезпечення

Для написання тестів скористаємось можливостями Visual Studio 2019. Відкривши розроблене рішення необхідно натиснути на ньому правою клавішею миші та вибрати в контекстному меню пункт «Add project». Після чого у вікні що появилось вибрати MSTest Test Project (.NET Core) (рис. 2.2).

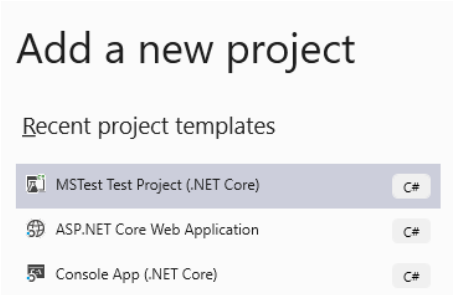


Рисунок 2.2 – Додавання проекту із тестами

Після створення проекту в якому будуть міститись класи із Unit тестуванням перед початком безпосереднього написання тестів необхідно

додати посилання на проект що буде тестуватись. Для цього необхідно натиснути правою клавішею миші на «Dependencies» та вибрати пункт «Add reference». У відповідному вікні потрібно вибрати проект в даному випадку це Libset (рис. 2.3).

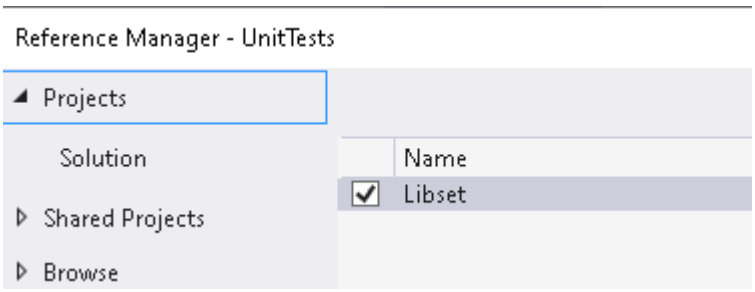


Рисунок 2.3 – Додавання проекту Libset до проекту тестування

Виконавши початкові налаштування можна приступати до написання тестів. Для цього необхідно створити клас Tests в якому за допомогою методів можна буде описати тести до методів для перевірки вхідних і вихідних параметрів. Розглянемо один із методів класу Tests, а саме UserAndPasswordAddUserAsync який виконуватиме перевірку на првильність роботи методу AddUserAsync сервісу IdentitySer. Даний тест зображено у лістингу 2.1.

Лістинг 2.1 –Методу UserAndPasswordAddUserAsync

```
[TestClass]
public class UnitTest1
{
  [TestMethod]
  public void UserAndPasswordAddUserAsync()
  {
    var userEntity = new UserEntity()
    {
        UserName = "roman",
        Password = "roman"
    };
    var optionsBuilder = new DbContextOptionsBuilder<LibsetContext>();
    var libsetContext = new LibsetContext(optionsBuilder.Options);
    var identityServiceMock = new Mock<IIdentityService>();
    var _dbContextFactory = new
Mock<IDesignTimeDbContextFactory<LibsetContext>>();
    _dbContextFactory.Setup(x =>
x.CreateDbContext(null)).Returns(libsetContext);
    var libsetContextMock = new Mock<LibsetContext>();
```

```
    libsetContextMock.Setup(x => x.Users.FirstOrDefault(u => u.UserName
== userEntity.UserName)).Returns(userEntity);
    var res = identityServiceMock.Setup(x =>
x.AddUserAsync(userEntity)).ReturnsAsync(userEntity);

    Assert.Equals(userEntity, res);
  }
}
```

З лістингу 2.1 видно що кожний клас тесту має атрибут TestClass та в свою чергу кожний метод має атрибут TestMethod. Загальний алгоритм побудови кожного Unit тесту наступний:

- опис даних які очікуються при поверненні;
- заглушення методів які виконуються для завершення тестуючого методу;
- виклик методу із відповідними вхідними параметрами;
- перевірка рівність очікуваного та фактичного результату виконання методу.

Детальний огляд всіх інших тестів немає необхідності так як вони будуються подібними чином.

Написавши всі тести можна приступити до перегляду результатів походження або не проходження тестів. Для цього необхідно в головного меню Visual Studio вибрати пункт «Test» потім «Test explorer» що запустить вікно дерева тестів. В даному вікні для запуску на виконання тестів необхідно натиснути на кнопку запуску у верхній панелі. Результати виконання тестів наведено на рисунку 2.4.

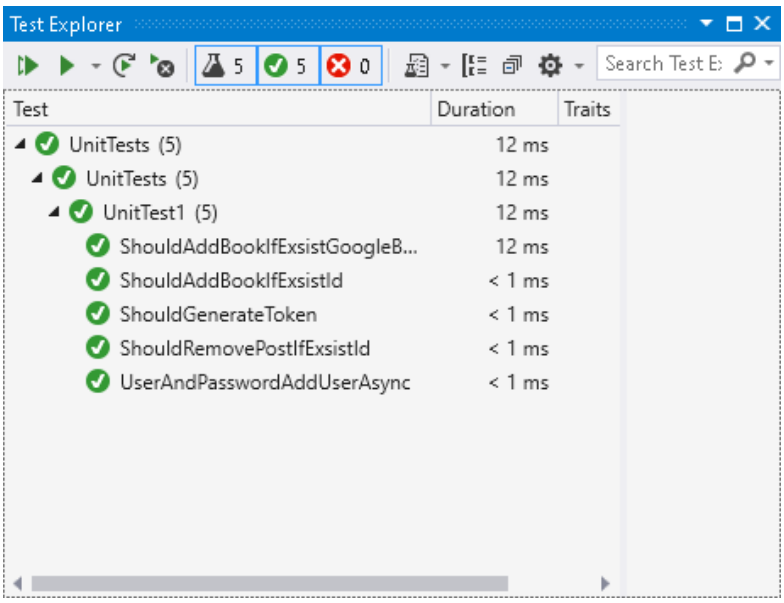


Рисунок 2.4 – Результат виконання тестів

Модульне тестування необхідне для забезпечення цілісності та правильності роботи коду та його основного алгоритму навіть при зміні його.

## 2.1 Розгортання програмної системи

В якості платформи для розгортання проекту виберемо Microsoft Azure так як це найбільш зручний хостинг для .NET додатків.

Microsoft Azure - це платформа для побудови та розміщення рішень за допомогою продуктів Microsoft та в їх центрах обробки даних. Це всеосяжний набір хмарних продуктів, що дозволяє користувачам створювати додатки корпоративного класу, не будуючи власну інфраструктуру.

Платформа служби Azure складається з трьох продуктів, орієнтованих на хмару: Windows Azure, SQL Azure та контролер додатка Azure App Fabric. Вони є додатком до інфраструктури хостингу додатків.

Платформа служби Azure - це значна частина ініціатив хмарних обчислень Microsoft. Він розроблений спеціально для хмари.

Microsoft Azure включає Windows Azure, яка є хмарною ОС, призначена для надання масштабованих послуг обчислення та зберігання.

Він підтримується Azure App Fabric, що представляє собою сукупність різних інструментів для підтримки додатків у хмарі. SQL Azure дозволяє зберігати, а також управляти даними, аналогічно звичайним послугам реляційних баз даних SQL-сервера.

Середовище розробки Visual Studio 2019 допоможе в розгортанні додатку у хмарі Azure. Для цього виконаємо наступні дії: відкрити проект за допомого VS 2019, натиснути праву клавішу миші на проекті, вибрати в контекстному меню пункт Публікація (Publish) див. рис. 2.5.

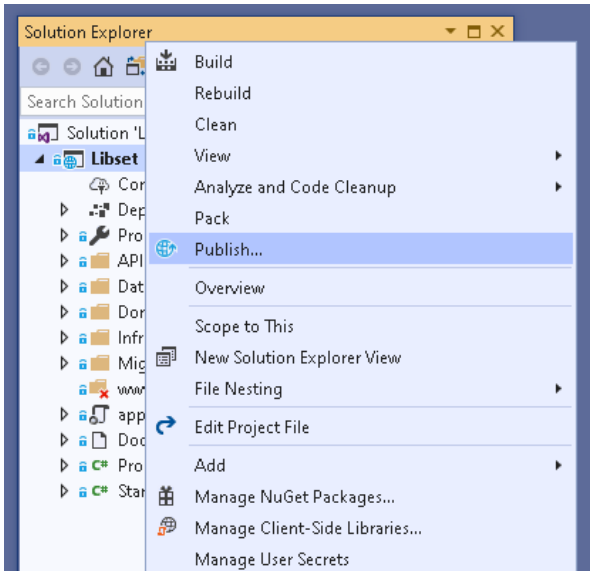


Рисунок 2.5 – Контекстне меню

Після виконання дій описаних вище на екрані появиться вікно майстра публікацій додатків. Після чого потрібно вибрати пункт App Service Linux так як розроблений додаток написаний на платформі .NET Core в якості операційної системи для серверної машини може використовуватись Linux, який є більш стабільним в якості серверної ОС. Вибравши всі необхідні параметри натиснемо Опублікувати (Publish) див. рис. 2.6.

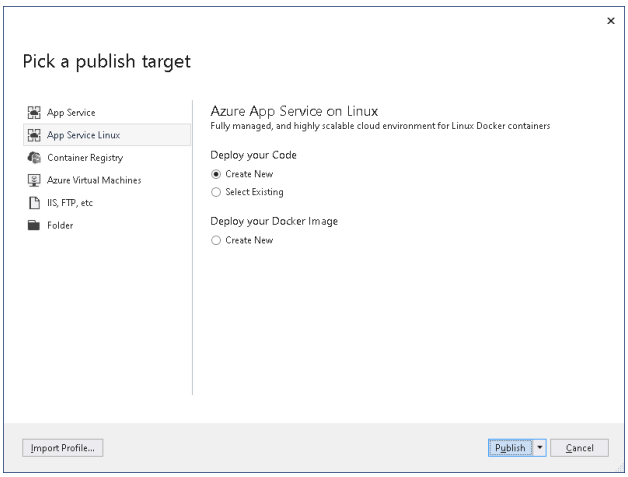


Рисунок 2.6 – Вікно вибору та створення сервісу

Далі необхідно вибрати додаткові ресурси яких потребує додаток. До них можна віднести базу даних MS SQL Server. Для цього потрібно вибрати необхідний ресурс та натиснути кнопку із значком “плюс” див. рис. 2.7.

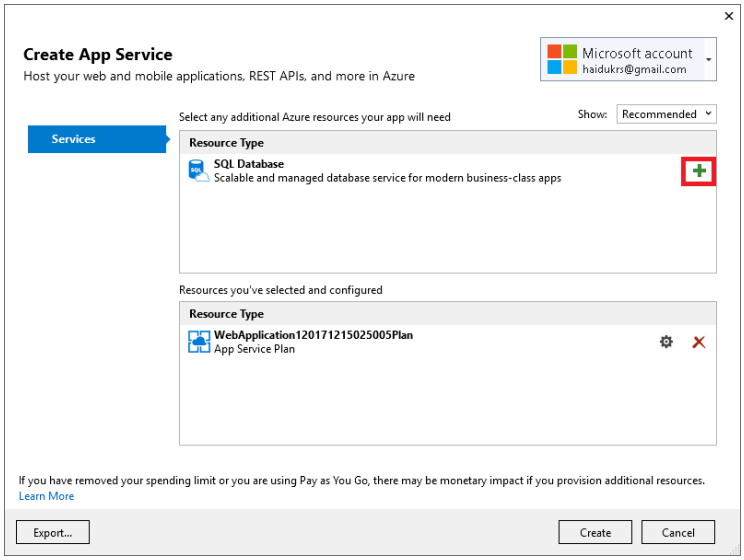


Рисунок 2.7 – Вікно додавання ресурсів

Після натиснення на кнопку Створити (Create) відкриється наступне вікно з налаштування підключення до бази даних. В даному вікні необхідно вибрати вкладку Налаштування (Settings), скопіювати стрічку підключення до бази даних та вибрати Execute Code First Migration, що виконає міграцію бази даних при запуску програми. Дане вікно зображено на рисунку 2.8.

