Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Програмне забезпечення для креативного письма при формуванні персонажів та сюжету авторських історій»
за освітньою програмою: «12 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»

Виконала: студентка групи ПЗ1911: (підпис) /Маргарита ВИСКАРКА/

Керівник: (підпис) / Ірина ШАПОВАЛ/

Нормоконтролер: (підпис) /Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2023

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Creative writing software for character and plot building of authors' stories»
according to educational curriculum «12 Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1911: _____ /Marharyta VYSKARKA/
(підпис)

Scientific Supervisor: _____ /Iryna SHAPOVAL/
(підпис)

Normative controller: _____ /Svitlana VOLKOVA/
(підпис)

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «12 Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу Бакалавр

студенту Вискарка Маргарита Юріївна

1. Тема роботи: «Програмне забезпечення для креативного письма при формуванні персонажів та сюжету авторських історій» Керівник роботи: Шаповал Ірина Вікторівна, старший викладач затверджені наказом № 1209 ст від 07.12.2022
2. Строк подання студентом роботи: 19.06.2023
3. Вихідні дані до роботи: _____.
4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Вступ;
   4.2. Збір та аналіз вимог до програмного забезпечення;
   4.3. Зовнішнє і внутрішнє проектування;
   4.4. Тестування та налагодження;
   4.5. Висновки;
   4.6. Література;

5. Перелік демонстраційного матеріалу:
   5.1. Доповідь;
   5.2. Презентація;
   5.3. Демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	01.04.2023 – 06.04.2023	
2	Огляд літератури та аналіз аналогів	07.04.2023 – 17.04.2023	
3	Розробка структур вхідних і вихідних даних, вимог до системи	18.04.2023 – 30.04.2023	
5	Узгодження та затвердження ТЗ, постановка задачі	01.05.2023 – 07.05.2023	30 %
6	Розробка та програмування логіки програми	08.05.2023 – 17.05.2023	
7	Розробка і реалізація інтерфейсу користувача	18.05.2023 – 21.05.2023	
8	Відлагодження програми	22.05.2023 – 28.05.2023	60%
9	Розробка, узгодження та затвердження програмної документації	29.05.2023 – 11.06.2023	
10	Розробка демонстраційних матеріалів	12.06.2023 – 18.06.2023	100%
	Подання кваліфікаційної роботи до кафедри	18.06.2023 – 25.06.2023	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.2023	

Студент _____ Маргарита ВИСКАРКА
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ ст. викл. Ірина ШАПОВАЛ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра 118 с., 29 рис., 7 табл., 7 джерел.

Метою роботи є створення застосунку для упорядкування процесу формування історії, її елементів та написання сюжету. Програма має надавати користувачу можливість створювати твори та працювати з ними, а саме будувати та описувати персонажів та світи, а також формувати та записувати сюжет історії. Іншою важливою функцією даного програмного забезпечення є надання користувачу вибору того, які саме елементи опису історії будуть присутні у програмі. За результатами роботи було створено застосунок, що виконує всі основні функції.

Пояснювальна записка складається з 8 розділів:

– вступ – в даному розділі описується актуальність, мета роботи та експлуатаційне призначення. Складається з 1 сторінки;

– збір та аналіз вимог до програмного забезпечення – у цьому розділі описуються та аналізуються аналоги програми, література по даній предметній області, а також проводиться опитування зацікавлених сторін для формування вимог до програмного забезпечення. Складається з 13 сторінок;

– зовнішнє і внутрішнє проектування – у цьому розділі відбувається визначення функціональних вимог до програмного забезпечення, його призначення, вхідних та вихідних даних, проектування архітектури системи, інтерфейсу користувача, бази даних. Складається з 23 сторінки;

– тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорної» та «білої» скриньки. Також аналіз помилок їх вплив на систему та вирішення проблеми. Складається з 8 сторінок;

– висновки. Складається з 1 сторінки;

– список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;

– додатки – містить робочий проект

Кількість таблиць: 7 штук.

Кількість рисунків: 29 штук.

Ключові слова: креативне письмо, історія, персонаж, світобудова, сюжет, лінія часу, подія, сцена, кастомізація.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Креативне письмо – написання творів нетрадиційних або незвичайних жанрів, яке потребує використання творчості та креативності, а також визначається наголосом на майстерності оповіді, розвитку персонажів і використанні літературних тропів.

Історія – сукупність вигаданих або реальних людей, фактів та події, які складають собою розповідь.

Персонаж – людина або створіння в історії.

Світобудова –. вигаданий всесвіт історії, який описує де та коли відбуваються її події.

Сюжет – сукупність подій у творі, які пов’язані між собою та разом утворюють оповідання історії.

Лінія часу – окрема хронологія подій, яка описує що відбувається та у якому порядку у загальній історії.

Подія – .явище, факт або вчинок, те, що привносить зміни у сюжет історії.

Сцена – окрема частина дії у творі.

Кастомізація – це дії по створенню або зміні продукту чи послуги відповідно до вподобань чи вимог людини чи компанії.

ВСТУП

В наш час майже будь-яка сфера зайнятості та інтересів розвивається паралельно та у зв’язку з новинками сучасних технологій. Все більше нашої діяльності переходить у цифровий формат по ряду причин: надійне збереження інформації, зручність обробки даних та легкість їх передачі. Професія та справа письменництва не є винятком, воно завжди дотримувалось тенденцій часу, так як спершу записи відбувались на папері пером, потім ручкою, згодом за допомогою друкарської машинки, а тепер на електронних девайсах.

Письмо потребує детального планування та обмірковування багатьох частин історії, таких як персонажі, світобудова, сюжет, а також знаходження необхідної інформації та самого написання твору. При такому обсязі даних можливо легко загубитись та втратити мотивацію на шляху до мети, так і не дозволивши історії побачити світ. Це визначає потребу в зручному застосунку для упорядкування процесу формування історії.

Написання твору є креативною та індивідуальною справою, кожна людина підходить до неї по-різному, кожна історія потребує окремого ставлення та набору даних, які в речах пов’язаних з творчістю зазвичай важко передбачити заздалегідь. Тому створення даного програмного забезпечення дозволить не тільки збереження необхідної для користувача інформації про персонажів, світ та сюжет історії, а ще й надасть письменнику можливість обирати які елементи історії він хоче зберігати.

Письменництво є одним з найпопулярніших захоплень у світі, тому користування цим застосунком розраховане на будь-кого, хто зацікавлений у письмі, включаючи як і тих, хто тільки починає писати, так і тих, хто цим займається вже довгий час. Для новачків застосунок допоможе розпочати писати та продумати деталі власних історій, в той час як для досвідчених письменників це буде зручним способом упорядкувати їх створення.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

1.1 Огляд програмних аналогів

Наразі для допомоги при письмі на ринку існує багато різних застосунків з широкою сферою призначення: для зручного запису історії, для організації інформації, для планування письма. В наслідок цього при відборі аналогів були визначені такі критерії до функціоналу, як можливість запису історії у застосунку, різноманітність категорій і функцій для повного та зручного запису інформації, а також можливість персоналізації. Для огляду було визначено програмні продукти «Fortelling» та «Writer's companion», так як вони найбільше підходять під вимоги.

1.1.1 Застосунок «Fortelling»

«Fortelling» є програмним забезпеченням для побудови новел. Застосунок розроблений AJP Digital Tools, він створений на основі Flutter, є доступним безкоштовно в мобільній і веб версії, а також в десктопній при платній підписці. В безкоштовних версіях користувачу доступно будувати, редагувати історії та книжки, описувати різні елементи, а також співпрацювати з іншими авторами (див. рис. 1.1).

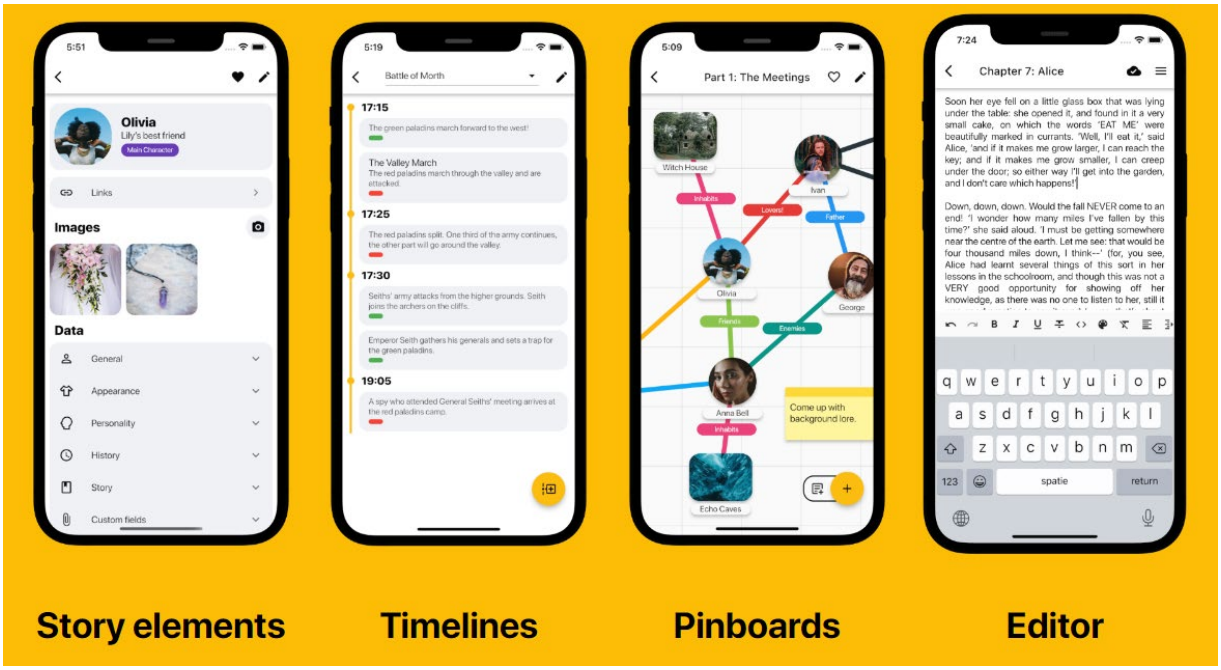

Рисунок 1.1 – Демонстрація інструментів «Fortelling» (зображення взято з офіційного сайту застосунку)

Fortelling надає інструменти, необхідні для створення сюжету та написання книги, а також упорядкування елементів історії. Робота з твором розбивається на такі основні розділи: письмо, історія, персонажі, світобудова та додаткові інструменти. При письмі письменник може набирати текст глави, задавати сцени та зберігати версії тексту, а також присутня можливість створювати стрічки подій історії. Користувач може додавати персонажів, створювати локації, заповнювати їх характеристики та займатись світобудовою.

Перевагами додатку є можливість детального опису персонажів та локацій, додавання різноманітних елементи роботи з сюжетом, створення дошок для пов’язування елементів та лейблів, які можна додавати до різних елементів історії.

В додатку також присутня можливість додавання персоналізованих полів до таких елементів як персонажі та локації, але немає загальної можливості налаштовувати поля та керувати ними.

1.1.2 Застосунок «Writer’s companion»

«Writer’s companion» є програмним забезпеченням для допомоги письменникам у плануванні своїх романів, організації їх змісту та досягненні цілей. Застосунок розроблений Pandaway Studios, доступна мобільна версія. В застосунку можливо додавати проекти, займатись світобудовою, упорядковувати інформацію, планувати створення проекту (див. рис. 1.2), а також ставити цілі.

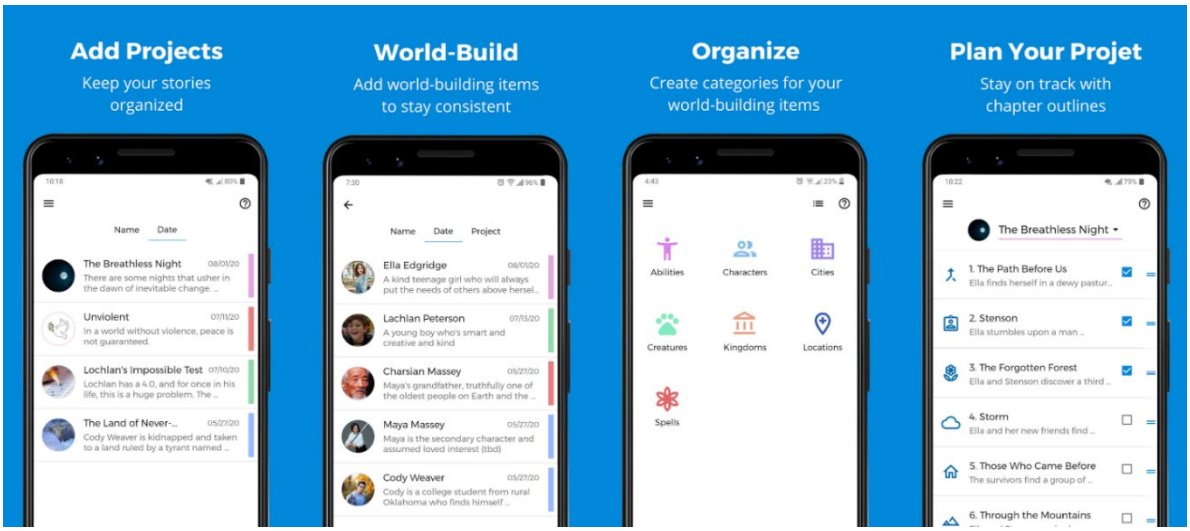

Рисунок 1.2 – Демонстрація функціоналу «Writer's companion» (зображення взято зі сторінки в крамниці застосунків)

В програмі присутні такі основні розділи: проекти, де користувач може додавати та коротко описувати історії; світобудова, де можливо створювати власні категорії та елементи для опису світу; план, де можливо створювати опис глав роботи, та цілі для визначення й досягнення мети.

Перевагами додатку є персоналізована світобудова та надання користувачу можливості самому створювати категорії, елементи в них та власні поля для їх опису, але користувачу не пропонуються ідеї, що саме можна створити.

Відміченим недоліком є також те, що застосунок слугує лише для планування історії та не призначений для самого письма, тому користувач може лише коротко описувати елементи твору.

Підсумувавши, при аналізі застосунку «Fortelling» було підмічено широкий функціонал, який досить повно дозволяє описати основні частини історії, а також надає можливість її написання. Було підмічено такі елементи історії та функції, які варто запровадити при розробці програмного забезпечення:

- детальний опис сюжету історії: глави, сцени, події та стрічки часу,
- детальний опис персонажів та світобудови,
- запис тексту історії у застосунку,
- теги або лейбли.

На відміну від «Fortelling», застосунок «Writer’s companion» був направленим лише на планування історії, тому мав більш вузький функціонал, але варто взяти до уваги його кастомізацію: користувач може створювати власні категорії та елементи. Однак, варто також відмітити відсутність пропозицій або прикладів для користувача при створенні цих елементів, тобто можлива ситуація, коли він може не знати, що йому може знадобитись.

1.2 Огляд літератури

1.2.1 Побудова історії

Для розробки дійсно зручного та корисного додатку для допомоги при креативному письмі спочатку необхідно дослідити, з чого саме складається сам процес написання історій та з яким їх елементами зазвичай працює письменник.

1.2.1.1 Процес створення історії

Побудова та написання історії, безумовно, є процесом креативним та індивідуальним, тому кожна людина до нього підходить по-різному. Як кожна історія може розкривати письменника як людину та митця, так і сам процес її створення не може не відбивати його особистість. Робота може залежати від того, що для творця є більш звичним, зручним або інтуїтивним, тому не можливо сказати що є якийсь «рецепт», по якому кожний письменник буде створювати свою історію.

Однак, все ж таки можливо звернути увагу на деякі етапи цього складного творчого процесу, на яких зупиняється багато тих, хто пише. Отже, можна виділити такі 5 основних кроків:

1. Попереднє написання або планування.
   Сюди входить знаходження та обмірковування ідей, вибір персонажів та місця подій, створення нарису роботи.
2. Дослідження, якщо воно потрібне.
   Знаходження джерел інформації та натхнення, які можуть допомоги при побудові історії.
3. Створення чернеток та записів.

Перетворення ідей та думок на конкретні речення та параграфи, створення тезису, написання історії.

4. Перегляд та переосмислення.

   Перечитування твору, упорядкування тексту, додаткові дослідження.

5. Редагування та коректура.

   Перевірка написання, пунктуації, правильності тексту [1][2].

Також варто відмітити, що робота над історію є рекурсивним процесом, тобто до початкових етапів письменник може неодноразово повертатись [3] (див. рис. 1.3).

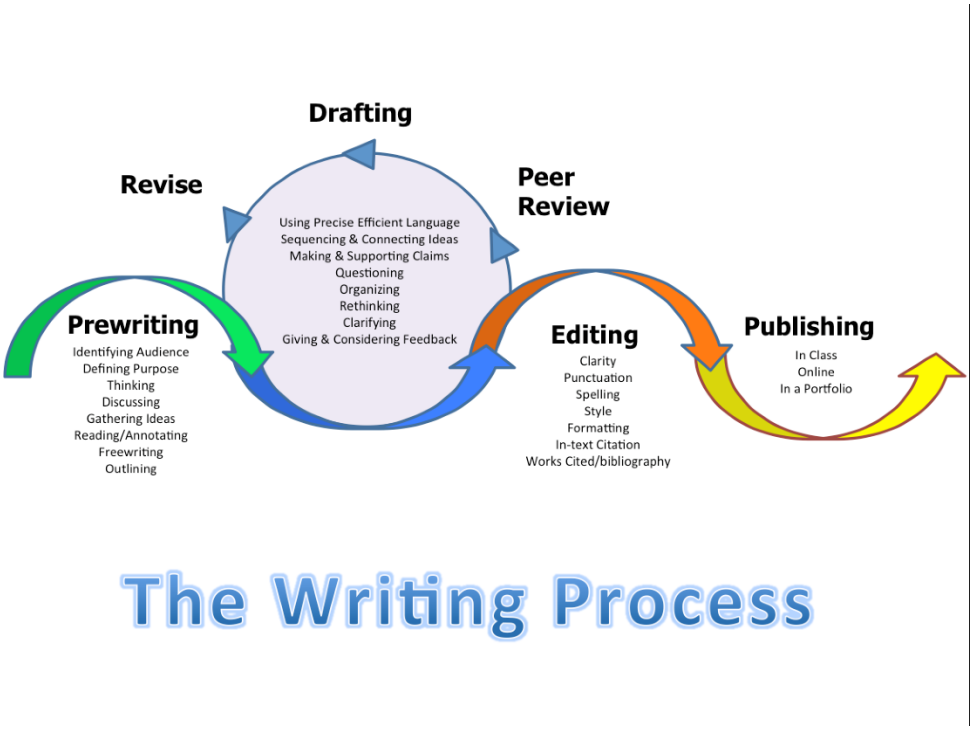

Рисунок 1.3 – Зображення процесу написання твору

Можна помітити, що у деяких аспектах цей процес дуже схожий на створення проектів та програмного забезпечення. Як приклад можна навести ітераційну модель життєвого циклу ПЗ, етапи якої включають: планування, дослідження з визначенням вимог та їх аналізом, проєктування, реалізація та тестування, а також їх повторення поки не буде досягнуто бажаного результату та не створено фінальний продукт (див. рис. 1.4) [4].

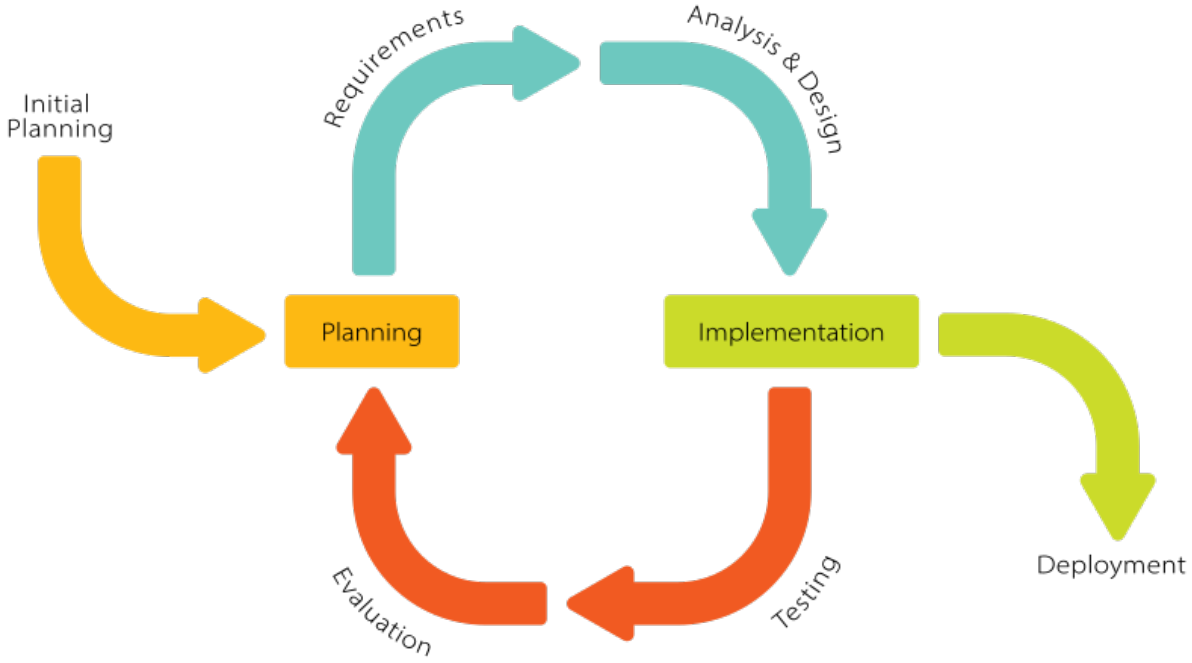

Рисунок 1.4 – Ілюстрація ітеративної моделі життєвого циклу ПЗ

1.2.1.2 Основні елементи історії

На відміну від процесу створення історії, який може залежати індивідуально від людини, саме вміст більшості успішних творів має деякі визнані елементи. Саме від них залежить наповнення історії та її цікавість.

Кількість даних елементів є варіативною, але можна виділити 5 фундаментальних:

1. персонажі,
2. оточення,
3. сюжет,
4. конфлікт,
5. резолюція [5].

Деякі ж письменники розширюють їх до 7 та більше, включаючи також Тему, Точку зору, Стиль та інші елементи твору [6]. Роздивимось саме фундаментальні 5, так як вони описують основні частини історії при її написанні.

Персонажі. Саме вони є однією з найважливіших частин історії, вони знаходяться у її серці та у центрі подій. Якщо читач не є зацікавленим у головному або в будь-яких інших персонажах, є шанс що історія не буде прочитаною.

Оточення. Оточення можна поділити на три основні частини: локація, період часу та настрій [5], всі ці аспекти описують де та коли саме відбувається історія та задають її атмосферу.

Сюжет. Описує всі події історії від початку і до кінця. Сюжет дає читачу знати, що відбувається, описує з якими проблемами зустрічаються персонажі, і надає деталі того, як саме вони намагаються їх вирішити [7].

Конфлікт. Це саме те, що персонажі історії намагаються перебороти. Конфлікт є важливою частиною сюжету, так як він розкриває сутність персонажів та робить історію цікавою.

Резолюція. Вона і є вирішенням тієї поставленої проблеми [5]. Якщо у історії вона слабка або погана, то це залишить читача незадоволеним від прочитаного, тому цього необхідно уникати при побудові історії.

1.2.2 Кастомізація

Кастомізація – це дії по створенню або зміні продукту чи послуги відповідно до вподобань чи вимог людини чи компанії. «Кастомізація дозволяє користувачам самостійно вибирати, що вони хочуть бачити, або встановлювати параметри для того, як інформація організовується або відображається. Це може покращити взаємодію з користувачем, оскільки дозволяє користувачам контролювати свою взаємодію», тобто таким чином користувач сам організовує використання застосунком та обирає що саме він хоче [8].

Кастомізація може бути чудовим способом покращити та персоналізувати використання програмного забезпечення користувачами, якщо вона буде реалізована правильно. Однак, недоліком кастомізації є те, що іноді користувачі самі не знають, чого вони бажають [8]. Тобто безліч вибору та можливості змін не завжди є оптимальним варіантом, а отже з цим необхідно бути обережним.

Підсумувавши, під час огляду літератури було проаналізовано та вивчено процес створення історії, її основні елементи та розглянуто користь кастомізації. Ґрунтуючись на вивченій інформації, можна зробити висновки про важливість планування письма. Процес створення твору включає обдумування великої

кількості інформації та створення чернеток, поки не буде досягненим створення фінального продукту.

1.3 Опитування зацікавлених сторін

Для більш точного визначення необхідних вимог до розроблюваного застосунку було проведено опитування ряду людей, як серед студентів кафедри КТС, так і інших. Усього було поставлено 9 запитань та отримано 16 відповідей.

Як видно на діаграмі (див. рис. 1.5), більша частина опитаних були потенційно зацікавлені у письмі, половина людей вже займались писемністю.

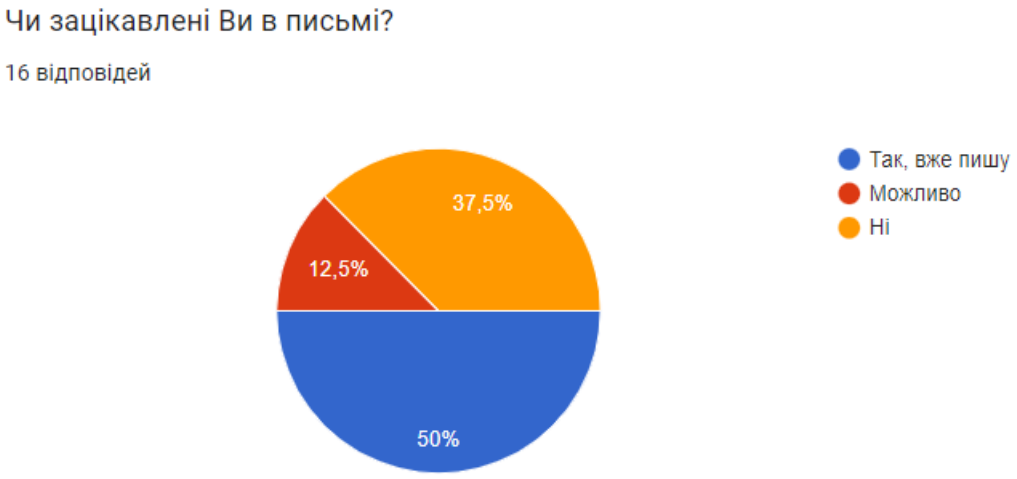

Рисунок 1.5 – Діаграма з першим питанням

Більшість людей, як видно на рис. 1.6, вже записують свої думки та ідеї.

Рисунок 1.6 – Діаграма з другим питанням

Також було поставлено основне питання корисності створення додатку, який допоміг би з об’єднанням даних задач. Майже всі люди, окрім тих, хто не зацікавлений в письмі, відповіли позитивно, що продемонстровано на рис. 1.7.

Рисунок 1.7 – Діаграма з третім питанням

Переважна частина людей не була знайома з існуючими аналогами (див. рис. 1.8).

Рисунок 1.8 – Діаграма з четвертим питанням

Також було поставлено питання важливості кастомізації та індивідуального досвіду роботи з додатком. Як видно на рис. 1.9, 10 з 16 людей знайшли дану функцію дуже корисною, з чим в трохи меншому обсязі погодились і інші опитувані.

Рисунок 1.9 – Діаграма з п'ятим питанням

Також майже всі опитувані люди проявили зацікавленість до можливості запису тексту історії одразу в застосунку (див. рис. 1.10).

Рисунок 1.10 – Діаграма з шостим питанням

Також було запропоновано на вибір деякі елементи роботи з сюжетом. Як видно з діаграми (див. рис. 1.11). Більшість людей надали перевагу главам, подіям, лініям часу та нотаткам.

Рисунок 1.11 – Діаграма з сьомим питанням

Абсолютно всі опитані люди знайшли можливість додавання посилань та фото корисною (див. рис. 1.12).

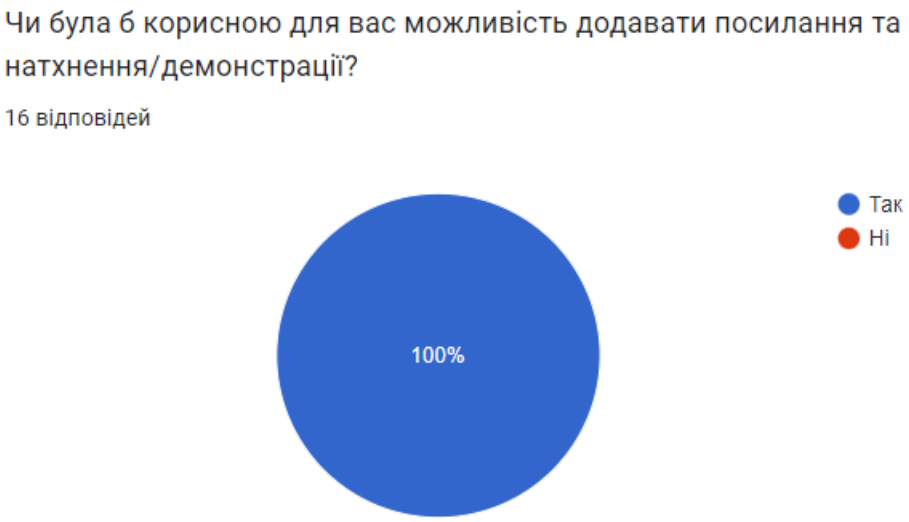

Рисунок 1.12 – Діаграма з восьмим питанням

З них більшість хотіли б мати можливість додавати їх до кожного розділу та додатково мати один спільний (див. рис. 1.13).

Рисунок 1.13 – Діаграма з дев'ятим питанням

Отже, опитування зацікавлених сторін підтвердило користь створення даного програмного забезпечення та надало інформацію про те, що саме хочуть бачити потенційні користувачі серед функцій та елементів, а саме:

- можливість написання історії в самому застосунку,
- наявність глав, подій, ліній часу та нотаток,
- можливість додавати посилання та фото до розділів.

Висновки до розділу 1

При розгляді аналогів та огляді літератури було визначено основний перелік вимог до продукту, що розробляється, а при опитуванні зацікавлених сторін було отримано відгук від потенційних користувачів програми.

З вивченої інформації про процес написання історії та її елементи було визначено такі фундаментальні її частини, які важливо включити у додаток: персонажі, оточення та сюжет, який і включає в себе конфлікт та резолюцію.

Під час огляду інформації про кастомізацію було визначено один її недолік, який може існувати при взаємодії користувачів з застосунком, а саме їх незнання того, чого саме вони бажають при доданні власних елементів. Ця проблема може бути вирішена наданням користувачу якоїсь початкової інформації, з якою він може розпочати працювати та надалі визначити, що саме йому може бути необхідно.

На основі розглянутої інформації було висунуто повний перелік вимог до створюваного застосунку:

- створення персонажів та світів, можливість детального та персоналізованого їх опису,
- додавання та опис різних елементів сюжету історії, таких як чернетки, глави, сцени, лінії часу та події,
- створення тегів на їх основі для позначення інформації,
- додавання фото та посилань,
- можливість написання історії у застосунку.

2 ПРОЄКТУВАННЯ

2.1 Зовнішнє проєктування

2.1.1 Функціональне призначення

Основним функціональним призначенням програми є організація процесу формування історії, її елементів, та її написання. Сюди відноситься упорядкування роботи, думок та інформації при креативному письмі.

2.1.2 Експлуатаційне призначення

Дане програмне забезпечення має таке експлуатаційне призначення:

- облегшення процесу створення та написання історії,
- зменшення часових витрат на роботу з інформацією.

2.1.3 Функціональні вимоги

Було визначено такі функціональні вимоги для програми:

- побудова, редагування та видалення історій та таких її елементів:
  - персонажів історії,
  - світобудови історії,
  - сюжету історії,
    - глав,
    - чернеток,
    - сцен,
    - ліній часу,
    - подій,
    - тегів,
  - персоналізованих секцій та полів,
  - посилань та фотографій,
  - нотаток,
- написання історії,
- пошук по історіям.