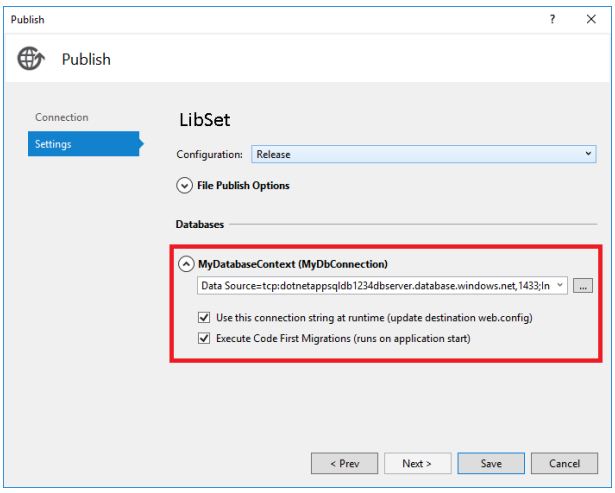


Рисунок 2.8 – Вікно налаштувань підключення до бази даних

Після виконання всіх вище зазначених дій створиться профіль розгортання , який може бути змінений або експортований на інші комп’ютери. Вікно готового профілю зображено на рисунку 2.9.

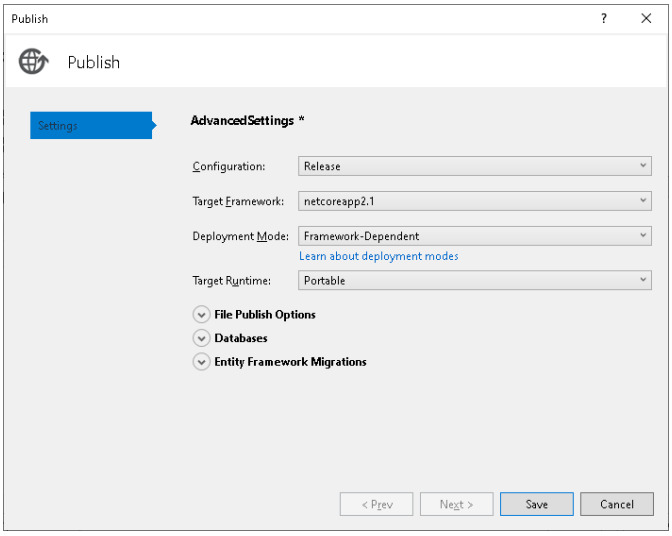


Рисунок 2.9 – Вікно профілю розгортань

Далі необхідно зберегти профіль розгортань та натиснути кнопку Почати (Start). Після чого перевірити роботу розгорнутого серверу за допомогою HTTP запиту. В результаті на сторінці вкладки браузера буде відображатись JWT стрічка (рис. 2.10).

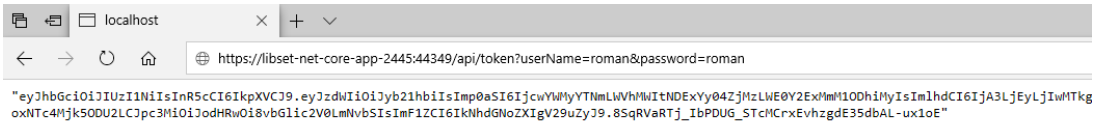


Рисунок 2.10 – Результат роботи розгорнутого серверу

## 2.2 Ілюстровані сценарії використання

Після того як було виконано розгортання системи можна приступити до тестування клієнтської та серверної частин разом.

При переході на сайт соціальної мережі спочатку відображатиметься сторінка входу в систему. Вона міститиме поля для імені користувача, пароля, кнопку «Увійти» та посилання «Реєстрація» (рис. 2.11).

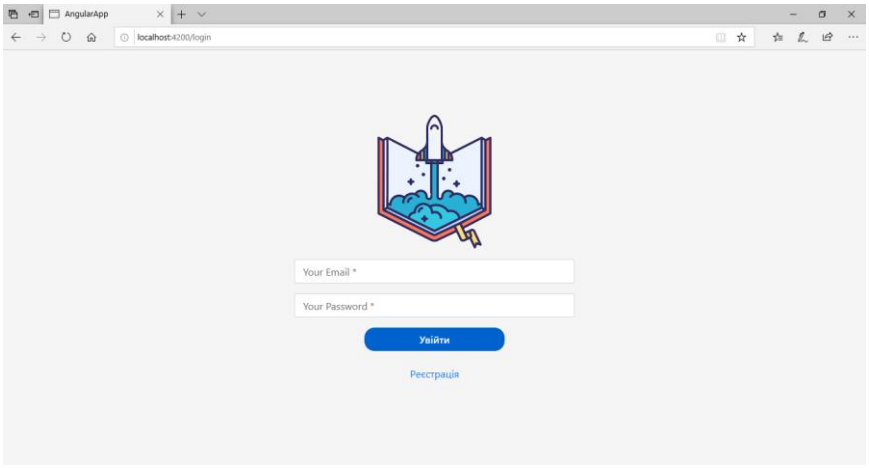


Рисунок 2.11 – Сторінка входу на сайт LibSet

Якщо облікового запису ще не існує необхідно натиснути на посилання «Реєстрація» яке перенаправить на сторінку реєстрації нового користувача (рис. 2.12).

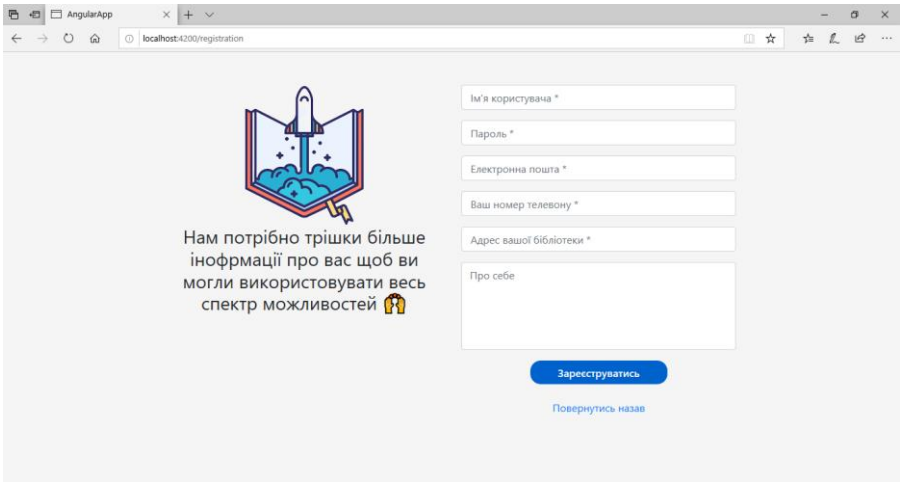


Рисунок 2.12 – Сторінка реєстрації нового користувача

При натисканні кнопки «Вхід» виконається вхід та перенаправить на початкову сторінку якою являється стрічка новин. На даній сторінці вгорі появляється головна панель навігації яка в собі містить наступні елементи:

- логотип та назву соціальної мережі;
- поле пошуку користувачів та книг;
- сповіщення;
- навігаційна кнопка переходу до угод;
- навігаційна кнопка переходу до профілю;
- додаткові дії (вихід із системи).

Нижче навігаційної панелі розміщено стрічку новин та найближчих подій. Сторінка стрічки новин наведено на рисунку 2.13.

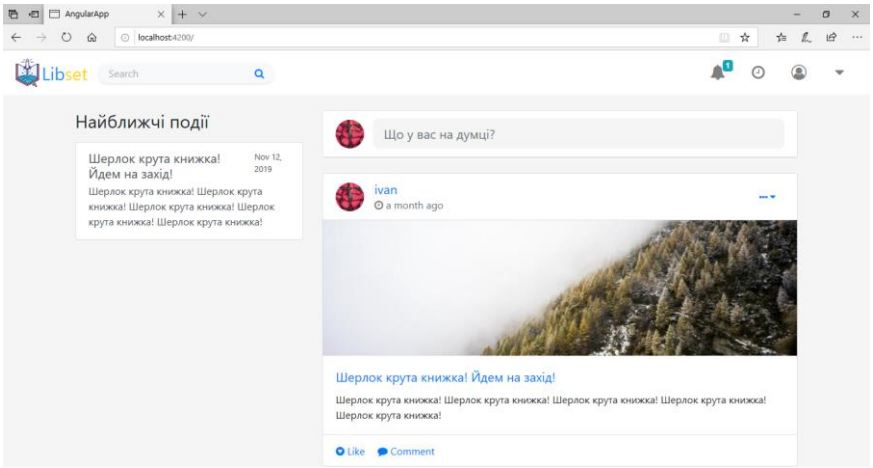


Рисунок 2.13 – Сторінка стрічки новин

Для створення нового посту необхідно навести курсор на першу картку та натиснути після чого появиться форма в якій будуть наступні елементи:

- заголовок;
- зображення;
- опис;
- чи пост є подією;
- локація та час проведення події;
- кнопка відміна та публікація.

Поля події відображаються при натисненні відповідного перемикача. При натисненні на кнопку публікації даний пост появиться у ваші та підписників стрічці новин. Форма створення нового посту наведено на рисунку 2.14.

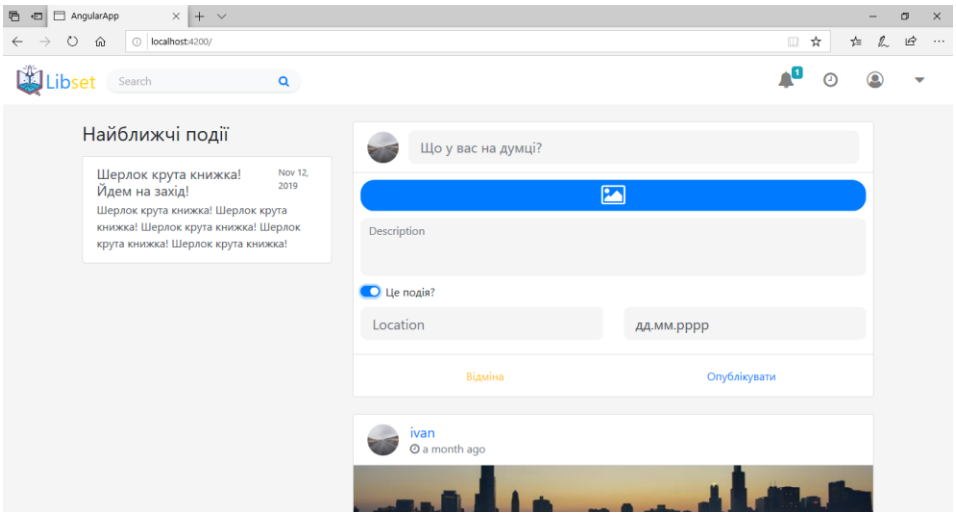


Рисунок 2.14 – Форма створення посту

Для переходу до сторінки власного профілю необхідно натиснути на відповідну кнопку навігації із зображенням «персони». Сторінка профілю складається із блоку інформації про користувача що мітиться в собі:

- фотографію профілю;
- ім’я користувача;
- кількості підписників та підписок;

- кнопки «Налаштувань» або «Підписатись» (якщо це профіль іншого користувача) .