2.1.4 Вхідні та вихідні дані

2.1.4.1 Вхідні дані

При проектуванні було визначено такий набір вхідних даних:

Таблиця 2.1 – Вхідні дані

Назва	Опис	Тип
username	ім'я користувача	набір символів, не більше 50
password	пароль	набір символів, не більше 50
story	обрана історія	ціле число, що позначає ідентифікатор історії
search	пошуковий запит	набір символів, не більше 50
story: cover	обкладинка історії	зображення
story: title	назва історії	набір символів, не більше 50
story: description	опис історії	набір символів, не більше 500
character	обраний персонаж	ціле число, що позначає ідентифікатор персонажу
character: icon	обкладинка персонажу	зображення
character: name	ім’я персонажу	набір символів, не більше 50
character: description	опис персонажу	набір символів, не більше 500
world	обраний світ	ціле число, що позначає ідентифікатор світу
world: icon	обкладинка світу	зображення
world: name	найменування світу	набір символів, не більше 50

Продовження таблиці 2.1

worlds: description	опис світу	набір символів, не більше 500
sectionname	назва секції	набір символів, не більше 50
section	обрана секція	ціле число, що позначає номер секції
fieldname	назва поля	набір символів, не більше 50
field	обране поле	ціле число, що позначає номер поля
fieldtext	текст поля	набір символів, не більше 100
picture	зображення	зображення
draft	обрана чернетка	ціле число, що позначає номер чернетки
chapter	обрана глава	ціле число, що позначає номер глави
chapter: name	назва глави	набір символів, не більше 50
chapter: description	опис глави	набір символів, не більше 500
chapter: text	текст глави	набір символів, не більше 30000
scene	обрана сцена	ціле число, що позначає ідентифікатор сцени
scene: name	назва сцени	набір символів, не більше 50

Закінчення таблиці 2.1

tagname	назва тегу	набір символів, не більше 50
timeline	обрана лінія часу	ціле число, що позначає номер лінії часу
timeline: name	назва лінії часу	набір символів, не більше 50
event	обрана подія	ціле число, що позначає ідентифікатор події
event: name	назва події	набір символів, не більше 50

2.1.4.2 Вихідні дані

При проектуванні було визначено такий набір вхідних даних:

Таблиця 2.2 – Вхідні дані

Назва	Опис	Тип
storyCard	побудована картка історії	контейнер
characterCard	побудована картка персонажу	контейнер
worldCard	побудована картка світу	контейнер
sectionCard	побудована картка секції	контейнер
chapterCard	побудована картка глави	Контейнер
field	створене поле секції	текстовий елемент
scene	створена сцена	текстовий елемент
event	створена подія	текстовий елемент
tag	створений тег	текстовий елемент

2.2 Внутрішнє проектування

2.2.1 Проєктування бази даних

2.2.1.1 Моделювання бази даних на концептуальному та логічному рівні

Даний проєкт має велику кількість інформації, яку необхідно зберігати для кожного користувача на сервері. Для її організації у базі даних було виділено такі основні сутності та зв’язки:

- Користувачі, які можуть створювати та працювати з Історіями;
- Історії, які вміщають у себе різні розділи, такі як Персонажі, Світи, різні елементи сюжету (Чернетки, Лінії часу), а також галереї цих розділів (Нотатки, Посилання, Зображення);
- Персонажі, які мають Секції та Атрибути,
- Світи, які мають Секції та Атрибути,
- Секції, які мають Атрибути,
- Атрибути, які належать Секції;
- Чернетки, які включають у себе Глави;
- Глави, які мають Статус та Теги, та яким відповідають Сцени;
- Сцени, які відповідають Главі,
- Лінії часу, які включають у себе Події;
- Події, які належать Лінії часу;
- Статуси;
- Теги, які можуть бути одного з Типів;
- Типи, які визначають тип Тегів, Нотаток, Посилань та Зображень;
- Нотатки, які мають Тип та відповідають Історії;
- Посилання, які мають Тип та відповідають Історії;
- Зображення, які мають Тип та відповідають Історії.

На основі визначених сутностей та зв’язків було побудовано ER-діаграму логічної схеми бази даних (див. рис. 2.1).

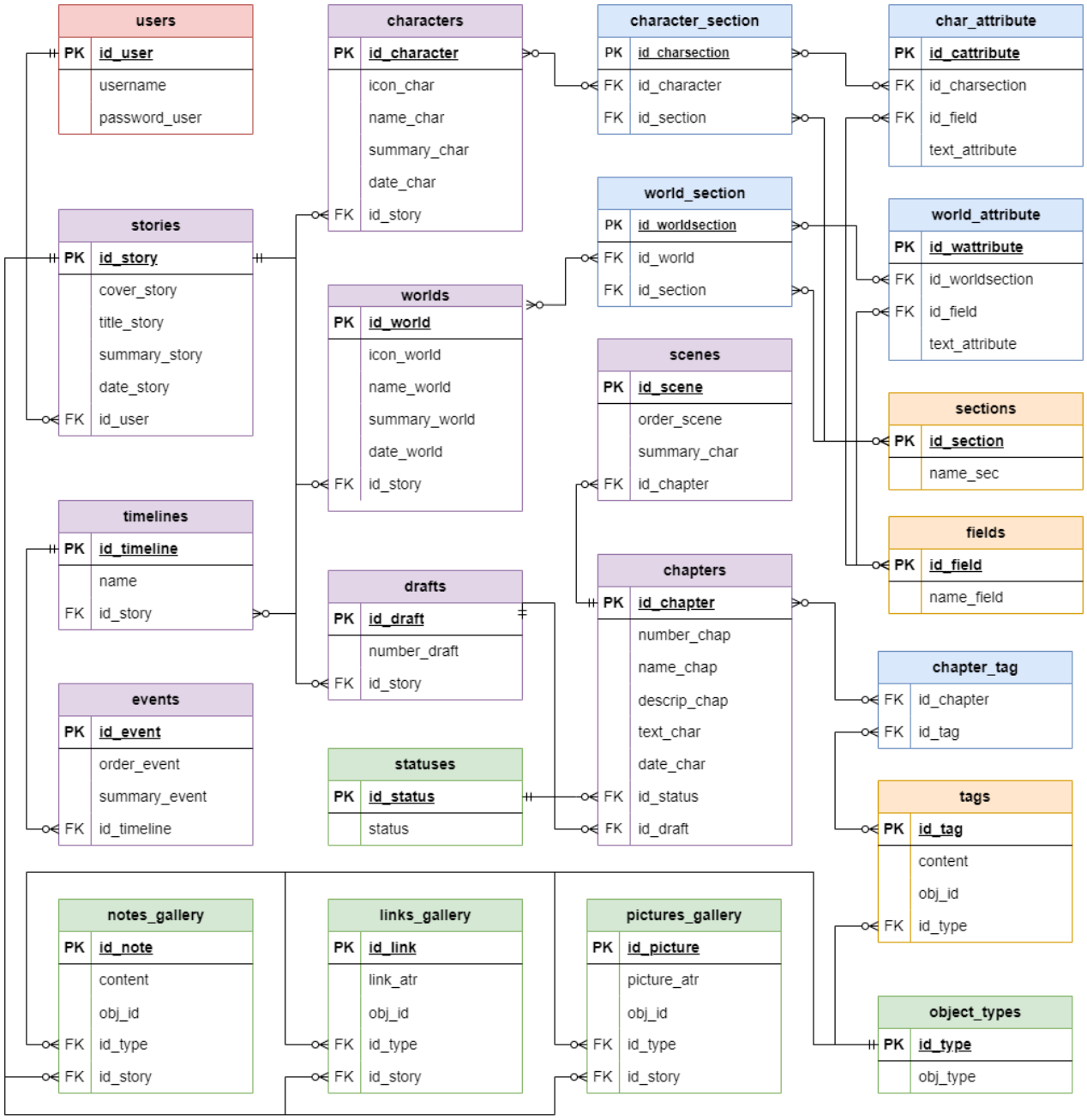

Рисунок 2.1 – Логічна схема бази даних

2.2.1.2 Нормалізація відносин бази даних

Нормалізація розробленою бази даних не потребується, так як вона вже знаходиться у 3 нормальній формі.

База даних знаходиться у 1НФ, так як значення полів всіх таблиць є неподільними. Винятками можуть бути такі поля, як Ім’я та Назва, так як неможливо визначити на скільки частин вони можуть поділятись.

База даних знаходиться у 2НФ, так як вона вже знаходиться у 1НФ та кожний неключовий атрибут неприводимо залежить від ключа.

База даних знаходиться у 3НФ, так як вона вже знаходиться у 2НФ та відсутні поля, які транзитивно залежні від первинного ключа. Кожне поле таблиць залежить лише від ключа, залежність полів між собою відсутня.

2.2.2 Проєктування архітектури системи

2.2.2.1 Моделювання словника системи

Під час аналізу зовнішніх специфікацій системи було виділено такі сутності: контролер, контролер форм, база даних, обробка зображень, список секцій, секція, поле, список чернеток, чернетка, глава. Також було виділено такі основні сутності інтерфейсів для взаємодії з користувачем: форма логіна, форма основної сторінки, форма історії, вкладка персонажів, сторінка персонажа, вкладка світів, сторінка світу, форма налаштування, вкладка сюжету, форма налаштування глав, вкладка натхнення, картка історії, картка секції, картка редагування секції, картка об’єкту, картка глави, форма додання елементу, форма додання глави, форма додання тегу, форма додання поля, форма редагування поля, форма видалення елементу, форма видалення поля.

Були ідентифіковані такі обов’язки:

- контролер – контроль взаємодії з об’єктами класів,
- контролер форм – контроль роботи форм,
- база даних – підключення та робота з базою даних,
- обробка зображень – перетворення та обробка зображень,
- список секцій – створення списку секцій для персонажу чи світу,
- секція – збереження інформації про секцію та створення списку полів,
- поле – збереження інформації про поле,
- список чернеток – створення списку чернеток для історії,
- чернетка – звернення до бази даних, збереження інформації про чернетку, створення списку глав для історії,
- глава – збереження інформації про главу,
- форма логіна – авторизація та реєстрація користувача,

- форма основної сторінки – робота з колекцією історій (вибір історії, створення історії, пошук),
- форма історії – заповнення інформації, збереження історії, її видалення,
- вкладка персонажів – робота з колекцією персонажів (вибір та створення персонажу),
- сторінка персонажа – робота з інформацією для персонажа (заповнення загальних відомостей та секцій; додання, редагування нотаток та зображень у галерею), збереження персонажу, його видалення,
- вкладка світів – робота з колекцією світів (вибір та створення світу),
- сторінка світу – робота з інформацією для світ (заповнення загальних відомостей та секцій; додання, редагування нотаток та зображень у галерею), збереження світу, його видалення,
- форма налаштування – перегляд, створення та редагування секцій та полів,
- вкладка сюжету – створення чернеток, глав, сцен, ліній часу, подій, нотаток, галереї зображень та робота з ними,
- форма налаштування глав – перегляд, редагування та видалення глав,
- вкладка натхнення – перегляд зображень та посилань історії,
- картка історії – відображення основної інформації про історію,
- картка секції – відображення інформації про секцію, редагування її полів,
- картка редагування секції – відображення та редагування рис секції та полів,
- картка об’єкту – відображення основної інформації про об’єкт (персонажа або історію,
- картка глави – відображення та редагування деталей глави,

- форма додання елементу – отримання інформації та створення необхідного елементу,
- форма додання глави – отримання інформації та створення глави,
- форма додання тегу – отримання інформації та створення тегу,
- форма додання поля – отримання інформації та створення поля,
- форма редагування елементу – перегляд, редагування інформації та видалення елементу,
- форма видалення елементу – вибір та видалення елементу,
- форма видалення поля – вибір та видалення поля.

Повні трибути та операції, необхідні для виконання обов'язків кожної сутності-класу можна буде побачити на діаграмі класів (див. рис. 2.3).

2.2.2.2 Моделювання залежностей

Результат визначення залежностей серед класів наведено у таблиці 2.3.

Таблиця 2.3 – Моделювання залежностей

Клас, який зв'язується	Клас, з яким зв’язуються	Тип зв’язку
контролер	база даних	Агрегація
	обробка зображень	Агрегація
	список секцій	Асоціація
	список чернеток	Асоціація
	секція	Агрегація
	чернетка	агрегація
контролер форм	форма логіна	агрегація
	форма основної сторінки	агрегація
	форма історії	агрегація
список секцій	контролер	агрегація
	секція	залежність
	поле	асоціація
секція	поле	залежність
список чернеток	контролер	агрегація

Продовження таблиці 2.3

	Чернетка	залежність
	глава	асоціація
чернетка	глава	залежність
форма логіна	контролер форм	агрегація
	контролер	агрегація
форма основної сторінки	контролер форм	агрегація
	контролер	агрегація
	картка історії	залежність
форма історії	контролер форм	агрегація
	контролер	агрегація
	вкладка персонажів	композиція
	сторінка персонажів	композиція
	вкладка світів	композиція
	сторінка світу	композиція
	вкладка сюжету	композиція
	вкладка натхнення	композиція
вкладка персонажів	контролер	агрегація
	картка об’єкту	залежність
сторінка персонажа	контролер	агрегація
	картка секції	залежність
	секція	асоціація
	сторінка налаштувань	асоціація
	форма додавання елементу	асоціація
	форма редагування елементу	асоціація
вкладка світів	контролер	агрегація
	картка об’єкту	залежність

Продовження таблиці 2.3

сторінка світу	Контролер	агрегація
	картка секції	залежність
	секція	асоціація
	сторінка налаштувань	асоціація
	форма додавання елементу	асоціація
	форма редагування елементу	асоціація
форма налаштування	контролер	асоціація
	картка редагування секції	залежність
	секція	асоціація
	форма додання елементу	асоціація
	форма додання поля	асоціація
	форма видалення елементу	асоціація
	форма видалення поля	асоціація
вкладка сюжету	контролер	агрегація
	форма налаштування глав	асоціація
	форма додавання тегу	асоціація
	форма додавання елементу	асоціація
	форма додавання глави	асоціація
	форма редагування елементу	асоціація
форма налаштування глав	контролер	агрегація

Закінчення таблиці 2.3

	форма видалення елементу	асоціація
вкладка натхнення	контролер	агрегація
картка секції	контролер	агрегація
	секція	агрегація
	поле	асоціація
картка редагування секції	контролер	агрегація
	секція	агрегація
	поле	асоціація
картка глави	контролер	агрегація
	чернетка	агрегація
	глава	асоціація
форма додання елементу	контролер	агрегація
форма додання глави	контролер	агрегація
форма додання тегу	контролер	агрегація
форма додання поля	контролер	агрегація
форма редагування елементу	контролер	агрегація
форма видалення елементу	контролер	агрегація
форма видалення поля	контролер	агрегація

На основі описаних вище сутностей та зв’язків спершу було побудовано діаграму для головних класів системи (див. рис. 2.2).