Нижче розташований блок електронної бібліотеки із доданими книгами. Дана сторінка зображена на рисунку 2.15.

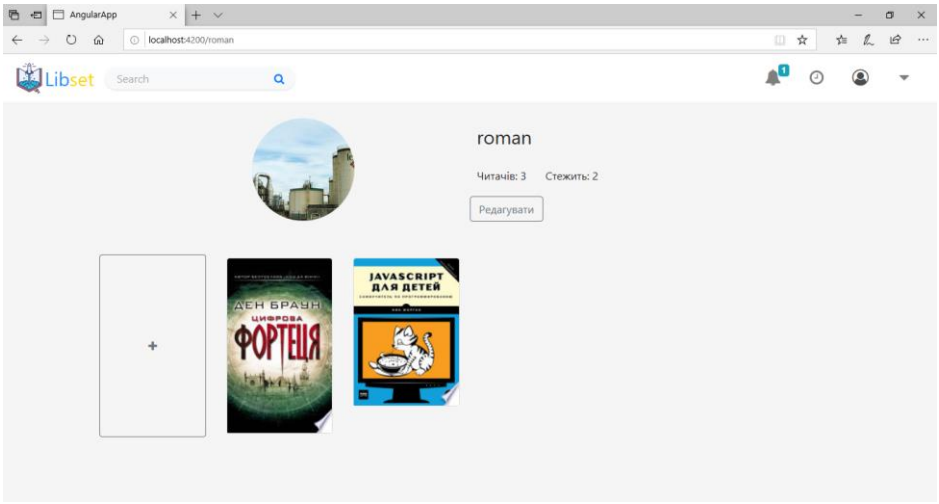


Рисунок 2.15 – Сторінка профілю користувача

Для додавання нової книги до електронної бібліотеки необхідно натиснути на кнопку із значком “плюса” після чого відкриється діалогове вікно пошуку необхідної книги із баз даних Google. З результатів пошуку необхідно на підходящій книзі натиснути кнопку із значком “плюса” (рис. 2.16).

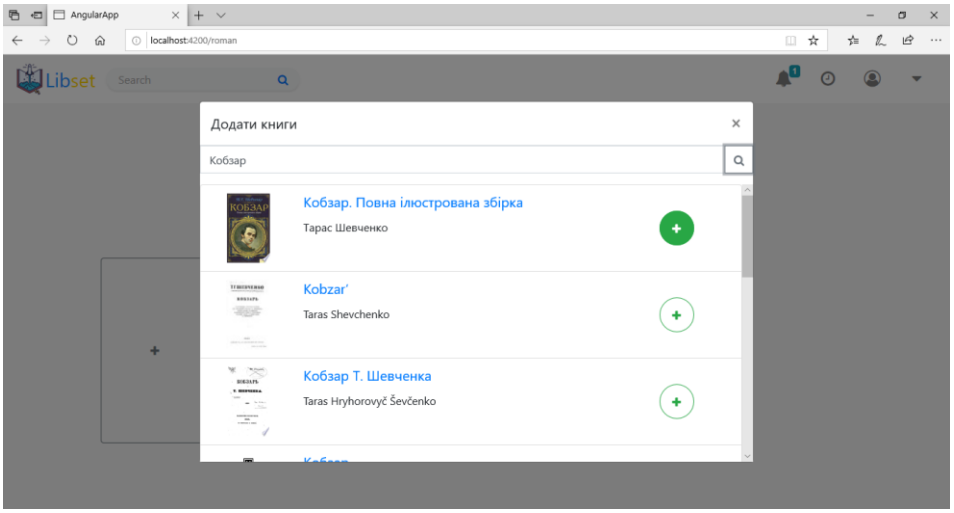


Рисунок 2.16 – Форма додавання нової книги

Після додавання нової книги діалог закриється, появиться сповіщення про успішне додавання та оновиться електронна бібліотека (рис. 2.17).

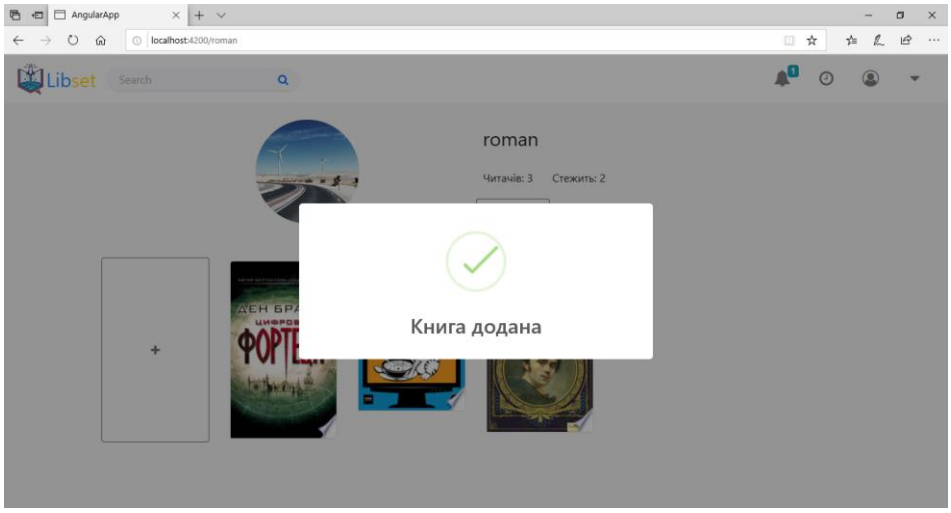


Рисунок 2.17 – Результат додавання книги

При наведенні на книгу висвітлюється коротка інформація про книгу, а саме назва як об’єкт посилання та автор (рис. 2.18).


Рисунок 2.18 – Коротка інформація про книгу

При натисненні на назву яка підсвічена синім кольором виконається перехід на сторінку книги. У верхній частин сторінки розташовані дані про книгу які в собі містять зліва перегляд всієї книги (якщо це можливо), а з права назва, автори та коротка анотація. У нижній частині розташовані дані

про бібліотеки що мають дану книгу та їх розміщення на карті. Дана сторінка зображена на рисунку 2.19.

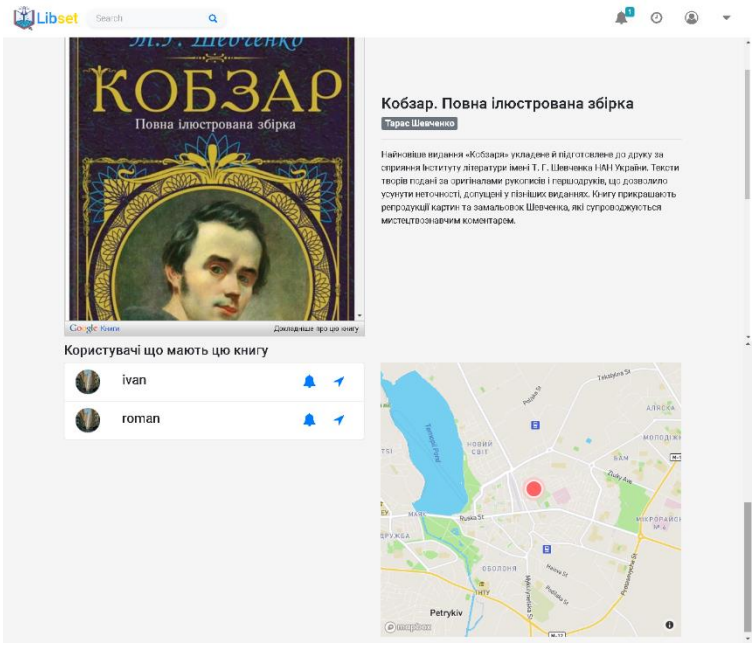


Рисунок 2.19 – Сторінка книги “Кобзар”

Для повідомлення користувачу про цікавість до книги необхідно натиснути на кнопку із значком «дзвінка». Після цього у користувача появиться відповідне сповіщення (рис. 2.20). При натисненні на дане сповіщення відкриється форма для вибору одного із наступних пунктів:

- відмовитись;
- домовитись про зустріч (надати контакти);
- видати книгу (у випадку якщо це безпосередньо бібліотека).

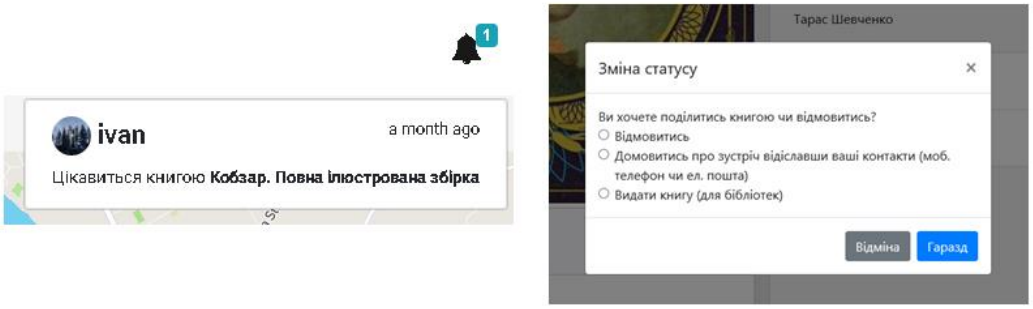


Рисунок 2.20 – Форма зміни статусу угоди

Після підтвердження угоди між користувачами час до повернення книги можна переглянути на сторінці угод. Для цього необхідно натиснути у навігаційному меню на кнопку із значком «годинника». На даній сторінці відображатимуться угоди виданих та взятих книг. В кожній угоді відображатиметься користувач з яким укладено угоду, книгу та час до завершення угоди. Вигляд даної сторінки наведено на рисунку 2.21.

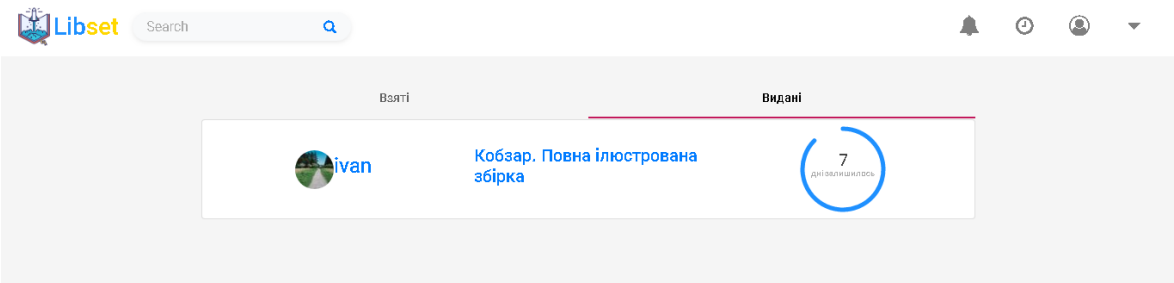


Рисунок 2.21 – Сторінка угод

# 3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

## 3.1 Планування стадій та етапів проєктування програмного забезпечення

Відповідно до ст. 1 Закону України «Про авторське право і суміжні права» від 04.11.2018 № 3792, комп'ютерна програма – це набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи у будь-якому іншому вигляді, виражених у формі, придатній для зчитування комп'ютером, які приводять його в дію для досягнення певної мети або результату. В такому самому значенні розглядають цей термін і з позиції податкового обліку (Податковий кодекс України). Статтею 8 ЗУ «Про авторське право і суміжні права» передбачено, що комп'ютерні програми є об'єктом авторського права.