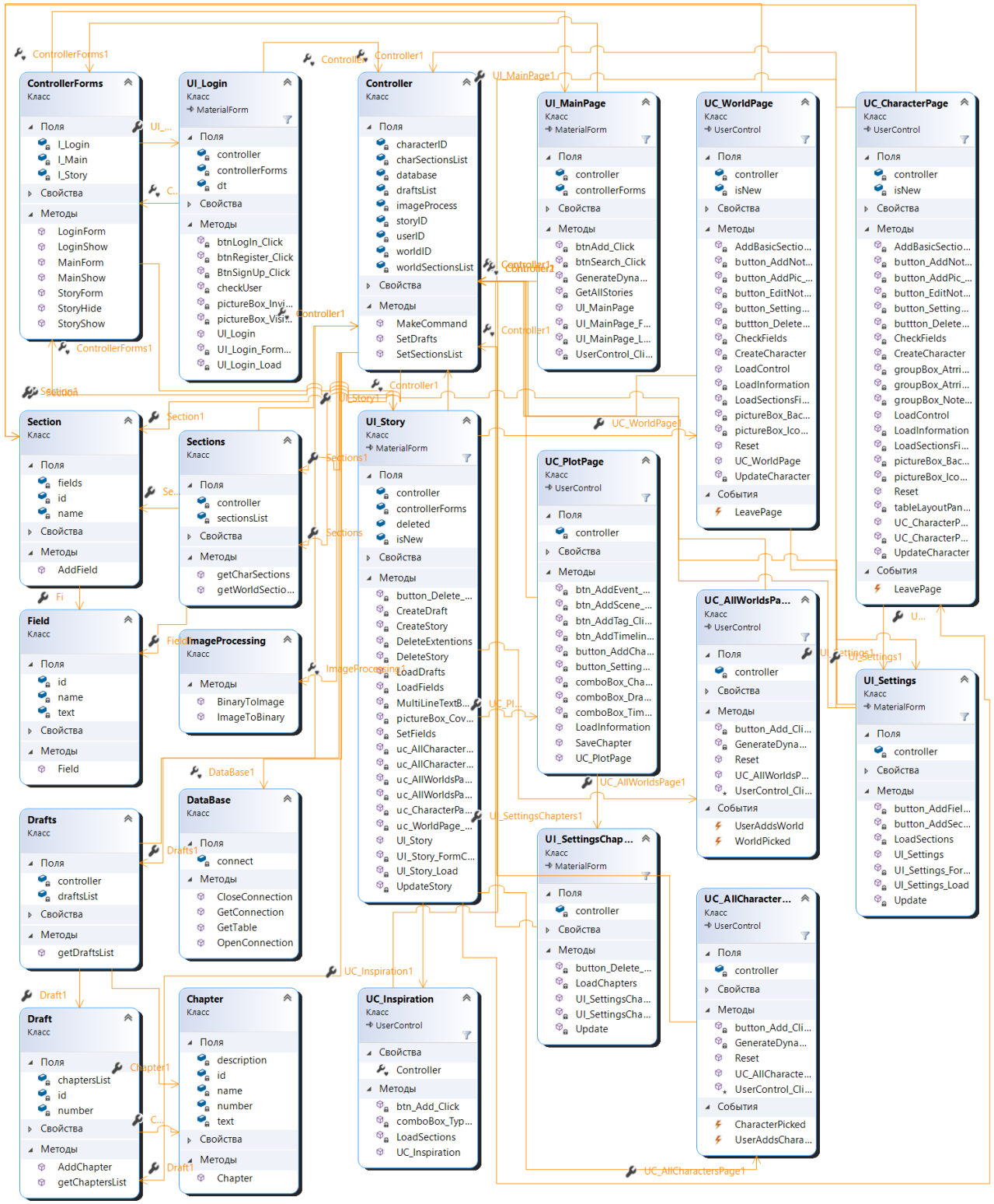

Рисунок 2.2 – Діаграма класів для головних класів системи

Після цього було представлено заключний результат моделювання у вигляді повної діаграми класів (рис. 2.3).

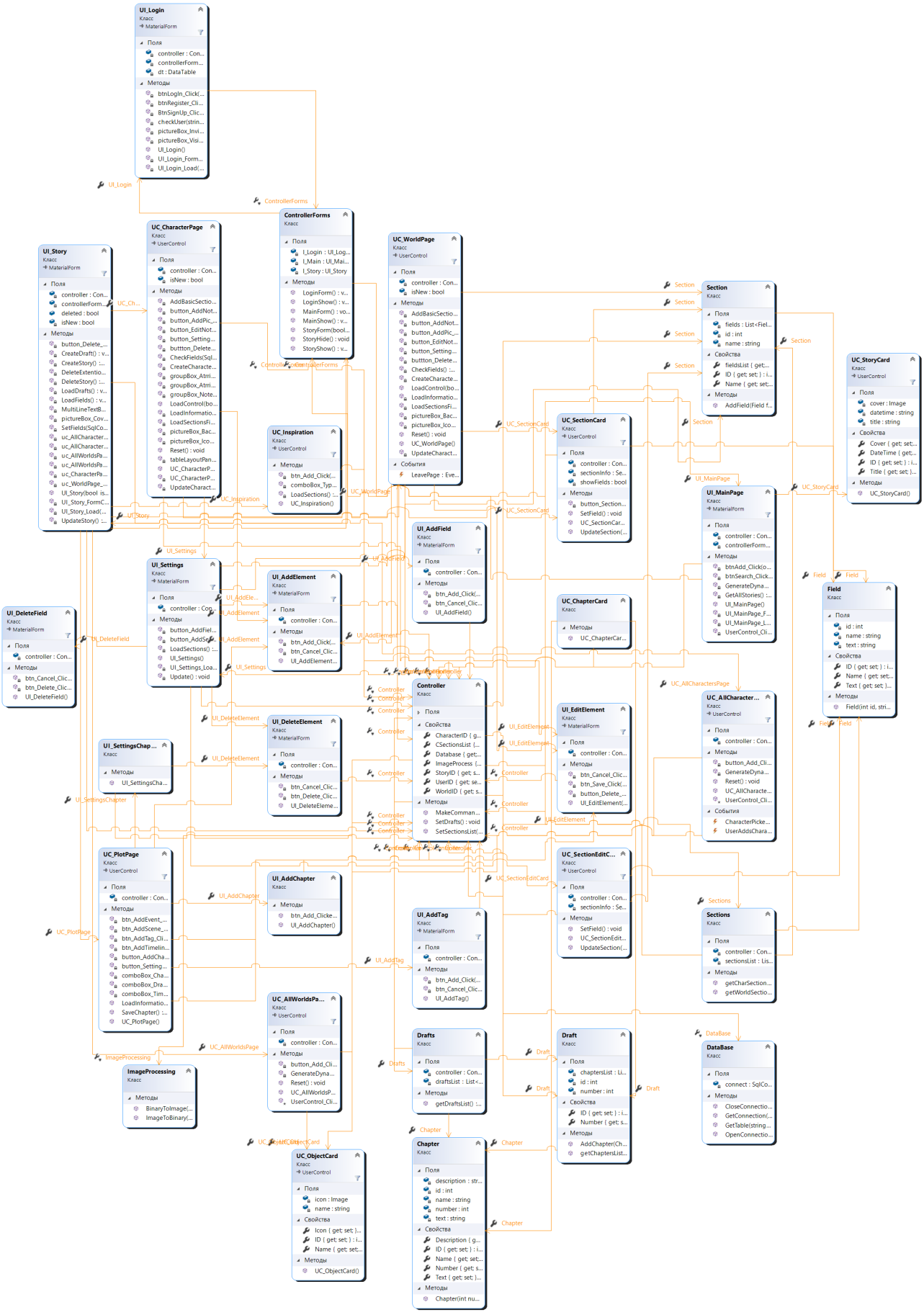

Рисунок 2.3 – Заключна діаграма класів

2.2.3 Проєктування інтерфейсу користувача

На основі функціональних вимог можна описати які можливості має надавати інтерфейс програми:

- вибір та створення історії,
- пошук по історіям,
- перегляд існуючих персонажів,
- вибір або створення персонажу,
- редагування його елементів,
- перегляд існуючих світів,
- вибір або створення світу,
- редагування його елементів,
- робота з сюжетом історії,
- редагування елементів історії.

На основі цих вимог можна визначити такі основні форми та вкладки:

- форма входу у застосунок,
- форма основної сторінки,
- форма сторінки історії,
    - вкладка персонажів,
    - сторінка персонажа,
    - вкладка світів
    - сторінка світу,
    - сторінка їх налаштувань,
    - вкладка сюжету,
    - сторінка налаштувань його елементів,
    - вкладка натхнення,
- допоміжні форми для додавання та видалення елементів.

Форма входу у застосунок (див. рис. 2.4) відповідає за авторизацію або реєстрацію користувачів.

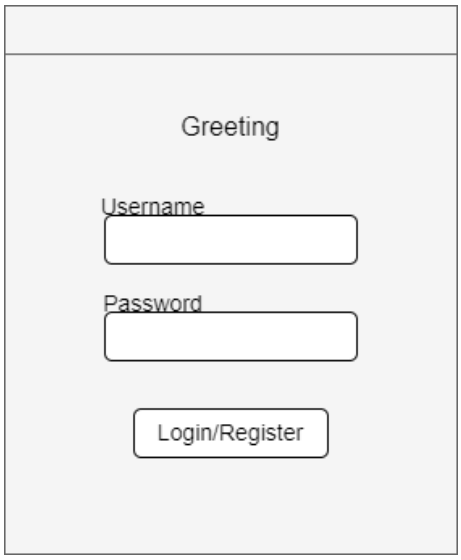

Рисунок 2.4 – Ескіз форми входу у застосунок

Форма основного вікна (див. рис. 2.5) має відображати доступні історії та надавати користувачу можливість їх додавати, а також здійснювати по ним пошук.

Рисунок 2.5 – Ескіз форми основного вікна

Враховуючи різні частини історії, з якими може працювати користувач, форма історії (див. рис. 2.6) буде мати меню з такими вкладками:

– загальна інформація,

- персонажі,
- світи,
- сюжет,
- натхнення.

Рисунок 2.6 – Ескіз форми історії

Розділ основної інформації буде містити лише основні дані про історію, такі як обкладинка, назва, опис, дата створення, та буде надавати користувачу можливість видалити історію (див. рис. 2.7).

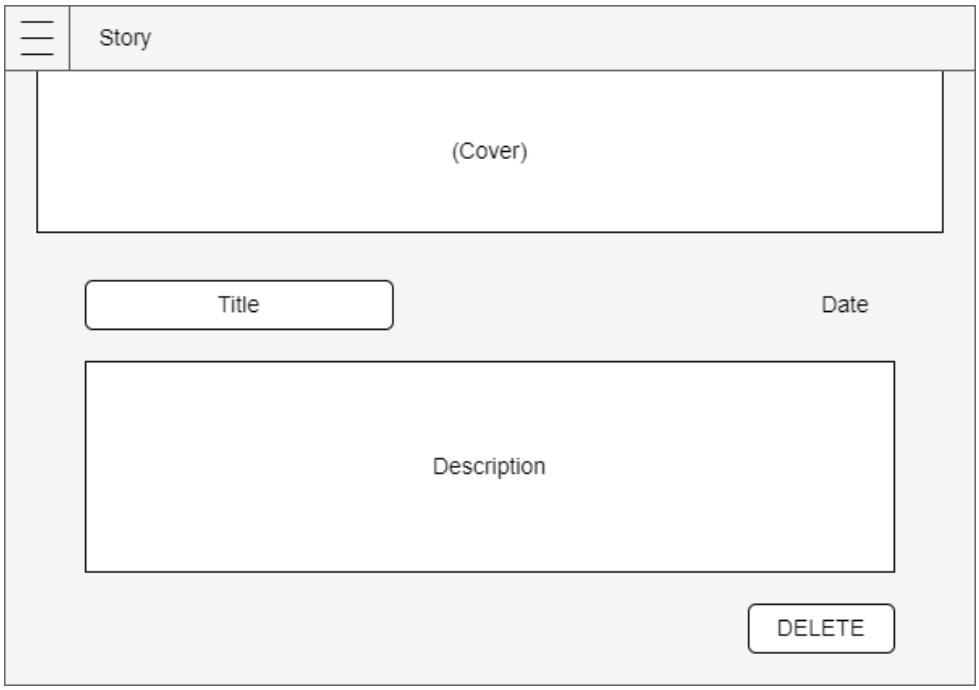

Рисунок 2.7 – Ескіз вкладки основної інформації

Розділ персонажів та світобудови є однаковими у їх будові. Тобто перейшовши в будь-який з розділів користувач побачить сторінку (див. рис. 2.8) з вже наявними екземплярами персонажів або світів та буде мати можливість додати новий.

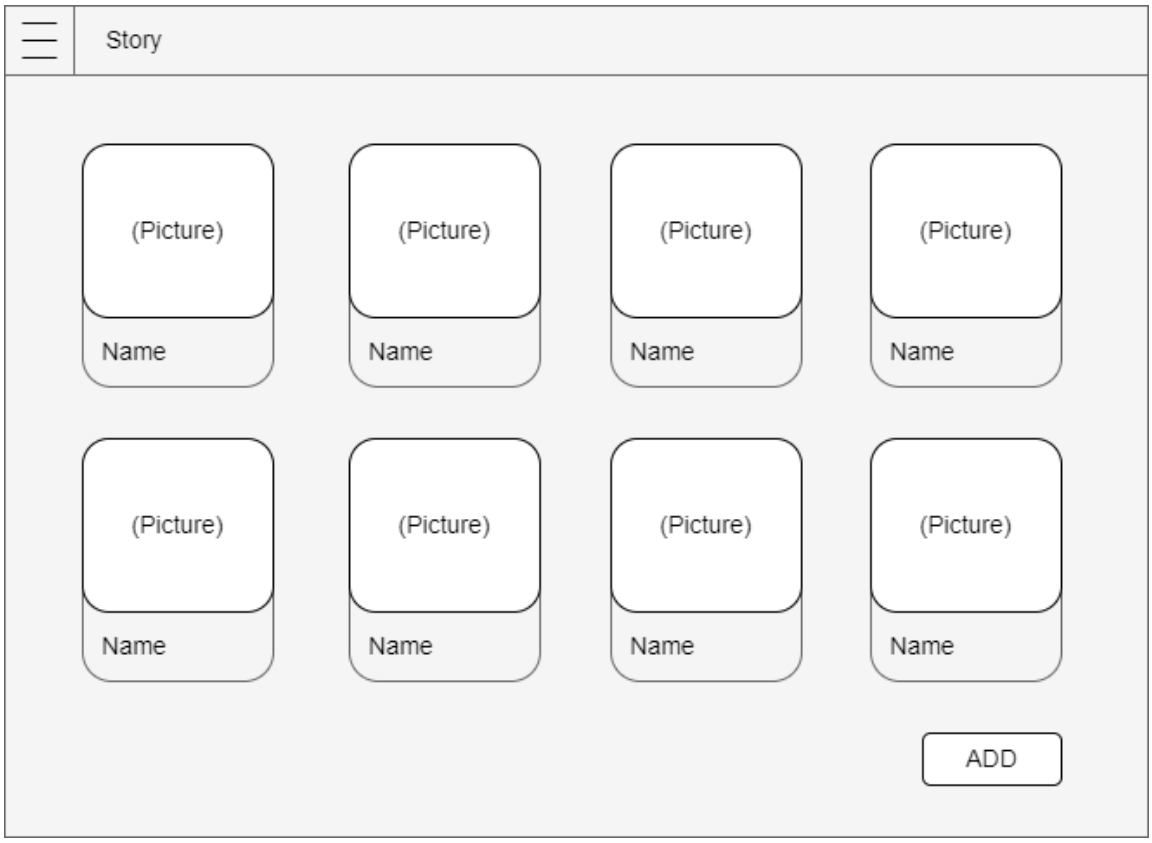

Рисунок 2.8 – Ескіз вкладки персонажів або світобудови

З цієї сторінки можна буде перейти на сторінку персонажу або історії (див. рис. 2.9) двома способами: вибір екземпляру персонажу або його додавання. У цій сторінці користувач буде бачити основну інформацію про персонажа або історію, а також мати можливість редагувати його.

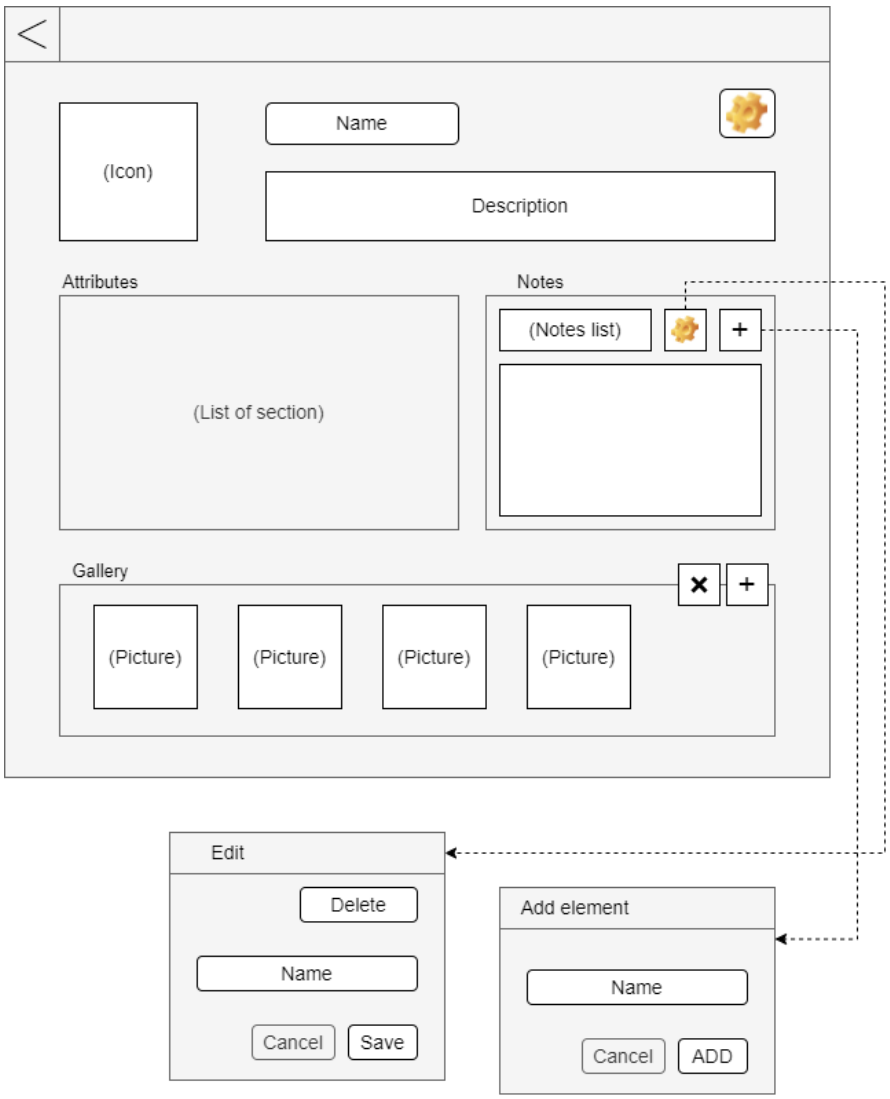

Рисунок 2.9 – Ескіз сторінки персонажу або світу

Список секцій буде складатись з повторюваних секцій (див. рис. 2.10).