Програмування – це процес створення (розробки) програми, який може бути представлений послідовністю таких кроків:

1) формулювання вимог до програми та розробка алгоритму;

2) кодування (запис алгоритму на вибраній мові програмування);

3) відлагодження;

4) тестування;

5) доопрацювання програми;

6) розробка довідкової системи;

7) запис інсталяційного CD/DVD-диска.

Проєктування програмного забезпечення (далі – ПЗ) складається з низки послідовних, цілеспрямованих, взаємозв'язаних та взаємообумовлених етапів, на які можна поділити весь часовий відрізок, що відводиться на розробку проєкту (табл. 3.1).

Процедурний підхід традиційно для розробки ПЗ, в основі якого лежать алгоритми, процедури і функції, передбачає розробку ПЗ як

монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію.

Об’єктно-орієнтований підхід, що ґрунтується на основі об’єктів певних класів, які описують визначену область, поведінку (методи) та володіють властивостями (атрибутами), зорієнтовані на варіанти використання та покомпонентний процес розробки.

Стадії теоретичної розробки, практичної реалізації та доробки всього комплексу програмного забезпечення описані з точки зору специфіки об’єктно-орієнтованого підходу із врахування ЖЦ розробки ПЗ, представлені із уточненням фахівців, залучених у кожний робочий процес, та артефактів (табл. 3.2).

Під час створення ПЗ за об’єктно-орієнтованим підходом необхідно залучити більшу кількість фахівців, вищу увагу приділити покроковій розробці, що, можливо, збільшить робочий час і витрати. Однак, цей підхід значно скорочує витрати на подальший супровід і модернізацію, що приводить до зменшення загальних витрат.

Таблиця 3.1

Перелік стадій та етапів і робіт по розробці проєкту

| Найменування | | Вид роботи | |
|---|---|---|---|
| стадії | етапу | шифр | зміст роботи |
| 1 Підготовча стадія | 1.1 Вивчення стану предметної області | 1.1.1 | Дослідження проблеми |
| | | 1.1.2 | Вибір платформи для реалізації проєкту |
| | | 1.1.3 | Вивчення та аналіз аналогічних розробок |
| | | 1.1.4 | Економічне обґрунтування доцільності виконання проєкту |
| | 1.2 Розробка технічного завдання | 1.2.1 | Складання ТЗ |
| | | 1.2.2 | Узгодження ТЗ із зацікавленими сторонами |
| | | 1.2.3 | Складання плану та розрахунок розробки |
| 2 Технічна пропозиція | 2.1 Аналіз ТЗ та техніко-економічне обґрунтування проєкту | 2.1.1 | Доведення техніко-економічного обґрунтування |
| | | 2.1.2 | Аналіз ТЗ та визначення пріоритетних аспектів розробки |
| | | 2.1.3 | Доведення та уточнення загального обсягу робіт, строків виконання та витрат |
| 3 Теоретична | 3.1 Теоретичне вивчення задачі | 3.1.1 | Дослідження технічних особливостей інформаційної системи |

| | | | |
|---|---|---|---|
| розробка | | 3.1.2 | Визначення переліку технологій, які використовуватимуться при розробці та мови програмування |

Продовження таблиці 3.1

| Найменування | | Вид роботи | |
|---|---|---|---|
| стадії | етапу | шифр | зміст роботи |
| 3 Теоретична розробка | 3.1 Теоретичне вивчення задачі | 3.1.3 | Визначення форматів даних та запитів і їх узгодження із розробниками проєкту |
| | | 3.1.4 | Розробка алгоритмів роботи програми на високому рівні |
| | | 3.1.5 | Розробка структури систем забезпечення та схеми взаємодії її компонент |
| 4 Практична реалізація | 4.1 Реалізація окремих модулів ПЗ | 4.1.1 | Розробка алгоритмів роботи програми на низькому рівні |
| | | 4.1.2 | Розробка окремих класів та компоновка їх у модулі |
| | | 4.1.3 | Розробка графічного користувацького інтерфейсу для системи |
| | 4.2 Відладка ПЗ | 4.2.1 | Автономна відладка окремих модулів системи |
| | | 4.2.2 | Комплексна відладка ПЗ системи |
| | 4.3 Розробка субмодуля | 4.3.1 | Розробка структурної схеми субмодуля |
| | | 4.3.2 | Розробка функціональної схеми субмодуля |
| | | 4.3.3 | Розробка алгоритму функціонування субмодуля |
| | | 4.3.4 | Обрунтування вибору елементного базису та розробка схеми |
| 5 Доробка всього комплексу | 5.1 Тестування всієї системи | 5.1.1 | Тестування системи у реальних умовах |
| | | 5.1.2 | Доробка систем із урахуванням результатів тестування |
| | 5.2 Підготовка документації по системі | 5.2.1 | Підготовка звіту про розробку системи |
| | | 5.2.2 | Підготовка технічної документації |
| | | 5.2.3 | Розробка довідкової системи, допоміжної і супроводжувальної документації, запис CD/DVD диска |
| 6 Заключ-на стадія | 6.1 Ознайомлення зацікавлених осіб з проєктом | 6.1.1 | Підготовка презентації |
| | | 6.1.2 | Демонстрація системи |
| | | 6.1.3 | Навчання осіб роботи з системою |

Таблиця 3.2

Робочі процеси життєвого циклу розробки ПЗ

| Робочі процеси | Діяльності | Співробітники | Артефакти |
|---|---|---|---|
| РП «Визначення вимог» | Ідентифікація актантів (А) і варіантів використання (ВВ), Модель ВВ<br>Описи (формалізація) ВВ і сценаріїв реалізації, Визначення пріоритетності ВВ<br>Створення прототипів інтерфейсів користувача (ІК) | Системний аналітик<br>Специфікатор ВВ | Модель ВВ<br>Актанти<br>Глосарій<br>ВВ<br>Прототип ІК |

Продовження таблиці 3.2

| | | | |
|---|---|---|---|
| РП «Проєктування» | Проєктування архітектури, Визначення вузлів та мережевих конфігурацій Визначення систем та їх інтерфейсів Визначення підсистем та зв'язків між ними Визначення інтерфейсів підсистем Визначення архітектурно значущих класів та класів проєктування, Визначення загальних механізмів проєктування Проєктування ВВ Проєктування класів Визначення вимог до реалізації | Архітектор Інженер з ВВ | Модель проєктування Модель розміщення Опис архітектури Проєкти реквізитів ВВ Класи про-ектування |
| РП «Реалізація» | Модель реалізації, Компоненти, Інтерфейси компонентів, Опис архітектури План збирання системи, Реалізація архітектури, Ітеративна реалізація класів Проєктування з генерацією програмного коду, Збірка системи, Планування білдів Реалізація білдів і системи в цілому | Системний інтегратор Інженер-програміст | Модель реалізації Опис архітектури План збірки Компоненти Підсистеми реалізації Інтерфейси |
| РП «Тестування» | Модель тестування Тестові приклади Процедура тестування Тестові компоненти План і оцінка тестування Розробка тестів Тестування цілісності Тестування системи Оцінка результатів тестування | Інженер з тестування Системний тестер | Модель тестування Тестові приклади План тестування Оцінювання тестів |

Перед безпосереднім виконанням роботи, треба скласти технічне завдання на розробку ПЗ, яке є основним документом, що регулює подальшу роботу, та містить точний опис необхідних функцій програми, інтерфейс, технології та ін. Вартість складання технічного завдання переважно становить до 15 % від планованої вартості розробки і становить від 300 до 6000 гривень (за даними софтверних компаній). Роботу зі складання технічного завдання веде керівник проєкту разом із програмістами, консультуючись при тому із замовником. Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях техпроцесу звести у таблицю (табл. 3.3).

Таблиця 3.3

Результати тривалості та трудозатратності розробки і реалізації програмного проєкту

| Шифр роботи | Найменування роботи | Виконавець, посада, спеціальність | К-сть виконавців | Тривалість виконання роботи, дні |
|---|---|---|---|---|
| 1.1.1 - 1.1.3 | Дослідження предметної області, вибір середовища для реалізації проєкту, вивчення та аналіз аналогічних розробок | Керівник проєкту | 2 | 7 |
| | | Інженер-програміст | | |
| 1.1.4 - 1.2.3 | Економічне обґрунтування доцільності виконання проєкту, складання ТЗ, узгодження ТЗ із зацікавленими сторонами, складання плану та розрахунок розробки | Керівник проєкту | 2 | 5 |
| | | Інженер-програміст | | |
| 2.1.1 - 2.1.3 | Доведення техніко-економічного обґрунтування, аналіз ТЗ та визначення пріоритетних аспектів розробки, доведення та уточнення загального обсягу робіт, строків виконання та витрат | Керівник проєкту | 2 | 4 |
| | | Інженер-програміст | | |
| 3.1.1 - 3.1.6 | Теоретична розробка процедурний підхід | Інженер-програміст, системний аналітик, архітектор | 3 | 30 |
| 3.1.1 - 3.1.6 | Теоретична розробка об’єктно-орієнтований підхід | Системний аналітик, архітектор, специфікатор ВВ, інженер з ВВ | 4 | 42 |
| 3.1.1 - 3.1.6 | Практична реалізація процедурний підхід | Інженер-програміст | 1 | 18 |
| 4.2.2 | Практична реалізація об’єктно-орієнтований підхід | Системний інтегратор, інженер-програміст | 2 | 38 |
| | Тестування програмного забезпечення | Тестувальник | 1 | 19 |
| 5.1.1 - 5.2.3 | Доробка всього комплексу програмного забезпечення | Інженер-програміст | 1 | 2 |
| 6.1.1 - 6.1.3 | Заключна стадія | Керівник проєкту | 2 | 2 |
| | | Інженер-програм. | | |

Як для процедурного, так і для об’єктно-орієнтованого підходу, підготовча і остаточна стадії та технічна пропозиція будуть виконуватися однаково, теоретична розробка і практична реалізація для об’єктно-орієнтованого підходу вимагатимуть залучення більшої кількості ІТ спеціалістів, що, можливо, збільшить тривалість виконання роботи на початковій стадії проєктування, але значно зменшить час розробки програмного коду.

Загальна кількість часу на виконання операцій технічного процесу в даному випадку становитиме 125 робочих днів для процедурного підходу і 90 робочих днів для об’єктно-орієнтованого підходу.

## 3.2 Розрахунок витрат на реалізацію проєкту та оцінка економічної ефективності

Оцінка вартості комп’ютерних програм базується на витратному підході: використанні первісної вартості об’єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. При цьому для оцінки нематеріальних активів використовують міжнародні стандарти оцінки розрахунку вартості об’єктів інтелектуальної власності, розроблені TIAVIS (The International Assets Vabulation Standarts Commitie).