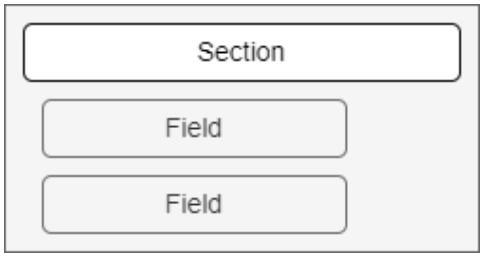

Рисунок 2.10 – Ескіз секції

Натиснувши на іконку у правому верхньому куту буде відкрите вікно налаштувань (див. рис. 2.11). Тут користувач зможе кастомізувати з якими саме секціями та полями він хоче працювати, зможе додавати нові та видаляти вже створені. Для цього буде присутні декілька форм для збору інформації від користувача, їх теж можна побачити на рисунку 2.11. Список секцій буде сформовано з раніше продемонстрованих секцій (див. рис. 2.10).

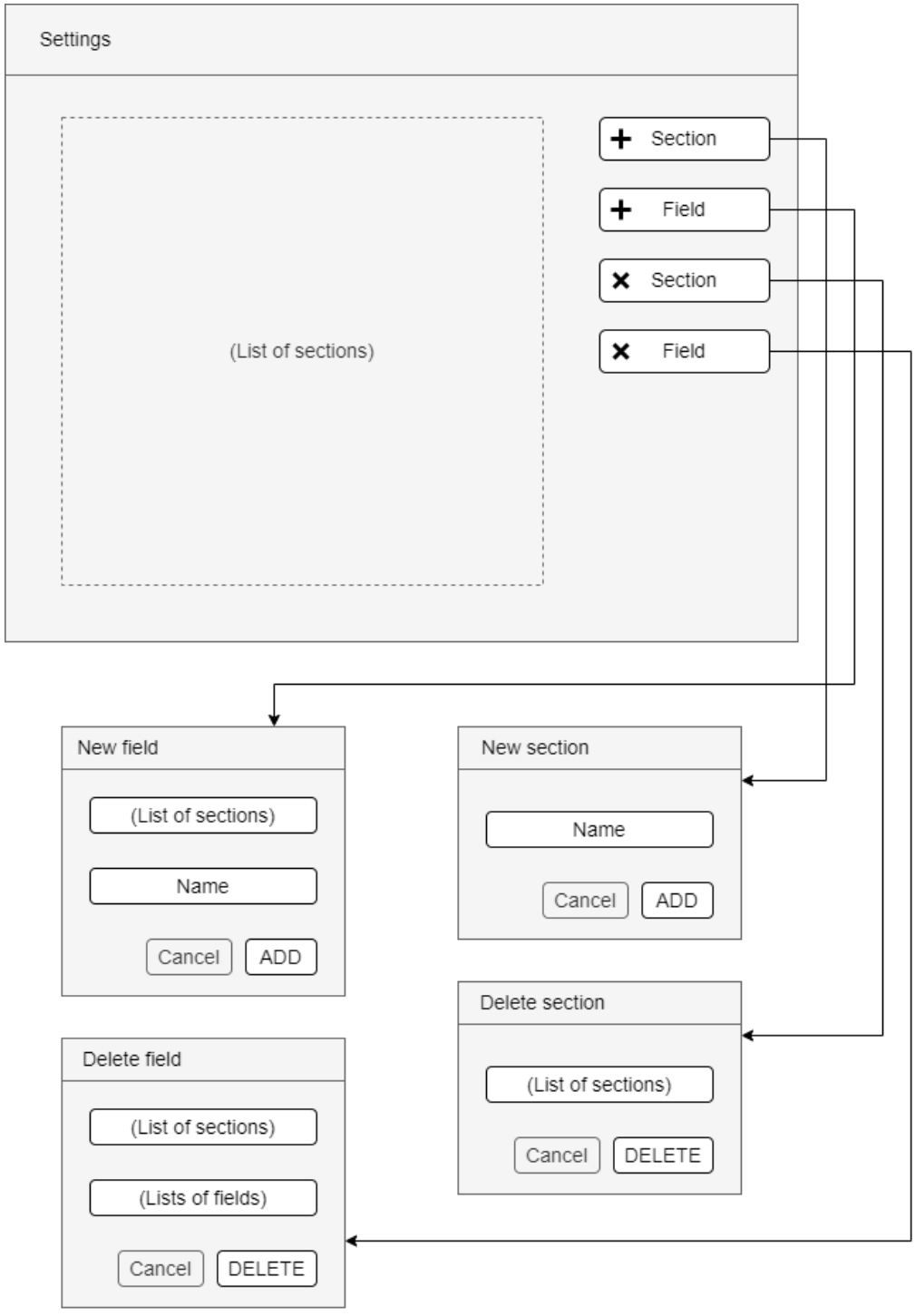

Рисунок 2.11 – Ескіз форми налаштувань персонажу або світу

Вкладка сюжету (див. рис. 2.12) надасть користувачу можливість працювати з різними елементами сюжету, писати та додавати глави, додавати до них теги та сцени, а також створювати часові стрічки та лінії подій в них.

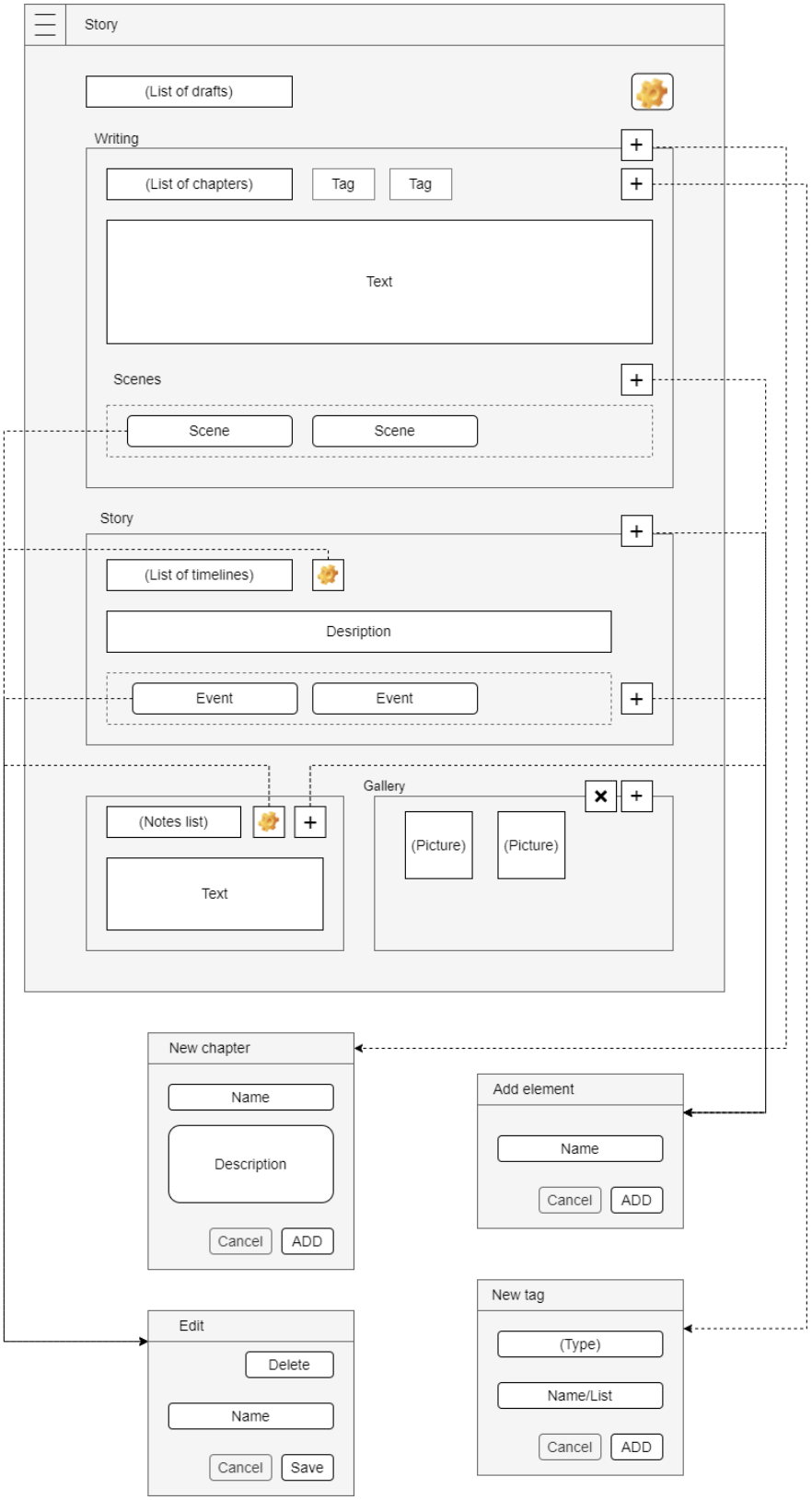

Рисунок 2.12 – Ескіз вкладки сюжету

Натиснувши на іконку у правому верхньому куту буде відкрите вікно налаштувань сюжету (див. рис. 2.13). Тут користувач редагувати та видаляти глави, для чого буде присутня форма для вибору необхідної глави.

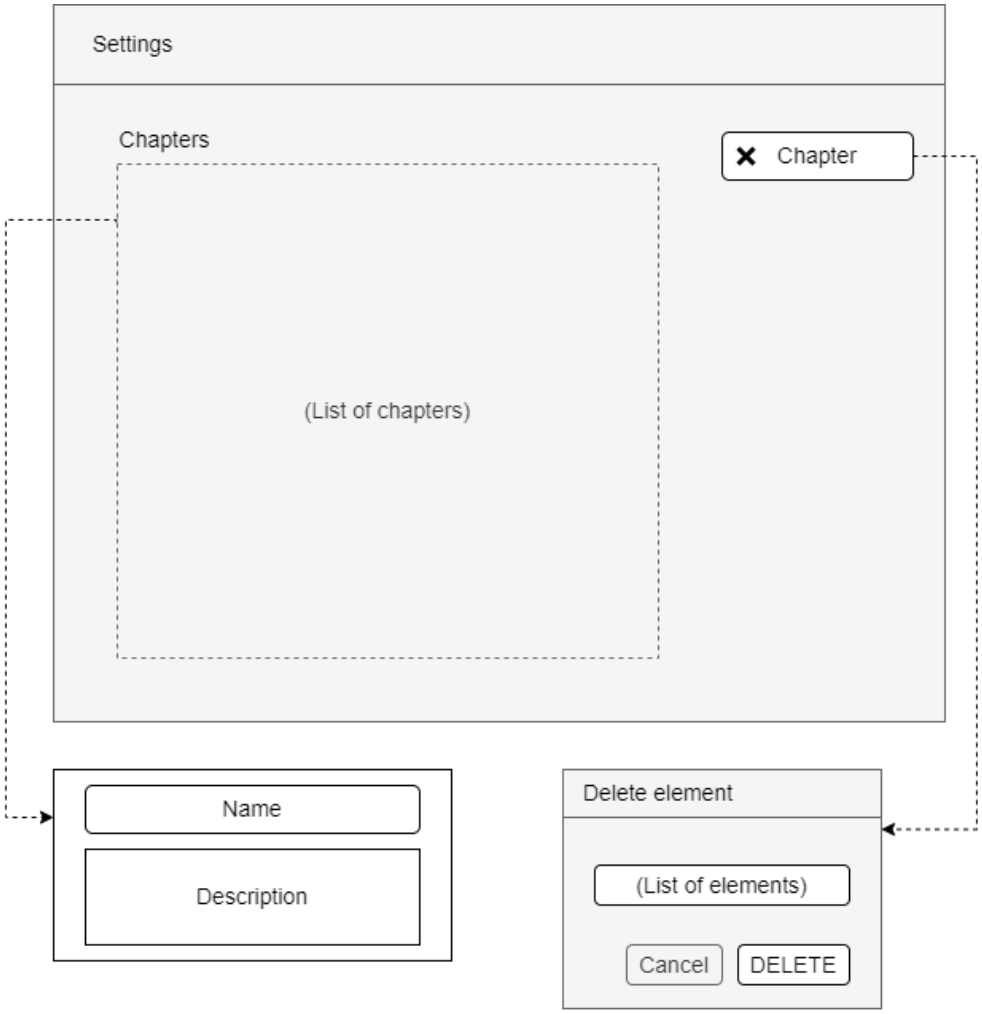

Рисунок 2.13 – Ескіз форми налаштувань історії

Вкладка натхнення (див. рис. 2.14) надасть користувачу можливість переглядати та розширювати галерею зображень та посилань для всієї історії.

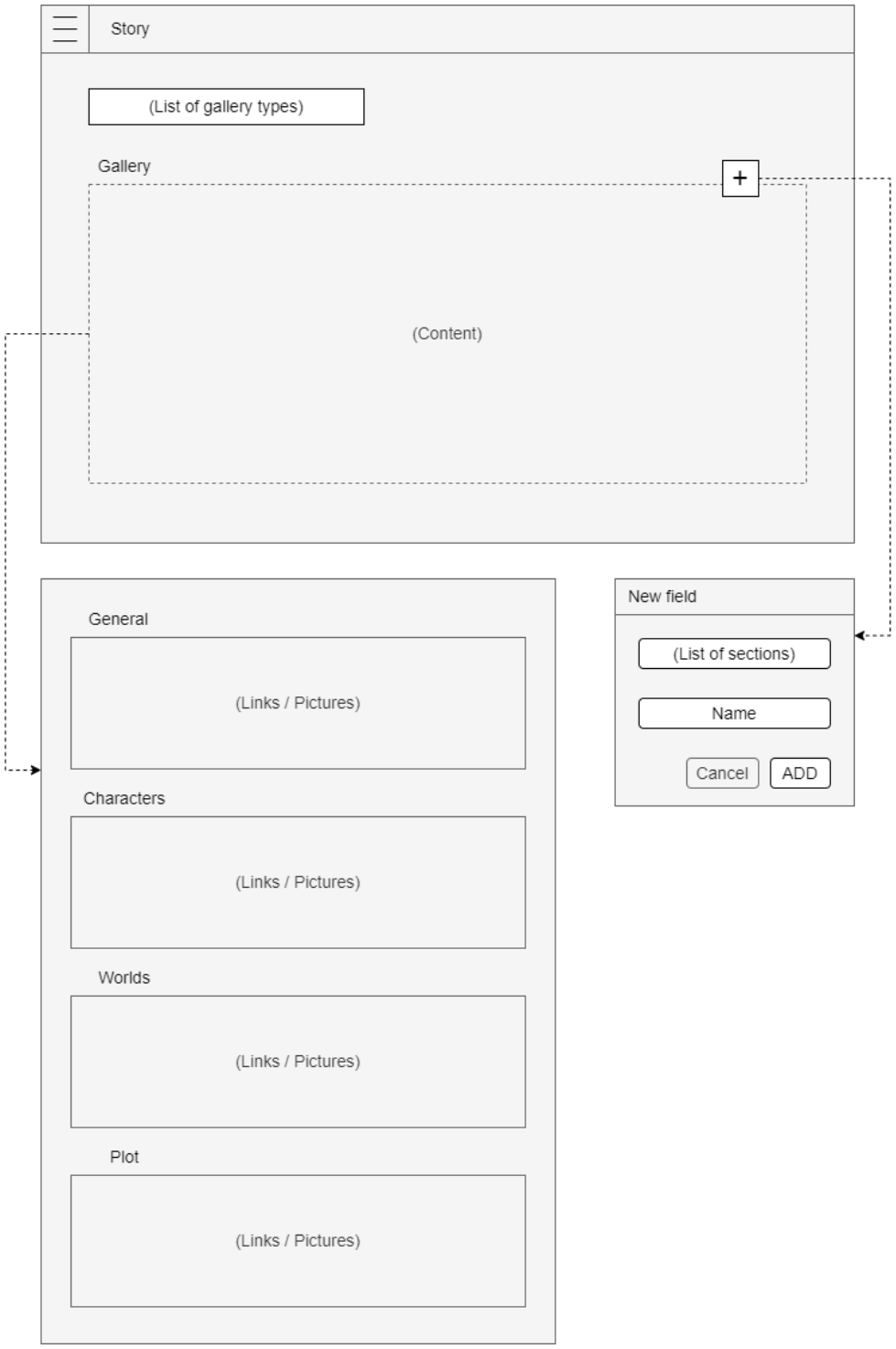

Рисунок 2.14 – Ескіз вкладки натхнення

2.2.4 Проєктування системи на фізичному рівні

На основі створеної та детально описаної логічної схеми бази даних (див. рис. 2.1) було створено фізичну схему бази даних (див. рис. 2.15).