Враховуючи залежність якості кінцевого продукту від кваліфікації програмістів, розробники часто йдуть на додаткові заходи їх стимулювання, які найчастіше втілюються у: певній системі преміювання, акційні відрахування за кожну продану одиницю програми.

Витрати на реалізацію проєкту – це витрати на розробку та реалізацію, що в перспективі будуть покриті доходом від реалізації споживачам. При поширенні проєкту розмір доходів буде значно більшим за витрати.

Разом з тим до створення ПЗ можуть бути залучені також позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох

випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором підряду, то все залежить від того, чи зареєстрований виконавець підприємцем. Також необхідно врахувати витрати на податкування відповідно.

Важливим етапом розробки ПЗ є його тестування, яке виконується тестувальником за певну винагороду. Якщо тестувальники не перебувають з підприємством у трудових відносинах, оплата виконується на основі договору підряду на виконання робіт з тестування ПЗ.

Законодавство України дозволяє розповсюдження на території України і прокат комп'ютерних програм та баз даних лише за умови їх маркування контрольними марками [12]. Контрольні марки видаються центральним органом виконавчої влади, що реалізує державну політику у сфері інтелектуальної власності. Вартість однієї такої марки складає 1-2 % неоподаткованого мінімуму доходу громадян, тобто 17 коп. (п. 19 Порядок № 1555) і її доцільно включати до виробничої собівартості продукції.

Вважатимемо, що усі програмісти працюють у штаті підприємства-розробника і їм встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлено у табл. 3.4.

Таблиця 3.4

Розрахункова вартість технологічного процесу

| Посада | Місячний оклад | Денна зар. плата | Об'єктно-орієнтований підхід | | Процедурний підхід | |
|---|---|---|---|---|---|---|
| | | | Днів | Сума | Днів | Сума |
| Керівник проєкту | 8 500,00 | 400,00 | 20 | 8 000,00 | 10 | 4 000,00 |
| Інженер-програміст | 7 540,00 | 390,00 | 55 | 21 450,00 | 15 | 5 850,00 |
| Тестувальник | 5 160,00 | 250,00 | 25 | 6 250,00 | 5 | 1 250,00 |
| Архітектор | 8 300,00 | 400,00 | 25 | 10 000,00 | 60 | 24 000,00 |
| Всього | | | 125 | 45 700,00 | 90 | 35 100,00 |

Слід зауважити, що невідповідність закону «авторське право і суміжні права» положенням податкового кодексу може значно ускладнити процедуру визнання підприємця як суб'єкта сектора програмного забезпечення.

– Розрахунок вартості технологічно процесу.

– Витрати на основну заробітну плату працівників. Це сума часткових заробітних плат за кожну роботу, і вона приведена в табл. 3.4.

Тому отримаємо:

$$ЗП_{осн1} = 45\,700{,}00 \cdot ЗП_{осн2} = 35\,100{,}00 \text{ грн.}$$

– Додаткова заробітна плата обчислюється за формулою

$$ЗП_{дод} = 0{,}2 \cdot ЗП_{осн}. \quad (3.1)$$

$$ЗП_{дод1} = 0{,}2 \cdot 45\,700{,}00 = 9\,140{,}00 \text{ грн},$$
$$ЗП_{дод2} = 0{,}2 \cdot 35\,100{,}00 = 7\,020{,}00 \text{ грн},$$

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП = ЗП_{осн} + ЗП_{дод}, \quad (3.2)$$

$$ФЗП_1 = 45\,700{,}00 + 9\,140{,}00 = 58\,840{,}00 \text{ грн},$$
$$ФЗП_2 = 35\,100{,}00 + 7\,020{,}00 = 42\,120{,}00 \text{ грн},$$

З цієї суми утримуються обов'язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 22 % від суми нарахованої заробітної плати [13] та податок на доходи фізичних осіб, який складає 18 % від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов'язкове соціальне страхування та податкової соціальної пільги [14]. Також 1,5 % військового збору.

Таким чином обов’язкові відрахування на заробітну плату для працівників складають:

– утримання єдиного соціального внеску

$$Відр_{ЄСВ1} = 0{,}22 \cdot ФЗП_1 = 0{,}22 \cdot 58\,840{,}00 = 12\,944{,}80 \text{ грн},$$

$$Відр_{ЄСВ2} = 0{,}22 \cdot ФЗП_2 = 0{,}22 \cdot 42\,120{,}00 = 9\,266{,}40 \text{ грн},$$

– податок на доходи фізичних осіб:

$$Відр_{ПДФО1} = 0{,}18 \cdot (ФЗП_1 - Відр_{ЄСВ1}),$$

$$Відр_{ПДФО1} = 0{,}18 \cdot (58\,840{,}00 - 12\,944{,}80) = 8\,261{,}13 \text{ грн},$$

$$Відр_{ПДФО2} = 0{,}18 \cdot (ФЗП_2 - Відр_{ЄСВ2}),$$

$$Відр_{ПДФО2} = 0{,}18 \cdot (42\,120{,}00 - 9\,266{,}40) = 5\,913{,}64 \text{ грн},$$

$$Відр_1 = Відр_{ЄСВ1} + Відр_{ПДФО1} = 12\,944{,}80 + 8\,261{,}13 = 21\,206{,}44 \text{ грн},$$

$$Відр_2 = Відр_{ЄСВ2} + Відр_{ПДФО2} = 9\,266{,}40 + 5\,913{,}64 = 15\,180{,}04 \text{ грн}.$$

– Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3 %, для підприємців розмір єдиного внеску залежно від класу професійного ризику виробництва становить від 36,76 % до 49,70 %. Зокрема, видання програмного забезпечення – 36,77 % [12, 15].

Нарахування на фонд оплати праці (ФОП) здійснюється за формулою:

$$ФОП_{ЄСВ} = 0{,}3677 \cdot ФЗП_{1,2}. \qquad (3.3)$$

$$ФОП_{ЄСВ1} = 0{,}3677 \cdot 58\,840{,}00 = 21\,635{,}46 \text{ грн},$$

$$ФОП_{ЄСВ2} = 0{,}3677 \cdot 42\,120{,}00 = 15\,487{,}52 \text{ грн},$$

Всього витрат по заробітній платі:

$$ВЗП_1 = ЗП_1 + ФОП_{ЄСВ1} = 58\,840{,}00 + 21\,635{,}46 = 80\,475{,}46 \text{ грн},$$

$$ВЗП_2 = ЗП_2 + ФОП_{ЄСВ2} = 42\,120{,}00 + 15\,487{,}52 = 57\,607{,}52 \text{ грн}.$$

– Розрахунок матеріальних витрат.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (див. табл. 3.5 та формулу 3.4).

$$Мв_i = q_i \cdot p_i, \tag{3.4}$$

де $q_i$ – кількість витраченого матеріалу $i$-го виду;

$p_i$ – ціна матеріалу $i$-го виду.

Звідси, загальні матеріальні витрати можна визначити за формулою 3.6:

$$З_{м.в.} = \sum Мв_i \tag{3.5}$$

Проведені розрахунки занесені у таблицю 3.5.

Таблиця 3.5

Матеріальні витрати

| Найменування матеріальних ресурсів | Од. виміру | Фактично витрачено матеріалів | Ціна 1-ці, грн. | Загальна сума витрат, грн. |
|---|---|---|---|---|
| DVD диски | шт. | 15 | 15,00 | 225,00 |
| Флешка | шт. | 3 | 170,00 | 510,00 |
| Папір для друку | листів | 700 | 0,25 | 175,00 |
| Чорнила для принтера | шт. | 1 | 163,00 | 163,00 |
| Всього | | | | 1 073,00 |

Отже, в результаті проведених розрахунків по матеріальних витратах загальна сума становить 1 073,00 гривні.

– Розрахунок витрат на електроенергію.

Витрати на електроенергію одиниці обладнання визначаються за формулою:

$$З_e = W \cdot T \cdot S, \tag{3.6}$$

де $W$ – необхідна потужність, кВт; $T$ – кількість годин роботи обладнання; $S$ – вартість кіловат-години електроенергії, $S$ = 2,50 грн. 1кВт/год (спожитий до 100 кВт/год електроенергії на місяць).

$$З_{е1} = W \cdot T \cdot S = 0{,}6 \cdot 1000 \cdot 2{,}50 = 1500 \text{ грн},$$

$$З_{е2} = W \cdot T \cdot S = 0{,}6 \cdot 720 \cdot 2{,}50 = 1080 \text{ грн}.$$

– Розрахунок суми амортизаційних відрахувань.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається за формулою 3.7:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}}, \quad (3.7)$$

де $C_Б$ – балансова вартість обладнання, грн; $N_A$ – норма амортизаційних відрахувань в рік, %; $T_{ГОД}$ – річний робочий фонд часу, год; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год.

Таблиця 3.6

Перелік обладнання необхідного для розробки комп'ютерної програми.

| Найменування | Кількість,шт. | Ціна за одиницю продукту, грн. | Сума,грн. |
|---|---|---|---|
| Комп'ютер | 5 | 16 000,00 | 80 000,00 |
| Принтер | 1 | 2 500,00 | 2 500,00 |
| Середовище розробки | 1 | 8 900,00 | 8 900,00 |
| Середовище створення 3D моделей | 1 | 9 700,00 | 9 700,00 |
| Всього (вартістю більше 1000 грн.) | 37 100,00 | | |
| Всього витрачено на амортизацію | | 10 993,27 | 7 678,78 |
| Всього | | 48 093,27 | 44 778,78 |

$$A_1 = \frac{37100 \cdot 0{,}6 \cdot 1005}{2035} = 10993{,}27 \text{ грн}, \quad (3.8)$$

$$A_2 = \frac{37100 \cdot 0{,}6 \cdot 702}{2035} = 7\ 678{,}78 \text{ грн}, \quad (3.9)$$

– Накладні витрати.

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 25–65 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн1,2}. \quad (3.10)$$

$НВ_1 = 0{,}5 \cdot 45\ 700{,}00 = 22\ 850$ грн, $НВ_1 = 0{,}5 \cdot 35\ 100 = 17\ 550$ грн.

Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни. Програмні продукти є дуже специфічним товаром. Основні витрати припадають на розробку програми і супровід, тоді як процес тиражування є, зазвичай, порівняно нескладною і недорогою процедурою копіювання. Отже, цей товар не має, власне, ринкової вартості, формованої з урахуванням суспільно-необхідних витрат праці. Ціна на програмний продукт часто може розраховуватися з урахуванням роялті і складається з щорічних відрахувань доходу споживачів на протязі періоду дії угоди. Розмір роялті встановлюється за угодою сторін в дуже широкому діапазоні.

Загалом, ціна програмного продукту залежить від висунутих функціональних вимог, технологій, які передбачається використовувати, обсягів робіт, терміну виконання, передачі майнових авторських прав на програмний продукт.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво:

$$Св_1 = 205\ 670{,}55 \text{ грн}, \quad Св_1 = 176\ 209{,}45 \text{ грн}.$$

Приймемо прибуток на рівні 35 %. Для нових інноваційних продуктів, що користуються високим попитом на ринку, ринкову вартість *Вр* можна встановити вищу.