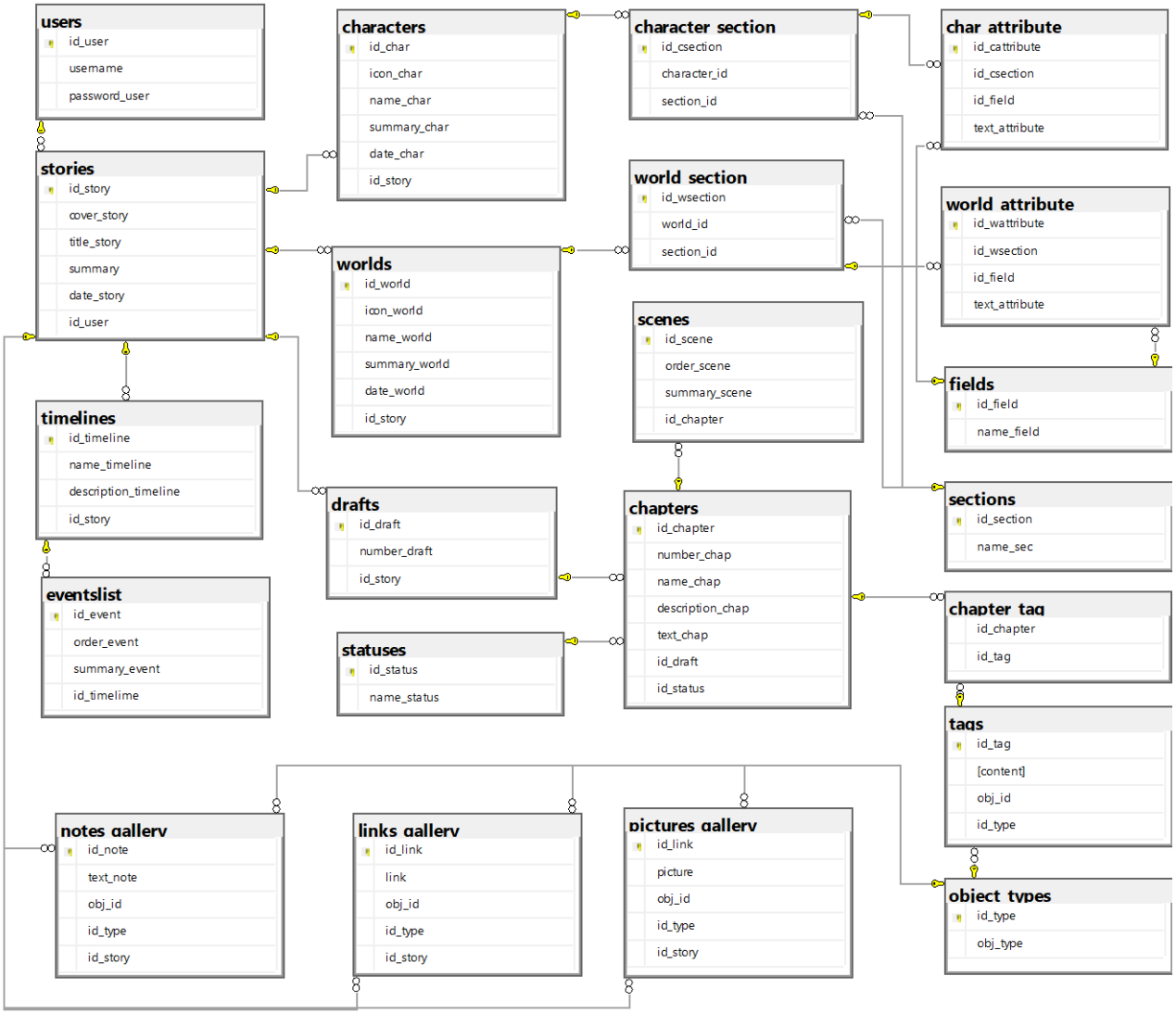

Рисунок 2.15 – Фізична схема бази даних

Також на рисунку 2.16 зображено діаграма компонентів для всієї розроблюваної системи. Writer.exe – це файл застосунку, який встановлює користувач. Він використовує бібліотеку System.Data.DLL, так як вона надає інструменти зв’язку та роботи з базою даних.

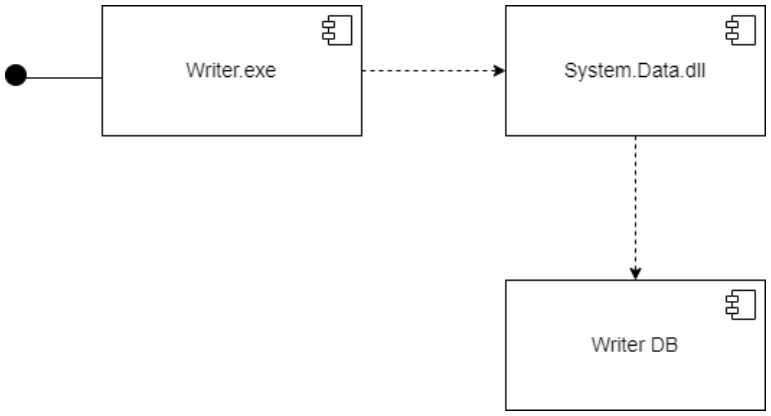

Рисунок 2.16 – Діаграма компонентів

2.2.5 Вибір мови та середовища розробки

Для розробки даного програмного забезпечення було обрано мову C# та середовище Visual Studio 2019. Такий вибір обґрунтований тим, що в Visual Studio присутня можливість створення проектів Windows Forms на платформі Framework .NET, які дозволяють зручно побудувати інтерфейс програми, а також надають доступ до бібілотек ADO.NET, які роблять можливим взаємодію з базою даних.

Для розробки бази даних для проекту було обрано Microsoft SQL Server, так як він має всі необхідні функції для збереження спроектованого обсягу інформації та зручної роботи з нею.

Висновки до розділу 2

У цьому розділі було визначено призначення проекту та вимоги до нього, спроектовано схеми бази даних, системи та їх взаємодії, а також було створено ескізи інтерфейсу користувача. Мову програмування та середовище розробки також були обрані базуючись на критеріях розробленої системи. Отже, на основі виконаної в цьому розділі роботи можна розпочинати реалізовувати програмне забезпечення, проводити його тестування та налагодження.

3 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМИ

3.1 Тестування методом «чорного скриньки»

Тестування за методом «чорного скриньки» передбачає відсутність доступу до програми користувачем. Для перевірки однієї з головних функцій програми (перевірка та збереження загальної інформації про історію) було розглянуто три методи для збереження, обробки та перевірки даних.

Функція 1: FormClosing.

Закриття форми, оновлення інформації, збереження історії або її видалення.

Заголовок private void UI_Story_FormClosing(object sender, FormClosingEventArgs e)

Вхідні дані у неявному вигляді:

- Deleted – флажок, що зазанчає чи була історія видалена
- isNew – флажок, що зазначає чи є історія новою чи існуючою,
- result – вибір користувача щодо дій з історією,
- TextBoxTitle – текстове поле назви історії.

Вихідні:

У явному вигляді функція не повертає значень.

Функція 2: SetFields.

Функція перевірки та встановлення наявних полей

Заголовок private void SetFields(SqlCommand command)

Вхідні дані:

У явному вигляді:

- Command – команда мови sql.

У неявному вигляді:

Image – зображення обкладенки історії,

Title:Text – текст назви історії,

Description:Text – текст опису історії.

Вихідні:

У явному вигляді функція не повертає значень.

Функція 3: PickingPicture.

Функція вибору обкладинки.

Заголовок private void pictureBox_Cover_DoubleClick(object sender, EventArgs e)

Вхідні дані у неявному вигляді:

- result – вибір зображення користувачем,
- imageCover – текстове поле назви історії.

Вихідні:

У явному вигляді функція не повертає.

Було застосовано метод еквівалентних розбиттів, створені класи еквівалетності можна побачити у таблиці 3.1.

Таблиця 3.1 – Класи еквівалентності

Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
Значення назви історії	Назва є набором символів від 1 до 50 (1)	Назва більше 50 символів (2)
		Назва відсутня (3)
Значення опису	Опис є набором символів від 0 до 500 (4)	Опис більше 500 символів (5)
Вибір зображення	Файл має допустиме розширення (png, jpg і тд) (6)	Файл має недопустиме розширення (7)
Вибір користувача щодо історії	Користувач хоче зберегти створену історію (8)	
	Користувач не хоче зберегти історію (9)	

Також було розроблено тести за даним методом (дис. табл. 3.2) для покриття всіх класів еквівалентності.

Таблиця 3.2 – Тести для методу еквівалентних розбиттів

Тест	Клас, що покривається	Вхідні дані	Очікуваний результат
1	1, 4, 6, 8	Title:Text – «short title», Description:Text – «this is a story», imageCover – «person.png», result – «yes»	Історія та її дані будуть збережені та вона буде показана у колекції
2	2, 5, 7, 9	Title:Text – «this text should be longer than 50 characters for sure» Description:Text – набір символів більше 500, imageCover – «program.cs», result – «no»	Набір тексту більше дозволеного буде неможливим; при виборі файлу з’явитсья повідомлення про помилку; історія буде видалена
3	3, 8	Title:Text – «», result – «yes»	З’явиться попередження про заповнення назви історії

Результати тестування наведені у таблиці 3.3.

Таблиця 3.3 – Результати тестування методом еквівалентних розбиттів

Тест	Результат
1	Історія з’явилась у колекції та всі дані були збережені
2	При введенні текст не заходив за межі припустимого, а при вставці тексту він обрізався; При виборі файлу не з тим розширенням з’явилось попередження, зображення не змінилось; Історія не з’явилась у колекції
3	З’явилось повідомлення про необхідність заповнення назви, вікно залишилось відкритим

3.2 Тестування методом «білої скриньки»

Тестування за методом «білої скриньки» передбачає доступ до коду програми. Для тестування було обрано метод покриття умов та рішень, так як він максимально покриває всі можливі сценарії. Для опису процесу тестування надається специфікація обраної функції, її код та деталі тестування.

Функція 1: FormClosing.

Закриття форми, оновлення інформації, збереження історії або її видалення.

Заголовок private void UI_Story_FormClosing(object sender, FormClosingEventArgs e)

Вхідні дані у неявному вигляді:

- Deleted – флажок, що зазанчає чи була історія видалена,
- isNew – флажок, що зазначає чи є історія новою чи існуючою,
- result – вибір користувача щодо дій з історією,
- TextBoxTitle – текстове поле назви історії.

Вихідні:

У явному вигляді функція не повертає значень.

Текст функції:

```
private void UI_Story_FormClosing(object sender, FormClosingEventArgs e)
      {
         if (deleted) {
            controllerForms.MainShow(); }
         else
         {
            if (isNew)
            {
               DialogResult saveRes = MaterialMessageBox.Show($"Do you want to save the story you created?",
                  "Saving", MessageBoxButtons.YesNo);
               if (saveRes == DialogResult.No)
               {
```

```
                    DeleteStory();
                    controllerForms.MainShow();
                    return;
                }
            }
            if (TextBox_Title.TextLength == 0)
            {
                DialogResult titleRes = MaterialMessageBox.Show("Fill out the title or you won't save the story!",
                    "Warning!", MessageBoxButtons.OKCancel);
                if(titleRes == DialogResult.Cancel)
                {
                    DialogResult deleteRes = MaterialMessageBox.Show("Maybe you want to delete the story instead?", "Deleting", MessageBoxButtons.YesNo);
                    if(deleteRes == DialogResult.Yes)
                    {
                        DeleteStory();
                        controllerForms.MainShow();
                        return;
                    }
                }
                e.Cancel = true;
            }
            else
            {
                UpdateStory();
                controllerForms.MainShow();
            }
        }
    }
```

Було підраховано, що у функції міститься 6 умов, а саме delete, isNew, saveRes == DialogResult.No, TextBox_Title.TextLength == 0, res == DialogResult.Cancel, res == DialogResult.Yes.

Для покриття всіх умов були створено 6 тестів.

Тест 1: історія вже видалена.

Вхідні дані: deleted = true.

Очікуваний результат: форма закрита.

Тест 2: історія є новою, але її не бажають зберегти.

Вхідні дані: deleted = false, isNew = true; saveRes = DialogResult.No.

Очікуваний результат: історія видалена.

Тест 3: історія є новою та має назву, її хочуть зберегти.

Вхідні дані: deleted = false, isNew = true; saveRes = DialogResult.Yes, TextBox_Title.TextLength = «name».

Очікуваний результат: історія додана до колекції.

Тест 4: стара історія без назви, але її хочуть ввести

Вхідні дані: deleted = false, isNew = false; TextBox_Title.TextLength = «», titleRes = DialogResult.OK.

Очікуваний результат: історія оновлено.

Тест 5: стара історія без назви, також було натиснуто відміну, але видаляти історію не хочуть.

Вхідні дані: deleted = false, isNew = false; TextBox_Title.TextLength = «», titleRes = DialogResult.Cancel, DeleteRes = DialogResult.No.

Очікуваний результат: продовження роботи у вікні.

Тест 6: стара історія без назви, яку хочуть видалити.

Вхідні дані: deleted = false, isNew = false; TextBox_Title.TextLength = «», titleRes = DialogResult.Cancel, DeleteRes = DialogResult.Yes.

Очікуваний результат: історія видалена.

З таблиці 3.4. видно, що можливі результати кожної умови у рішенні були отримані принаймні раз, як і всі рішення.

Таблиця 3.4 – Результати тестування методом покриття умов та рішень

Умова/Тест		1	2	3	4	5	6
Delete	+	*					
	-		*	*	*	*	*
isNew	+		*	*			
	-				*	*	*
saveRes == DialogResult.No	+		*				
	-			*			
TextBox_Title.TextLength == 0	+				*	*	*
	-			*			
titleRes == DialogResult.Cancel	+					*	*
	-				*		
deleteRes == DialogResult.Yes	+					*	
	-						*

3.3 Налагодження програми

Під час розробки програми неодноразово виникали помилки різного характеру, які можна поділити на декілька типів.

Перший тип – це проблеми роботи з базою даних, які поділялись на семантичні, коли опис команди був виконаний не вірно, і логічні, коли сам запит був неправильно сформований, а тому отримувались не ті дані, що очікувались. Другий тип, який теж іноді зустрічався, – це логічні проблеми з самим кодом програми.