Отже, вартість розробленого програмного забезпечення:

$$Вр_1 = Св_1 + 0{,}3 \cdot Св_1 = 205\ 670{,}55 + 0{,}3 \cdot 205\ 670{,}55 = 267\ 371{,}71 \text{ грн},$$

$$Вр_2 = Св_2 + 0{,}3 \cdot Св_2 = 176\ 209{,}45 + 0{,}3 \cdot 176\ 209{,}45 = 229\ 072{,}28 \text{ грн}.$$

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (*Е*) полягає у відношенні результату виробництва до затрачених ресурсів (формула 3.11).

$$E = \frac{П}{Св}, \tag{3.11}$$

де $П$ – прибуток, $П = В - Св$ ;

$Св$ – собівартість.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Отже,

$$E_1 = (267\ 371{,}71 - 205\ 670{,}55) / 205\ 670{,}55 = 0{,}30,$$

$$E_1 = (229\ 072{,}28 - 176\ 209{,}45) / 176\ 209{,}45 = 0{,}30.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_{ок}$):

$$T_{ок} = \frac{1}{E}. \qquad (3.12)$$

У нашому випадку $T_{ок} = T_{ок1} = T_{ок2} = 1/0{,}30 = 3{,}33$ років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Даний розрахунок виконаний у розрахунку на 1 екземпляр програмного продукту без врахування його тиражування.

## 3.3 Визначення витрат на супровід і модернізацію програмного продукту та уточнений аналіз ефективності вкладених інвестицій

Ключові питання супроводу ПЗ – це управлінські, вимірювальні і вартісні. Відомий фахівець в області ПЗ Дж. Леман (1970 р.) запропонував розглядати супровід як еволюційну розробку програмних систем, оскільки здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. За різними оцінками фахівців витрати на підтримку і модернізацію програмного забезпечення, написаного процедурним методом, становлять більше 50 % витрат на його створення, а інколи і перевищують доходи від його реалізації. Об’єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми довжиною до декількох десятків тисяч рядків, що значно скорочує подальші витрати на супровід і модернізацію.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за процедурним методом 60 % від початкових витрат, а за об’єктно-орієнтованим – 20 %.

Собівартість модернізації:

$$СвМ_1 = 0{,}2 \cdot Св_1 = 0{,}2 \cdot 205\,670{,}55 = 41\,134{,}11 \text{ грн},$$

$$СвМ_2 = 0{,}6 \cdot Св_2 = 0{,}6 \cdot 176\ 209{,}45 = 105\ 725{,}67 \text{ грн.}$$

Для споживача вартість модернізації:

$$M_1 = 0{,}2 \cdot B_1 = 0{,}2 \cdot 267\ 371{,}71 = 53\ 474{,}34 \text{ грн,}$$

$$M_2 = 0{,}6 \cdot B_2 = 0{,}6 \cdot 229\ 072{,}28 = 137\ 443{,}37 \text{ грн.}$$

Таким чином, уже після першої модернізації, загальні витрати на створення і супровід ПЗ для виробника за об'єктно-орієнтованим методом менші, ніж за процедурним.

$$ЗВ_{1(вир)} = 205\ 670{,}55 + 41\ 134{,}11 = 246\ 804{,}66 \text{ грн,}$$

$$ЗВ_{2(вир)} = 176\ 209{,}45 + 105\ 725{,}67 = 281\ 935{,}12 \text{ грн.}$$

Як і для споживача:

$$ЗВ_1 = 267\ 371{,}71 + 53\ 474{,}34 = 320\ 846{,}05 \text{ грн,}$$

$$ЗВ_2 = 229\ 072{,}28 + 137\ 443{,}37 = 366\ 515{,}65 \text{ грн.}$$

Річна економія витрат за всіма можливими напрямами і додатковими витратами, пов'язаними з супроводом і тільки одноразовою модернізацією (у розрахунку на одиницю продукції) при об'єктно-орієнтованому методі порівняно із процедурним:

$$\Delta C_{(вир)} = ЗВ_{2(вир)} - ЗВ_{1(вир)} = 281\ 935{,}12 - 246\ 804{,}66 = 35\ 130{,}46 \text{ грн,}$$

$$\Delta C = ЗВ_2 - ЗВ_1 = 366\ 515{,}65 - 320\ 846{,}05 = 45\ 669{,}6 \text{ грн.}$$

При постійному супроводі і модернізації програмного забезпечення різниця між загальними витратами за двома варіантами значно збільшується.

Чистий приведений дохід (ЧПД) визначається як різниця між сукупними доходами (сукупний грошовий потік) і сукупними витратами (сукупними інвестиціями) взятими за весь період життя інвестицій і дисконтова ними в кожному році на фактор часу. Дисконтування являє собою визначення вартості майбутніх грошових потоків у теперішній момент часу. Ефективним вважається той проєкт, який забезпечує максимум ЧПД, оскільки при цьому досягається найвища дохідність власників інвестицій.

Визначення ЧПД відбувається за формулою

$$ЧПД = \sum_{i=1}^{t} ГП_i \alpha_{TBi} - \sum_{i=1}^{t} ІК_i \alpha_{TBi}\,. \qquad (3.13)$$

де $ГП_i$ – грошовий потік *i*-го розрахункового року;

$ІК_i$ – сума інвестицій *i*-го розрахункового року;

$\alpha_{TBi}$ – коефіцієнт дисконтування (коефіцієнт приведення інвестицій і грошового потоку до теперішньої вартості).

Грошовий потік – це фінансовий показник, що характеризує ефект інвестицій у вигляді грошових коштів, які повертаються інвестору.

Коефіцієнт дисконтування показує, яку величину грошових коштів ми отримаємо з урахуванням фактору часу та ризиків. Він дозволяє перетворити майбутню вартість у вартість на даний момент.

Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$\alpha = (\frac{1}{1+i^n}), \qquad (3.14)$$

де *i* – ставка дисконтування або норма дисконту, *i* = 0,2; *n* – час або кількість періодів (років), протягом якого планується отримання доходу.

$$\alpha_0 = 1, \qquad \alpha_1 = (\frac{1}{1+0,2}) = 0,83.$$

Вважатимемо, що обидва програмних продукти однаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми однаково вплинуть на розмір його додаткових доходів на вкладений капітал. Тому приймемо цю величину за постійну, а порівняння дохідності двох проєктів проведемо тільки за витратами.

$$ЧПД_1' = ГП + 0,83 \cdot ГП - 267\,371,71 - 0,83 \cdot 53\,474,34 = 1,83ГП - 311\,755,41 \text{ грн.}$$

$$ЧПД_2' = ГП + 0{,}83 \cdot ГП - 229\,072{,}28 - 0{,}83 \cdot 137\,443{,}37 = 1{,}83ГП - 343\,150{,}28$$ грн.

Чим менші витрати, тим більша дохідність проєкту.

Економія витрат у випадку придбання, супроводу і одноразової модернізації програмного продукту, створеного за об'єктно-орієнтованим підходом, становить 31 394,87 грн.

Отже, сучасну комп'ютерну програму доцільно виконувати по об'єктно-орієнтованій парадигмі.

Усі результати розрахунків, приведених у даному розділі, зведені у таблицю 3.7.

Таблиця 3.7

Загальні витрати на впровадження програмного продукту

| | Об'єктно-орієнтований підхід | Процедурний підхід |
|---|---|---|
| Зарплата основна, грн | 45 700,00 | 35 100,00 |
| Зарплата додаткова, грн. | 9 140,00 | 7 020,00 |
| Фонд заробітної плати, грн. | 58 840,00 | 42 120,00 |
| Відрахування на ФОП, грн. | 12 944,80 | 9 266,40 |
| *Разом на оплату праці, грн.* | 80 475,46 | 57 607,52 |
| Матеріальні витрати, грн. | 1 073,00 | 1 073,00 |
| Електроенергія, грн. | 1500 | 1080 |
| Амортизація, грн. | 10 993,27 | 7 678,78 |
| Накладні витрати, грн. | 22 850,00 | 17 550,00 |
| *Разом на ін. витрати, грн.* | 33 843,27 | 26 680,86 |
| Собівартість | 205 670,55 | 176 209,45 |
| Прибуток | 61 701,16 | 52 862,83 |
| Вартість розробленого ПЗ | 267 371,71 | 229 072,28 |
| Економічна ефективність | 0,30 | 0,30 |
| Термін окупності, років | 3,33 | 3,33 |
| Собівартість модернізації | 41 134,11 | 105 725,67 |
| Супровід | 53 474,34 | 137 443,37 |
| Загальні витрати на розробку | 246 804,66 | 281 935,12 |
| Порівняльна економія витрат (для виробника) | 35 130,46 | |
| Загальні витрати (для споживача на придбання програмного прод.) | 320 846,05 | 366 515,65 |
| Порівняльна економія витрат (для споживача) | 45 669,6 | |
| Дохідність проєкту для споживача за витратною частиною | – 311 755,41 | – 343 150,28 |
| Економія | 31 394,87 | |

Після проведеного техніко-економічного обґрунтування програмного продукту чітко простежується той фактор, що створення програмного продукту за об’єктно-орієнтованим підходом є значно ефективнішим, як за витратами часу так і за матеріальними витратами, економія витрат за цим методом для виробника становить 35 130,46 грн, а економія витрат при купівлі для кінцевого споживача становить 45 669,6 грн, так і в подальшому при підтримці та модернізації програмного продукту спостерігаються значна економія витрат порівняно з процедурним підходом [16].

Як було згадано даний програмний продукт окупить себе вже після третього року використання, тобто почне приносити прибутки.

# 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

## 4.1 Охорона праці

Тема магістерської роботи пов’язана із розробкою програмної системи для перегляду та зберігання інформації в бібліотеках. Приміщення, в якому проводилася розробка, маючи площу 30м$^2$ та об’єм 92м$^3$, містить 4 комп’ютеризовані робочі місця. Для зберігання документів використовуються дві шафи. Приміщення містить два вікна та одні двері, які розташовані по середині, між робочими місцями. Детальний план приміщення наведено на рисунку 4.1.

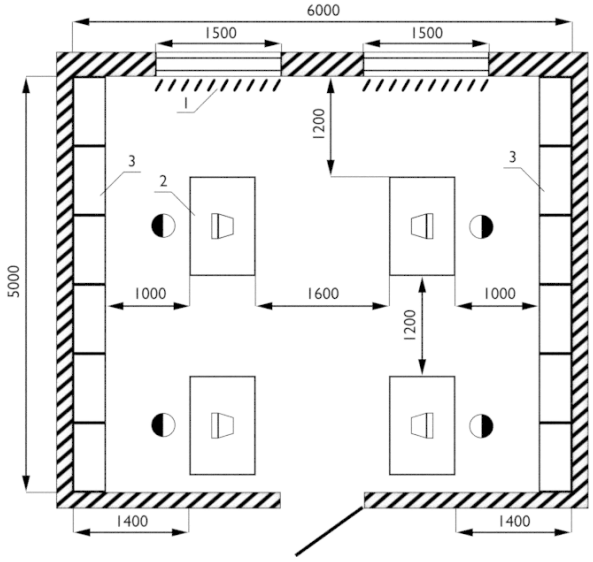


Рисунок 4.1 – План приміщення

Умовні позначки на рисунку: 1 – комп’ютеризоване робоче місце з ВДТ; 2 – сонцезахисні жалюзі; 3 – шафи для зберігання.

Розглянемо відповідність характеристик комп’ютеризованого робочого місця розробника нормативним. Площа, на якій розташовується одне робоче місце з ВДТ, повинна становити не менше 6,0 кв м. В нашому випадку площа становить 7,5 кв м.