Наприклад, при виконанні тестування методам «білої скриньки» під час виконання тесту №2 виникла помилка при видаленні історії із бази даних. Було виявлено, що помилка ставалась через те, що до сюжету історії було додано чернетку, яка пов’язувалась з історією, але попереднього її видалення перед видаленням історії не було реалізовано, тому виникала помилка через обмеження ключів. Цю помилку було вирішено доданням необхідного запиту для попереднього видалення чернеток та іншої прив’язаної інформації.

Загалом налагодження програми відбувалась за допомогою методів індукції та зворотнього відстеження, іноді запити до бази даних перевірялись напряму у Microsoft SQL Server.

Висновки до розділу 3

У цьому розділі було проведено тестування однієї з основних функцій програми – можливості створення, збереження та видалення історії та інформації про неї. Була виявлена проблема під час тестування системи методом «білої скриньки» та було виконано налагодження програми.

ВИСНОВКИ

Наразі існує багато варіацій програмного забезпечення для допомоги з створенням та написанням творів. Воно і не дивно, письмо – це дуже складний та індивідуальний процес, який потребує уяви, креативу та інколи упорядкування. Деякі застосунки слугують лише як текстові редактори, інші ж дозволяють планувати створення кастомізованої історії, але не надають можливості її написання, а є навіть і такі, що дозволяють детально описувати твір, але залишають мало місця для індивідуальних змін. Іншими словами важко знайти те, що може дійсно облегшити творчий процес і водночас сповна підлаштуватись під креативність людини. Виходячи з цього була визначена потреба у програмному забезпеченні для упорядкування написання твору та надання користувачу можливості вибору.

В процесі виконання даної дипломної роботи було спроектовано базу даних та систему необхідну для реалізації застосунку. На основі виконаних планувань було розроблено проєкт Windows Forms на мові C# та на основі .NET Framework, який взаємодіє з реалізованою базою даних Microsoft SQL Server. Програма була протестована й налагоджена, вона успішно виконує свої основні функції.

Як результат було створено застосунок для організації процесу формування історії, її елементів, та написання сюжету. Розроблене програмне забезпечення надає користувачу можливість створювати, редагувати та видаляти твори, будувати та описувати для них власних персонажів та світи, а також створювати різні елементи сюжету та записувати вміст глав одразу у застосунку.

Під час роботи з персонажами та світобудовою письменник має можливість сам обирати, які елементи опису вони будуть мати та що в них буде міститись. Тобто створивши, наприклад, дієву особу, користувачу може не підійти стандартний її опис, тому він буде в змозі видалити непотрібні поля та секції та охарактеризувати його за допомогою створення нових рис.

Написання історії відбувається у розділі сюжету, де письменник може додавати глави до чернетки, описувати оповідання за допомогою таких елементів як сцени, лінії часу та події, та найголовніше втілювати ідеї у життя.

ЛІТЕРАТУРА

1. KU Writing Center | KU Writing Center. – Режим доступу: https://writing.ku.edu/sites/writing/files/images/Writing_Guide_Rev_2022/Writing%20Process.png (дата звернення: 15.06.2023). – Назва з екрана.
2. 5 Steps To Creativity In Writing [Електронний ресурс] // Writers Write. – Режим доступу: https://www.writerswrite.co.za/creativity-in-writing/ (дата звернення: 19.06.2023). – Назва з екрана.
3. Resources for Writers: The Writing Process [Електронний ресурс] // MIT Comparative Media Studies/Writing. – Режим доступу: https://cmsw.mit.edu/writing-and-communication-center/resources/writers/writing-process/ (дата звернення: 19.06.2023). – Назва з екрана.
4. 5 Elements of a Story Explained (Free Worksheet) | Imagine Forest [Електронний ресурс] // Imagine Forest Blog. – Режим доступу: https://www.imagineforest.com/blog/elements-of-a-story/ (дата звернення: 20.06.2023). – Назва з екрана.
5. lucycrichton. The Seven Elements Of A Successful Story - The People's Friend [Електронний ресурс] / lucycrichton // The People's Friend. – Режим доступу: https://www.thepeoplesfriend.co.uk/2018/09/13/the-seven-elements-of-a-successful-story/ (дата звернення: 20.06.2023). – Назва з екрана.
6. What Are the Key Elements of a Story? [Електронний ресурс] // Prodigy. – Режим доступу: https://www.prodigygame.com/main-en/blog/story-elements/ (дата звернення: 20.06.2023). – Назва з екрана.
7. Schade A. Customization vs. Personalization in the User Experience [Електронний ресурс] / Amy Schade // Nielsen Norman Group. – Режим доступу: https://www.nngroup.com/articles/customization-personalization/ (дата звернення: 20.06.2023). – Назва з екрана.

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖЕНО
1116130.01287-01-ЛЗ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КРЕАТИВНОГО ПИСЬМА ПРИ ФОРМУВАННІ ПЕРСОНАЖІВ ТА СЮЖЕТУ АВТОРСЬКИХ ІСТОРІЙ

Технічне завдання

1116130.01287-01

Листів 11

2023

ЗМІСТ

1 ВВЕДЕННЯ

Програмне забезпечення для креативного письма при формуванні персонажів та сюжету авторських історій, призначене для упорядкування процесу створення історії та надання користувачу можливості вибору елементів для її опису.

Так як письмо є складною задачею, що потребує роботи з великою кількістю даних, з’явилась необхідність у створенні зручного застосунку для збереження необхідної для користувача інформації, формування та написання історії.

Користуватися даним програмним продуктом може будь-хто, хто є зацікавленим у письмі.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.22 №1209ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

Тема дипломної роботи – “Програмне забезпечення для креативного письма при формуванні персонажів та сюжету авторських історій”. Керівник - ст. викладач Шаповал І. В.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – організація процесу формування історії, її елементів та написання сюжету. Сюди відноситься упорядкування роботи, думок та інформації при креативному письмі.

Експлуатаційне призначення – облегшення процесу письма завдяки збереженню необхідної для користувача інформації про персонажів, світ та сюжет історії та зменшення часових витрат на роботу з інформацією.

4 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

Програмний продукт має надавати користувачу можливість створювати, редагувати та видаляти твори, будувати та описувати для них власних персонажів та світи, а також створювати різні елементи сюжету та записувати вміст глав одразу у застосунку

4.2 Вимоги до надійності

Вимоги до надійності наступні: контроль вхідної і вихідної інформації; збереження даних на сервері.

4.3 Вимоги експлуатації

Вимоги до кліматичних умов: температура – 21-25 С, відносна вологість 40-60%. Обслуговування не потрібне.

Працювати з програмним забезпеченням можу будь-хто, хто має досвід роботи із ПК.

4.4 Вимоги до складу та параметрів технічних засобів

Склад технічних засобів для використання даного продукту: процесор з тактовою частотою 2 ГГЦ або вище; 2 Гб. оперативної пам’яті; клавіатура.

4.5 Вимоги до інформаційної та програмної сумісності

Програма має функціонувати в середовищі ОС Windows 10\11, в якому має бути встановлений .NET Framework 4.5 або вище.

4.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги.

На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7 Вимоги до транспортування та зберігання

Транспортування повинно проводитись в упаковці та забезпечувати цілісність продукту.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- текст програми.

Програмна документація повинна відповідати вимогам ДСТУ [1].

.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця А.1 – Стадії та етапи розробки

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	01.04.2023 – 06.04.2023	
2	Огляд літератури та аналіз аналогів	07.04.2023 – 17.04.2023	
3	Розробка структур вхідних і вихідних даних, вимог до системи	18.04.2023 – 30.04.2023	
5	Узгодження та затвердження ТЗ, постановка задачі	01.05.2023 – 07.05.2023	30 %
6	Розробка та програмування логіки програми	08.05.2023 – 17.05.2023	
7	Розробка і реалізація інтерфейсу користувача	18.05.2023 – 21.05.2023	
8	Відлагодження програми	22.05.2023 – 28.05.2023	60%
9	Розробка, узгодження та затвердження програмної документації	29.05.2023 – 11.06.2023	
10	Розробка демонстраційних матеріалів	12.06.2023 – 18.06.2023	100%
	Подання кваліфікаційної роботи до кафедри	18.06.2023 – 25.06.2023	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.2023	

7 ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль виконання здійснює керівник розробки Шаповал І.В. Прийом здійснюється уповноваженою комісією.

8 БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
17.12.22

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КРЕАТИВНОГО ПИСЬМА ПРИ ФОРМУВАННІ ПЕРСОНАЖІВ ТА СЮЖЕТУ АВТОРСЬКИХ ІСТОРІЙ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01287-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.06.23

Керівник розробки
Ірина ШАПОВАЛ
18.06.23

Виконавець
Маргарита ВИСКАРКА
18.06.23

Нормоконтролер
Світлана ВОЛКОВА
18.06.223

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
17.12.22

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КРЕАТИВНОГО ПИСЬМА ПРИ ФОРМУВАННІ ПЕРСОНАЖІВ ТА СЮЖЕТУ АВТОРСЬКИХ ІСТОРІЙ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01287-01 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.06.23

Керівник розробки
Ірина ШАПОВАЛ
18.06.23

Виконавець
Маргарита ВИСКАРКА
18.06.23

Нормоконтролер
Світлана ВОЛКОВА
18.06.223

2023

ДОДАТОК В
Текст програми

ЗАТВЕРДЖЕНО
1116130.01287-01 12 01-ЛЗ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КРЕАТИВНОГО ПИСЬМА ПРИ ФОРМУВАННІ ПЕРСОНАЖІВ ТА СЮЖЕТУ АВТОРСЬКИХ ІСТОРІЙ

Текст програми

1116130.01287-01 12 01

Листів 44

2023

Текст основної програми:

Controllers.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace Writer
{
   class ControllerForms
   {
      private static UI_Login I_Login;
      private static UI_MainPage
I_Main;
      private static UI_Story I_Story;

      /// \brief Creating form UI_Login
      public void LoginForm()
      {
         I_Login = new UI_Login();
         I_Login.ShowDialog();
      }

      /// \brief Creating form
UI_MainPage
      public void MainForm()
      {
         I_Main = new UI_MainPage();
         I_Main.ShowDialog();
      }

      /// \brief Creating form UI_Story
      public void StoryForm(bool
isNew)
      {
         I_Story = new
UI_Story(isNew);
         I_Story.ShowDialog();
      }

      /// \brief Showing form UI_Login
      public void LoginShow()
      {
         I_Login.Show();
      }

      /// \brief Showing form
UI_MainPage
      public void MainShow()
      {
         I_Main.Show();
      }

      /// \brief Showing form UI_Story
      public void StoryShow()
      {
         I_Story.Show();
      }

      public void StoryHide()
      {
         I_Story.Visible = false;
      }

   }

   class Controller
   {
      private static DataBase database =
new DataBase();
      private ImageProcessing
imageProcess = new
ImageProcessing();
      private static List<Section>
charSectionsList = new
List<Section>();
      private static List<Section>
worldSectionsList = new
List<Section>();
      private static List<Draft>
draftsList = new List<Draft>();
      private static int userID;
      private static int storyID;
```

```
        private static int characterID;
        private static int worldID;

        public DataBase Database
        {
            get { return database; }
        }

        public int UserID
        {
            get { return userID; }
            set { userID = value; }
        }

        public int StoryID
        {
            get { return storyID; }
            set { storyID = value; }
        }

        public int CharacterID
        {
            get { return characterID; }
            set { characterID = value; }
        }

        public int WorldID
        {
            get { return worldID; }
            set { worldID = value; }
        }

        public List<Section>
CSectionsList
        {
            get { return charSectionsList; }
            set { charSectionsList = value;
}
        }

        public ImageProcessing
ImageProcess
        {
            get { return imageProcess; }
        }

        public SqlCommand
MakeCommand(string query)
        {
            SqlCommand command = new
SqlCommand(query,
database.GetConnection());
            return command;
        }

        public void SetSectionsList(bool
isChar)
        {
            Sections sections = new
Sections();

            if(isChar)
            {
                charSectionsList =
sections.getCharSections();
            }
            else
            {
                worldSectionsList =
sections.getWorldSections();
            }
        }

        public void SetDrafts()
        {
            Drafts drafts = new Drafts();
            draftsList =
drafts.getDraftsList();
        }

    }
}
```

DataBase.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Data;
using System.Data.SqlClient;

namespace Writer
{
   class DataBase
   {
      private SqlConnection connect =
new SqlConnection(@"Data Source =
DESKTOP-KL9BC6I\SQLEXPRESS;
Initial Catalog = Writer; Integrated
Security =True");

      public void OpenConnection()
      {
         if(connect.State ==
System.Data.ConnectionState.Closed)
         {
            connect.Open();
         }
      }

      public void CloseConnection()
      {
         if (connect.State ==
System.Data.ConnectionState.Closed)
         {
            connect.Open();
         }
      }

      public SqlConnection
GetConnection()
      {
         return connect;
      }

      public DataTable GetTable(string
query)
      {
         SqlCommand command = new
SqlCommand(query,
this.GetConnection());
         SqlDataAdapter adapter = new
SqlDataAdapter();
         DataTable table = new
DataTable();
         adapter.SelectCommand =
command;
         adapter.Fill(table);
         return table;
      }
   }

}
```

Drafts.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using System.Data.SqlClient;
using System.Text;
using System.Threading.Tasks;

namespace Writer
{
   public class Drafts
   {
      private Controller controller =
new Controller();
      private List<Draft> draftsList =
new List<Draft>();

      public List<Draft> getDraftsList()
      {
         draftsList = new List<Draft>();

         //Loading drafts
         string query = $"SELECT
id_draft, number_draft FROM drafts
WHERE id_story =
'{controller.StoryID}'";
         DataTable dtDrafts =
controller.Database.GetTable(query);

         foreach (DataRow rowDraft in
dtDrafts.Rows)
```

```
        {
            Draft draft = new Draft();

            //Loading chapters for each
draft
            query = $"SELECT
id_chapter, number_chap, name_chap,
description_chap, text_chap FROM
chapters " +
                $"WHERE id_draft =
'{(int)rowDraft["id_draft"]}'";
            DataTable dtChapters =
controller.Database.GetTable(query);

            draft.ID =
(int)rowDraft["id_draft"];

            draft.Number =
(int)rowDraft["number_draft"];

            foreach (DataRow
rowChapter in dtChapters.Rows)
            {
                Chapter chapter = new
Chapter((int)rowChapter["number_cha
p"], (int)rowChapter["id_chapter"],
rowChapter["name_chap"].ToString(),

rowChapter["description_chap"].ToStri
ng(),
rowChapter["text_chap"].ToString()) ;
                draft.AddChapter(chapter);
            }

            draftsList.Add(draft);

        }

        return draftsList;
    }
}

public class Draft
{
    private int number;
    private int id;

    private List<Chapter>
chaptersList = new List<Chapter>();

    public List<Chapter>
getChaptersList()
    {
        chaptersList = new
List<Chapter>();
        return chaptersList;
    }

    public int Number
    {
        get { return number; }
        set { number = value; }
    }

    public int ID
    {
        get { return id; }
        set { id = value; }
    }

    public void AddChapter(Chapter
chapter)
    {
        chaptersList.Add(chapter);
    }

}

public class Chapter
{
    private int number;
    private int id;
    private string name;
    private string description;
    private string text;
```

```
        public Chapter(int num, int id,
string name, string description, string
text)
        {
            number = num;
            this.id = id;
            this.name = name;
            this.description = description;
            this.text = text;
        }

        public int Number
        {
            get { return number; }
            set { number = value; }
        }

        public string Text
        {
            get { return text; }
            set { text = value; }
        }

        public int ID
        {
            get { return id; }
            set { id = value; }
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        public string Description
        {
            get { return description; }
            set { description = value; }
        }