При розміщенні робочих столів з персональними комп'ютерами слід дотримувати:

- відстань між бічними поверхнями персональних комп'ютерів 1,2 м;
- відстань від тильної поверхні одного персонального комп'ютера до екрана іншого – 2,5 м;
- робочі місця з ВДТ розміщуються на відстані не менше 1 м від стіни зі світловими прорізами (вікнами);
- прохід між рядами робочих місць має бути не меншим за 1 м.

В даному випадку робочі місця розміщено так, що в поле зору робітника не може попасти екранна сторона дисплею, окрім його власного. Також відстань між бічними поверхнями, відстань від стін із світловими прорізами та проходи між рядами (рис. 4.1) задовольняють нормам.

Рекомендовані розміри столу для робочого місця з ВДТ становлять: висота – 725 мм, ширина – 600-1400 мм, глибина 800-1000 мм. В даному приміщенні використовуються робочі столи з розмірами: ширина – 1200 мм, глибина – 800 мм.

Комп'ютеризовані робочі місця розміщені рядами вздовж стіни з вікнами. Це дає змогу виключити дзеркальне відбиття на екрані ВДТ джерел природного світла (вікон) та потрапляння останніх у поле зору операторів, що погіршує їх зорову роботу.

Відповідно до ДСН 3.3.6.042-99 роботи, що виконуються користувачами ЕОМ, відносяться до легких фізичних робіт – категорії Іа. У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату.

Згідно НПАОП 0.00-7.15-18 приміщення, що розглядається, повинне мати природне і штучне освітлення.

Природне освітлення приміщення відбувається за системою однобічного бічного освітлення. Природне світло проникає у приміщення через два світлові віконні отвори, які мають регулювальні пристрої для відкривання. Також наявні штори (жалюзі) з можливістю захисту працюючих

від прямого попадання сонячних променів і регулювання рівня освітленості в приміщенні. Вікна приміщення орієнтовані на північний схід. Оскільки будинок розташований у відносній віддаленості від прилеглих будівель, то які-небудь перешкоди природному освітленню розглянутого приміщення відсутні. Всередині приміщення стіни обклеєні світлими шпалерами, стеля побілена, у якості підлогового покриття використаний лінолеум світло-жовтого кольору.

У приміщенні присутні внутрішні джерела постійного шуму – вентилятори блоків ЕОМ, дисководи. Зовнішніми джерелами шуму і вібрації в приміщенні є проїжджаючі транспортні засоби. Наявність постійного шуму в робочій зоні приводить до розладу центральної нервової системи і до таких захворювань як неврози, однак фактичний обмірюваний рівень шуму в робочій зоні склав 37 дБА, що задовольняє нормативному рівню шуму (не повинний перевищувати 50 дБА згідно з ДСН 3.3.6.037- 99).

Джерелом електромагнітного випромінювання в сучасному офісі є візуальні дисплейні термінали. Нормування електромагнітного випромінювання ВДТ здійснюється згідно положень ДСанПіН 3.3.2-007-98.

Приміщення відноситься до зони П-IIа згідно з ДНАОП 0.00-1.31-99 і до категорії пожежної небезпеки В. Горючі рідини, пил та волокна у приміщенні не використовуються і не виділяються.

Ймовірними причинами виникнення пожежі можуть бути несправність електрообладнання, короткі замикання внаслідок виходу з ладу електроустаткування, порушення правил протипожежної безпеки тощо.

Для своєчасного попередження пожеж та підвищення оперативності реагування у приміщенні використовується такий комплекс заходів:

- обов’язковий інструктаж персоналу з питань охорони праці;
- зокрема, правила пожежної безпеки у приміщеннях з ЕОМ;
- заборона використання відкритого вогню у приміщенні;
- наявність системи автоматичної пожежної сигналізації з димовими пожежними оповіщувачами;

- наявність шляхів евакуації при виникненні пожежі;
- розміщення схеми евакуації людей при пожежі і ознайомлення з нею персоналу.

В даному розділі було проаналізовано основні вимоги, які ставляться до приміщення в цілому, робочого місця користувача комп’ютером, освітлення, шуму, електромагнітних випромінювань, пожежної безпеки.

Можна стверджувати, що при розробці даної програмної системи було дотримано усіх необхідних вимог та норм охорони праці.

# 5 ЗАБЕЗПЕЧЕННЯ ЖИТТЄДІЯЛЬНОСТІ ПРИ РОБОТІ З ПК

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась би без використання комп'ютерної техніки. Через масовий характер робіт, що виконуються працівниками за допомогою комп'ютера, законодавством України чітко врегульовано норми та вимоги до використання комп'ютерної техніки на підприємстві [17].

Для забезпечення життєдіяльності людини при роботі з ПК необхідно визначити види небезпек та яким чином їх можна уникнути.

На користувачів під час роботи з комп'ютерною технікою можуть діяти такі види небезпек:

- ураження електричним струмом;
- енергетична небезпека (виникає через коротке замикання: опіки, електрична дуга, викид розплавленого металу);
- небезпека загоряння;
- термонебезпека (дія високих температур через нагрівання конструктивних елементів);
- механічна небезпека (травми через падіння, дію рухомих частин, поріз за гострі частини конструктивних елементів);
- небезпека випромінювання (дія звукового (акустичного), високочастотного, інфрачервоного, ультрафіолетового й іонізуючого випромінювання, а також видимого світла когерентної високої інтенсивності (лазерного випромінювання);
- хімічна небезпека (контакт із деякими хімікатами, які використовують для того, щоб обслуговувати обладнання, або від вдихання їх парів).

Заходи щодо усунення небезпеки ураження електричним струмом зводяться до правильного розміщення устаткування та електричних кабелів. Інші заходи щодо забезпечення електробезпеки, збігаються з загальними заходами пожежо- та електробезпеки.

В якості профілактичних заходів для забезпечення пожежної безпеки слід використовувати приховану електромережу, надійні розетки з пожежобезпечних матеріалів, силові мережі живлення устаткування виконувати кабелями, розрахованими на підключення в 3-5 разів більшого навантаження, включати й виключати живлення обладнання за допомогою штатних вимикачів. Треба регулярно робити очистку внутрішніх частин комп'ютерів, іншого устаткування від пилу, розташовувати комп'ютери на окремих неспалюваних столах. Для запобігання іскріння необхідно рідше встромляти і виймати штепсельні вилки з розеток [17].

Екран дисплея повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він розташований під кутом, то стає причиною сутулості. Відстань від дисплея до очей повинна трохи перевищувати звичну відстань між книгою та очима. Перед екраном монітора, особливо старих типів, повинен бути спеціальний захисний екран. При його відсутності треба сидіти на відстані витягнутої руки від монітора. Ще одним моментом, який стосується зору, є необхідність створення неоднорідного поля зору. Для цього можна розвісити на поверхнях (стінах) плакати та картини, виконані у спокійних тонах. Наприклад, пейзажі.

Важливою є форма спинки крісла, яка повинна повторювати форму спини. Висота крісла повинна бути такою, щоб користувач не почував тиску на куприк або стегна. Крісло бажано обладнати бильцями. Його потрібно встановити так, щоб не треба було тягтися до клавіатури. Періодично користувачу необхідно рухатися, вчасно змінювати положення тіла і робити перерви у роботі.

При напруженій роботі за комп'ютером щогодини необхідно робити перерву на 15 хвилин через кожну годину і треба займатися іншою справою.

Декілька разів на годину бажано виконувати серію легких вправ для розслаблення.

Наслідками регулярної роботи з комп'ютером без застосування захисних засобів можуть бути: захворювання органів зору (60% користувачів); хвороби серцево-судинної системи (20%); захворювання шлунково-кишкового тракту (10%); шкірні захворювання (5%); різноманітні пухлини.

Режим праці та відпочинку при роботі з персональною електронно-обчислювальною машиною (ПЕОМ) залежить від категорії трудової діяльності. Всі роботи з ПЕОМ ділять на три категорії. Перша - епізодичне зчитування і робота з інформацією не більше 2-х годин за 8-годинну робочу зміну. Друга - зчитування інформації або творча робота не більше 4-х годин за восьми годинну зміну. Третя - зчитування інформації або творча робота тривалістю більше 4-х годин за зміну [17].

Якщо у приміщенні експлуатується більше одного комп'ютера, то треба врахувати, що на користувача одного комп'ютера можуть впливати випромінювання від інших, в першу чергу бокових, а також і задньої стінки сусіднього дисплея. Тому необхіден захист спеціальними фільтрами і щоб користувач розміщався від бічних і задніх стінок інших дисплеїв на відстані не ближче одного метра.

Отже, щоб запобігти негативним впливам необхідно знати й небезпечні сторони самого комп'ютера і правила безпечної роботи, знати засоби запобігання небезпек. Вони пов'язані перед усім із загально відомими небезпечними факторами - поразками електричним струмом, пожежонебезпечністю.

Негативні фактори впливу на здоров'я. Всесвітня організація охорони здоров'я (ВООЗ) роботу з персональним комп'ютером віднесла до небезпечних, бо їй притаманний фактор постійно діючого стреса. З-за цього небезпеці піддаються всі життєво важливі органи людини, з'являється ризик виникнення серйозних хвороб.

Електромагнітні поля біля комп'ютера (особливо низькочастотні) негативно впливають на людину і в першу чергу на її центральну нервову систему, викликаючи головний біль, запаморочення, нудоту, депресію, безсоння, відсутність апетиту, виникнення синдрому стресу. Причому нервова система реагує навіть на короткі за тривалістю впливи слабких полів: змінюється гормональний стан організму, порушуються біоструми мозку. Це призводить до погіршення зору, ускладненню серцево-судинних захворювань, зниженню імунітету, виникають негативні впливи на плин вагітності [17].

Характерною рисою професії оператора ПК є статичний режим роботи: великий обсяг праці треба виконувати в сидячому положенні. При цьому більшість груп м'язів постійно напружені, що призводить до швидкої стомлюваності, сприяє розвитку фахових патологічних вигинів хребта: грудному гіперкифозу, сплощенню шийного лордозу і формуванню сколіозів. Неправильне розташування дисплеїв по висоті - занадто низьке або високе, під неправильним кутом - є головною причиною появи сутулості. Занадто високе розташування дисплея призводить до тривалої напруги шийного відділу хребта, що, зрештою, може призвести до розвитку остеохондрозу. Ненормальний стан хребта може стати причиною захворювання всього організму.

При тривалій та інтенсивній роботі за комп'ютером з'являється синдром комп'ютерного стресу (СКС), який проявляється головною біллю, запаленням очей, алергією, дратівливістю, млявістю і депресією, погіршенням зосередженості і працездатності.

Причинами різноманітних симптомів СКС є 5 основних чинників: неправильна робота очей і поза тіла, носіння невідповідних окулярів або контактних лінз, неправильна організація робочого місця, розподілення фізичних, розумових, візуальних навантажень, низький рівень візуальної підготовленості для роботи з комп'ютером.

Отже, щоб цьому запобігти потрібна висока комп’ютерно-інформаційна грамотність, і перед усім спеціалістів у будь-якій сфері діяльності - промисловій, науковій, навчальній.

Додатково, для збереження належного рівня здоров’я та професійної придатності робітників, рекомендується виділити на підприємстві окреме побутове приміщення для перепочинку працівників і зняття ними нервово-емоційного напруження, що виникає при роботі з комп’ютером.

# ВИСНОВКИ

В результаті виконаної роботи розроблено соціальну мережу для бібліотек, яка дозволяє створювати електронні бібліотеки, моніторинг виданих та взятих книг, публікація постів та організація подій, перегляд книги онлайн, відображення на карті бібліотек із книгою що цікавить. Система надає доступ до функціоналу на основі шифрованого ключа. Створено підхід для зручного додавання книг на базі Google Books API.

Для розробки серверної частини було застосовано платформу .NET з використанням каркасу ASP.NET Core 2.0. Реалізовано авторизацію на основі системи Asp Net Identity та використанням відкритого стандарту JSON Web Token. Клієнтська частина була розроблена на основі сучасного каркасу Angular 5, для стилізації графічного інтерфейсу було застосовано Bootstrap 4. Для роботи з картами було застосовано безкоштовну платформу MapBox. Для перегляду онлайн книг використано Google Ember Viewer.

До переваг розробленої системи можна віднести універсальність, тобто можливість її застосування не лише офіційним бібліотекам але й простим користувачам із власною колекцією книг, легкий спосіб додавання нових книг без заповнення довгих форм, схема бази даних яка забезпечує відсутність дублювання інформації. Використано сучасні технології та каркаси, побудовано архітектуру, що дозволить спростити супровід системи.

До недоліків системи можна віднести відсутність можливості імпортування великих об’ємів книг зараз.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Rational Unified Process Best Practices for Software Development Teams. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://www.ibm.com/developerworks/rational/library/content.
2. Introduction to JSON Web Tokens. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://jwt.io/introduction.
3. Сведения о .NET Core. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/core/about.
4. What is middleware?. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://azure.microsoft.com/en-us/overview/what-is-middleware.
5. Dependency Injection. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://www.tutorialsteacher.com/ioc/dependency-injection.
6. Object-Relational Mapping (ORM). [Електронний ресурс] – 08.12.2019 – Режим доступу: https://www.techopedia.com/definition/24200/object-relational-mapping--orm.
7. Entity Framework Core. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://www.techopedia.com/definition/24200/object-relational-mapping--orm.
8. What is database (DB). [Електронний ресурс] – 08.12.2019 – Режим доступу: https://searchsqlserver.techtarget.com/definition/database.
9. Object oriented programming in C#. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://searchwindevelopment.techtarget.com/definition/C.
10. MySQL Workbench Manual. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://dev.mysql.com/doc/workbench/en/wb-intro.html.
11. Unit Testing, Software Testing. [Електронний ресурс] – 08.12.2019 – Режим доступу: https://www.geeksforgeeks.org/unit-testing-software-testing/.

12. Закон України «Про розповсюдження примірників аудіовізуальних творів фонограм, відеограм, комп’ютерних програм, баз даних» №1587 від 01.01.2013.
13. Закон України «Про збір та облік єдиного внеску на загальнообов’язкове державне соціальне страхування» №2464-VI від 08.07.2010.
14. Податковий кодекс України, п.164.6 ст.164.
15. Постанова Кабінету Міністрів України «Про затвердження Порядку визначення класу професійного ризику виробництва за видами економічної діяльності» № 237 від 08.02.2012.
16. Методичні вказівки для виконання розділу магістерської роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. – 34 с.
17. Безпека життєдіяльності людини та суспільства. Правила безпеки при роботі з комп'ютером [Електронний ресурс] – 09.11.2017 – Режим доступу:https://pidruchniki.com/18210712/bzhd/pravila_bezpeki_pri_roboti_kompyuterom.

# ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп’ютерно-інформаційних систем і програмної інженерії
Кафедра програмної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою
програмної інженерії
докт. фіз.-мат. наук, професор Петрик М. Р.
“___ “_____ 201_ р.

# ТЕХНІЧНЕ ЗАВДАННЯ

**на виконання магістерської роботи**

«Розробка соціальної мережі для бібліотек на основі .NET технологій»
Гайдуку Роману Степановичу СПм-61-____ ТЗ

Керівник роботи
канд. техн. наук, доцент Михалик Д. М.
“___” _____ 201_р.
_____

Виконавець
студент групи СПм-62
Гайдук Роман Степанович
“___” _____ 201_р.
_____

м. Тернопіль – 2019

ЗМІСТ

# 1 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік , та згідно наказу на розробку дипломної роботи.

Тема роботи: «Розробка соціальної мережі для бібліотеки на основі .NET технологій».

# 2 ПРИЗНАЧЕННЯ РОЗРОБКИ

При пошуку різного роду джерел інформації виникає потреба у швидкому її отриманні. Не все глибокі знання в певній сфері можна знайти в інтернеті. Також багатьом людям просто подобається читати не з екранів пристроїв, а з паперових сторінок. Виходячи з цього вникає необхідність у використанні класичних джерел інформації, тобто книг.

Пошук книг в соціальному середовищі за допомогою бібліотек та різного роду людей є найоптимальнішим рішення з фінансової точки зору, але підвищуються затрати часу пов’язані із пересуванням та комунікацією. Соціальна мережа яка забезпечить швидкий пошук, обмін книгами та думками вирішить ці проблеми.

Метою виконання дипломної роботи є система, яка дозволить створювати електронну бібліотеку, вести облік виданих та взятих книг, перегляд книги онлайн, відображення розташування книги що цікавить.

Для реалізації серверної частини буде використано платформу .NET Core 2.0 та каркас ASP.NET, який використовує сучасні підходи для реалізації архітектури системи, що дозволить зменшити витрати та спростити її супровід. Для клієнтської частини – каркас Angular версії 8. Також необхідно використати Google Books API для отриманні інформації про книги та карти від компанії MapBox.

# 3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Функціональні характеристики

Програмний продукт має забезпечити виконання наступних дій:

а) авторизація в системі:

1) створення облікового запису;
2) вхід зареєстрованих користувачів;

б) користувацька частина:

1) додавання та видалення книг з електронної бібліотеки;
2) перегляд вмісту книги;
3) створення угод обміну книгами між користувачами;
4) облік взятих, виданих книг;
5) публікація постів та організація подій;
6) відображення на карті користувачів із книгою що цікавить;

## 3.2 Склад та параметри технічних засобів

Планується розміщення компонентів системи у хмарній платформі Azure. Для серверної складової можна використати базовий план другої категорії, який включає:

- 2 ядра процесора;
- 3 ГБ оперативної пам’яті;
- 10 ГБ дискового простору.

Для клієнтської частини буде достатньо базового плану першої категорії, який включає:

- 1 ядро процесора;
- 1.75 ГБ оперативної пам’яті;
- 10 ГБ дискового простору.

### 3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати в різних браузерах та платформах . Програмний продукт має назву «LibSet» і для серверної частини повинен бути застосований каркас ASP.NET Core 2.0 на основі мови програмування C# та з використанням реляційної бази даних MSSQL. Клієнтська частина повинна бути реалізована на основі каркасу Angular 8 та відобразити усі функціональні вимоги у зручному користувацькому інтерфейсі.

## 4 СТАДІЇ РОЗРОБКИ

- аналіз предметної області;
- пошук акторів та варіантів використання;
- вибір та проектування бази даних;
- встановлення необхідного ПЗ;
- розробка серверної частини;
- розробка клієнтської частини;
- розгортання програмної системи;
- тестування програмної системи;
- оформлення супровідної документації;
- здача проекту.

## 5 ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Технічне завдання;
- Пояснювальна записка.

# 6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Приймання проводиться спеціально створеною комісією в термін до “__” _____ 2019 р.

Р.Гайдук
Тернопільський національний технічний університет імені Івана Пулюя, Україна

# РОЗРОБКА СОЦІАЛЬНОЇ МЕРЕЖІ ДЛЯ БІБЛІОТЕК З ВИКОРИСТАННЯМ .NET ТЕХНОЛОГІЙ

R. Haiduk

DEVELOPING A SOCIAL NETWORK FOR LIBRARIES USING .NET TECHNOLOGIES

Розвиток сучасних інформаційних і комунікаційних технологій, зростання кількості інформації дедалі більше визначають сутність нашої епохи. Інформаційно-комунікативні технології мають визначальну роль у забезпеченні взаємозв'язку між різними суспільними структурами й соціальними групами, громадянським суспільством і владою, а також у системах підготовки та поширенні інформації, орієнтованої на масового користувача. Розвиток інформаційних технологій має центральне місце в процесі інтелектуалізації суспільства, розвитку освіти, науки й культури. Сама ж інформація є глобальним, невичерпним потенціалом людства, яке ввійшло в нову епоху розвитку цивілізації, а саме в епоху становлення інформаційного суспільства [1].

Аналіз наукових поглядів щодо нинішнього стану інноваційних і трансформаційних змін у бібліотечній галузі, а також сучасних теоретичних досліджень стратегій розвитку бібліотек підтвердив необхідність та своєчасність дослідження щодо функціонування бібліотек відповідно до вимог часу й потреб інформаційного суспільства. Саме це й становить актуальність цієї наукової роботи – як синтетичної, узагальнюючої праці з дослідження інформаційних процесів як фактора впливу на розвиток новітніх форм роботи бібліотек. Саме інформаційна діяльність бібліотечних установ має найбільш дійсні й позитивні перспективи в реалізації та закріпленні людських ідеалів, цінностей і норм гармонійного розвитку в сучасному світі.

Світовий досвід показує, що важливим кроком у розвитку бібліотек і забезпечення ними інформаційних потреб користувачів є обов'язкове створення інформаційно-бібліотечних мереж. Будучи однією з ланок у мережі бібліотечних та інформаційних установ, вона зможе працювати так, щоб забезпечувати доступ до інформації кожній людині, де б та не перебувала. Така мережа має бути спрямована не лише на розвиток інформаційного потенціалу країни, а й на те, щоб забезпечити рівність усіх громадян у можливості доступу до потрібних їм джерел, задовольнити їхні особисті й суспільні інтереси в інформації та підняти престиж освіченості, культури й авторитет бібліотечних установ [1].

Створення єдиного інформаційно-бібліотечного простору дасть змогу забезпечити широкі можливості для кооперування зусиль бібліотек у наданні користувачам розвинутого сервісу в корпоративному середовищі спільних інформаційних ресурсів і сервісів регіону, більше того, сприятиме побудові інформаційного суспільства в Україні. Упровадження новітніх електронних технологій дає змогу реалізувати комплексний підхід до вирішення інформаційних завдань. Користувач може отримати повний комплект різноманітних інформаційних матеріалів, у тому числі й інтернет-ресурси, а також консалтингові послуги. Бібліотеки сьогодні усвідомили свою роль навігаторів у безмежному масиві інформації, що вигідно відрізняє їх від інших комунікативних суспільних структур. Їхнє місце й значення можна охарактеризувати за такими критеріями:

– технологічний – інформаційна технологія широко застосовується в бібліотечних установах, у системі організації надання послуг;

– соціальний – інформаційні послуги бібліотек виступають важливим стимулятором зміни якості життя, формується й утверджується «інформаційна свідомість» при широкому доступі до інформації;

**Література**

1. Центр досліджень соціальних комунікацій / Бібліотека в сучасному інформаційному просторі [Електронний ресурс]. – Режим доступу: http://nbuviap.gov.ua/index.php?option=com_content&view=article&id=3172:biblioteka-v-suchasnomu-informatsijnomu-prostori&catid=81&Itemid=415