    }

}
```

Sections.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using System.Data.SqlClient;
using System.Text;
using System.Threading.Tasks;

namespace Writer
{
    public class Sections
    {
        Controller controller = new
Controller();
        List<Section> sectionsList = new
List<Section>();

        public List<Section>
getCharSections()
        {
            //Loading sections
            string query = $"SELECT
sections.name_sec,
character_section.id_csection FROM
sections INNER JOIN
character_section " +
                $"ON
character_section.section_id =
sections.id_section AND
character_section.character_id =
'{controller.CharacterID}'";
            DataTable dtSections =
controller.Database.GetTable(query);

            foreach (DataRow rowSection
in dtSections.Rows)
            {
                Section section = new
Section();

                //Loading fields for each
section
```

```
            query = $"SELECT
char_attribute.id_cattribute,
fields.name_field,
char_attribute.text_attribute FROM
char_attribute INNER JOIN fields " +
                $"ON
char_attribute.id_field = fields.id_field
AND char_attribute.id_csection =
'{(int)rowSection["id_csection"]}'";
            DataTable dtFields =
controller.Database.GetTable(query);

            section.ID =
(int)rowSection["id_csection"];

            section.Name =
rowSection["name_sec"].ToString();

            foreach (DataRow rowField
in dtFields.Rows)
            {
              Field field = new
Field((int)rowField["id_cattribute"],
rowField["name_field"].ToString(),
rowField["text_attribute"].ToString());
                section.AddField(field);
            }

            sectionsList.Add(section);

        }

        return sectionsList;
    }

    public List<Section>
getWorldSections()
    {
        // TODO: add proper code fro
world
        //Loading sections
        string query = $"SELECT
sections.name_sec,
character_section.id_csection FROM
sections INNER JOIN
character_section " +
            $"ON
character_section.section_id =
sections.id_section AND
character_section.character_id =
'{controller.CharacterID}'";
        DataTable dtSections =
controller.Database.GetTable(query);

        foreach (DataRow rowSection
in dtSections.Rows)
        {
            Section section = new
Section();

            //Loading fields for each
section
            query = $"SELECT
char_attribute.id_cattribute,
fields.name_field,
char_attribute.text_attribute FROM
char_attribute INNER JOIN fields " +
                $"ON
char_attribute.id_field = fields.id_field
AND char_attribute.id_csection =
'{(int)rowSection["id_csection"]}'";
            DataTable dtFields =
controller.Database.GetTable(query);

            section.ID =
(int)rowSection["id_csection"];

            section.Name =
rowSection["name_sec"].ToString();

            foreach (DataRow rowField
in dtFields.Rows)
            {
                Field field = new
Field((int)rowField["id_cattribute"],
rowField["name_field"].ToString(),
rowField["text_attribute"].ToString());
                section.AddField(field);
```

```
            }

            sectionsList.Add(section);

        }

        return sectionsList;
    }


}

public class Section
{
    private int id;
    private string name;
    private List<Field> fields = new
List<Field>();

    public List<Field> fieldsList
    {
        get { return fields; }
        set { fields = value; }
    }

    public int ID
    {
        get { return id; }
        set { id = value; }
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public void AddField(Field field)
    {
        fields.Add(field);
    }

}

public class Field
{
    private int id;
    private string name;
    private string text;

    public Field(int id, string name,
string text)
    {
        this.id = id;
        this.name = name;
        this.text = text;
    }

    public string Text
    {
        get { return text; }
        set { text = value; }
    }

    public int ID
    {
        get { return id; }
        set { id = value; }
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
}
```

ImageProcessing.cs:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace Writer
{
```

```
   class ImageProcessing
   {

      public Image
BinaryToImage(byte[] data)
      {
         using (MemoryStream ms =
new MemoryStream(data))
         {
            return
Image.FromStream(ms);
         }
      }

      public byte[]
ImageToBinary(Image img)
      {
         using (MemoryStream ms =
new MemoryStream())
         {
            img.Save(ms,
img.RawFormat);
            return ms.ToArray();
         }
      }

   }
}
```

UI_Login.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class UI_Login :
MaterialForm
   {
      private ControllerForms
controllerForms = new
ControllerForms();
      private Controller controller =
new Controller();
      private DataTable dt;

      public UI_Login()
      {
         InitializeComponent();

         //Setting design
         var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
         materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);

      }

      private void
UI_Login_Load(object sender,
EventArgs e)
      {
         textBox_Username.MaxLength
= 50;
         textBox_Password.MaxLength
= 50;
         btnSignUp.Visible = false;
         pictureBox_Invisible.Visible =
false;
      }
```

```
        private void
btnLogIn_Click(object sender,
EventArgs e)
        {
            var username =
textBox_Username.Text;
            var password =
textBox_Password.Text;

            if (checkUser(username,
password, false) == true)
            {
                this.Hide();
                //Setting this user as current
user
                controller.UserID =
(int)dt.Rows[0][0];
                controllerForms.MainForm();
            }
            else
            {

MaterialMessageBox.Show("Check
your username or password! Otherwise
sign up!", "Wrong information!");
            }

        }

        private void
btnRegister_Click(object sender,
EventArgs e)
        {
            textBox_Password.Clear();
            textBox_Username.Clear();
            btn_Return.Visible = true;
            btnRegister.Visible = false;
            btnLogIn.Visible = false;
            btnSignUp.Visible = true;
            textBox_Password.Password =
false;
        }

        private void
BtnSignUp_Click(object sender,
EventArgs e)
        {
            var username =
textBox_Username.Text;
            var password =
textBox_Password.Text;


if(textBox_Username.Text.Length ==
0 || textBox_Password.Text.Length ==
0)
            {

MaterialMessageBox.Show("Fill out
both fields!", "Check input");
                return;
            }

            if (checkUser(username,
password, true) == true)
            {

MaterialMessageBox.Show("Such user
already exists!", "Exists!");
            }
            else
            {
                SqlCommand command =
controller.MakeCommand($"insert into
users (username, password_user)
values('{username}', '{password}')");


controller.Database.OpenConnection();
                if
(command.ExecuteNonQuery() == 1)
                {

MaterialMessageBox.Show("You
signed up!", "Success!");

textBox_Username.Clear();
```

```
                textBox_Password.Clear();
                btnRegister.Visible = true;
                btnLogIn.Visible = true;
                btnSignUp.Visible = false;
            }
            else
            {

MaterialMessageBox.Show("You
didn't signed up!", "Fail!");
            }

controller.Database.CloseConnection();
        }
    }

    private Boolean checkUser(string
username, string password, bool
registering)
    {
        string query;
        if (registering)
        {
            query = $"select id_user,
username, password_user from users
where username = '{username}'";
        }
        else
        {
            query = $"select id_user,
username, password_user from users
where username = '{username}' " +
                $"and password_user =
'{password}'";
        }

        dt =
controller.Database.GetTable(query);
        if (dt.Rows.Count == 1)
        {
            return true;
        }
        else return false;
    }

    private void
UI_Login_FormClosed(object sender,
FormClosedEventArgs e)
    {
        Application.Exit();
    }

    private void
pictureBox_Visible_Click(object
sender, EventArgs e)
    {
        pictureBox_Invisible.Visible =
true;
        pictureBox_Visible.Visible =
false;
        textBox_Password.Password =
false;
        textBox_Password.Refresh();
    }

    private void
pictureBox_Invisible_Click(object
sender, EventArgs e)
    {
        pictureBox_Invisible.Visible =
false;
        pictureBox_Visible.Visible =
true;
        textBox_Password.Password =
true;
        textBox_Password.Refresh();
    }

    private void
btn_Return_Click(object sender,
EventArgs e)
    {
        textBox_Password.Clear();
        textBox_Username.Clear();
        btnRegister.Visible = true;
        btnLogIn.Visible = true;
        btnSignUp.Visible = false;
    }
```

```
    }
}
```

UI_MainPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UI_MainPage : MaterialForm
    {
        private ControllerForms controllerForms = new ControllerForms();
        private Controller controller = new Controller();

        public UI_MainPage()
        {
            InitializeComponent();
            //Setting design
            var materialSkinManager = MaterialSkinManager.Instance;

            materialSkinManager.AddFormToManage(this);
            materialSkinManager.Theme = MaterialSkinManager.Themes.DARK;

            materialSkinManager.ColorScheme = new ColorScheme(Primary.Amber800, Primary.Grey800, Primary.Grey400, Accent.Amber700, TextShade.WHITE);
        }

        private void UI_MainPage_Load(object sender, EventArgs e)
        {

            GenerateDynamicUserControl(GetAllStories());
        }

        private DataTable GetAllStories()
        {
            //Selecting all strories from database
            string query = $"SELECT id_story, cover_story, title_story, date_story " +
                $"FROM stories WHERE id_user = '{controller.UserID}' ORDER BY date_story DESC";
            DataTable dt = controller.Database.GetTable(query);
            return dt;
        }

        //Creating a collections of story cards
        private void GenerateDynamicUserControl(DataTable stories)
        {

            flowLayoutPanel_Stories.Controls.Clear();

            UC_StoryCard[] storyCards = new UC_StoryCard[stories.Rows.Count];

            foreach (DataRow row in stories.Rows)
            {
```

```
            storyCards[0] = new
UC_StoryCard();
            storyCards[0].ID =
(int)row["id_story"];
            if (row["cover_story"] !=
DBNull.Value)
            {
               storyCards[0].Cover =
controller.ImageProcess.BinaryToImag
e((byte[])row["cover_story"]);
            }
            else { storyCards[0].Cover =
Properties.Resources.storycover; }

            storyCards[0].Title =
row["title_story"].ToString();
            storyCards[0].DateTime =
row["date_story"].ToString();


flowLayoutPanel_Stories.Controls.Add
(storyCards[0]);
            storyCards[0].Click += new
System.EventHandler(this.UserControl
_Click);
         }
      }

      void UserControl_Click(object
sender, EventArgs e) //A card was
picked
      {
         this.Hide();
         UC_StoryCard card =
(UC_StoryCard)sender;
         controller.StoryID = card.ID;

controllerForms.StoryForm(false);

GenerateDynamicUserControl(GetAllS
tories());
      }

      private void btnAdd_Click(object
sender, EventArgs e)
      {
         this.Hide();

controllerForms.StoryForm(true);

GenerateDynamicUserControl(GetAllS
tories());
      }

      private void
UI_MainPage_FormClosed(object
sender, FormClosedEventArgs e)
      {
         this.Hide();
         controllerForms.LoginForm();
      }

      private void
btnSearch_Click(object sender,
EventArgs e)
      {
         string query = $"SELECT
id_story, cover_story, title_story,
date_story " +
            $"FROM stories WHERE
id_user = '{controller.UserID}' " +
            $"AND title_story LIKE
'%{TextBox_Search.Text}%' OR
summary LIKE
'%{TextBox_Search.Text}%' ORDER
BY date_story DESC";
         DataTable result =
controller.Database.GetTable(query);

GenerateDynamicUserControl(result);
      }
   }
}
```

UI_Story.cs:

```
using System;

using System.IO;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class UI_Story : MaterialForm
   {
      private ControllerForms controllerForms = new ControllerForms();
      private Controller controller = new Controller();
      private bool isNew;
      public bool deleted;

      public UI_Story(bool isNew)
      {
         InitializeComponent();
         //Setting design
         var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);
         materialSkinManager.Theme = MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme = new ColorScheme(Primary.Amber800, Primary.Grey800, Primary.Grey400, Accent.Amber700, TextShade.WHITE);

         this.isNew = isNew;

      }

      private void UI_Story_Load(object sender, EventArgs e)
      {
         pictureBox_Cover.SizeMode = PictureBoxSizeMode.CenterImage;
         pictureBox_Cover.Image = Properties.Resources.storycover;
         pictureBox_Cover.Name = "default";

         if (isNew) // User adds a story
         {
            //Adding the story to the table
```

```
            CreateStory();
            CreateDraft();
            Label_DateTime.Text =
DateTime.Now.ToString("dd/MM/yyy
y HH:mm");
        }
        else // User opened an existing
story
        {
            LoadFields();
        }
        LoadDrafts();
    }

    private void CreateDraft()
    {
        SqlCommand command =
controller.MakeCommand($"INSERT
INTO drafts (number_draft, id_story)
values (1, '{controller.StoryID}')");

controller.Database.OpenConnection();

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with creating a draft!", "Fail!");
        }
controller.Database.CloseConnection();
    }

    private void LoadDrafts()
    {
        controller.SetDrafts();
    }

    private void LoadFields()
    {
        string query = $"SELECT
cover_story, title_story, date_story,
summary" +
        $" FROM stories WHERE
id_story = '{controller.StoryID}'";

        DataTable dt =
controller.Database.GetTable(query);

        if (dt.Rows.Count == 1)
        {
            TextBox_Title.Text =
dt.Rows[0]["title_story"].ToString();
            this.Text =
$"{dt.Rows[0]["title_story"].ToString()
}";

            Label_DateTime.Text =
dt.Rows[0]["date_story"].ToString();
```

```
            if (dt.Rows[0]["cover_story"] != DBNull.Value)
            {
                pictureBox_Cover.Image = controller.ImageProcess.BinaryToImage((byte[])dt.Rows[0]["cover_story"]);
            }


            if (dt.Rows[0]["summary"] != DBNull.Value)
            {

                MultiLineTextBox_Desciption.Text = dt.Rows[0]["summary"].ToString();
            }


        }
    }

    private void pictureBox_Cover_DoubleClick(object sender, EventArgs e)
    {
        OpenFileDialog Ofd = new OpenFileDialog();
        if (Ofd.ShowDialog() == DialogResult.OK)
        {
            //textBoxFile.Text = Ofd.FileName;
            try
            {
                var imageCover = Image.FromFile(Ofd.FileName);

                pictureBox_Cover.SizeMode = PictureBoxSizeMode.CenterImage;
                pictureBox_Cover.Image = imageCover;
                pictureBox_Cover.Name = null;
            }
            catch (Exception ex)
            {

                MessageBox.Show("Wrong file!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }


        }
    }

    #region Operations with story (Create, Update, Delete)

    private void CreateStory()
    {
        SqlCommand command = controller.MakeCommand($"INSERT INTO stories (date_story, id_user) values ('{DateTime.Now}', '{controller.UserID}')");
```

```
controller.Database.OpenConnection();

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

        //Setting created story's ID as
current one
        string query = $"SELECT
SCOPE_IDENTITY() AS
[SCOPE_IDENTITY]";
        DataTable dt =
controller.Database.GetTable(query);

        controller.StoryID =
Convert.ToInt32(dt.Rows[0]["SCOPE_
IDENTITY"]);

controller.Database.CloseConnection();

    }

    private void
UI_Story_FormClosing(object sender,
FormClosingEventArgs e)
    {
        //If story was deleted
        if (deleted) {

controllerForms.MainShow(); }
        else
        {
            //If story was just created
            if (isNew)
            {
                DialogResult saveRes =
MaterialMessageBox.Show($"Do you
want to save the story you created?",
                    "Saving",
MessageBoxButtons.YesNo);
                if (saveRes ==
DialogResult.No)
                {
                    DeleteStory();

controllerForms.MainShow();
                    return;
                }
            }

            //Checking if the title filled
out for saving/updating
            if
(TextBox_Title.TextLength == 0)
            {
                DialogResult titleRes =
MaterialMessageBox.Show("Fill out
the title or you won't save the story!",
```

```
                "Warning!",
MessageBoxButtons.OKCancel);
            if(titleRes ==
DialogResult.Cancel)
            {
                DialogResult deleteRes
= MaterialMessageBox.Show("Maybe
you want to delete the story instead?",
"Deleting",
MessageBoxButtons.YesNo);
                if(deleteRes ==
DialogResult.Yes)
                {
                    DeleteStory();

controllerForms.MainShow();
                    return;
                }
            }
            e.Cancel = true;

        }
        else
        {
            UpdateStory();

controllerForms.MainShow();
        }
    }
}
```

```
    private void UpdateStory()
    {

controller.Database.OpenConnection();

        SqlCommand command =
controller.MakeCommand($"UPDATE
stories SET cover_story = @Cover,
title_story = @Title, " +
            $"summary = @Description,
date_story = @DateTime WHERE
id_story = '{controller.StoryID}'");

        SetFields(command);

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

controller.Database.CloseConnection();
    }

    private void
SetFields(SqlCommand command)
    {
        if (pictureBox_Cover.Name ==
"default")
```

```
        {

command.Parameters.AddWithValue("
@Cover",
System.Data.SqlTypes.SqlBinary.Null)
;
        }
        else {
command.Parameters.AddWithValue("
@Cover",
controller.ImageProcess.ImageToBinar
y(pictureBox_Cover.Image)); }



command.Parameters.AddWithValue("
@Title", TextBox_Title.Text);

command.Parameters.AddWithValue("
@DateTime", DateTime.Now);


        if
(MultiLineTextBox_Desciption.Text
== "Description")
        {

command.Parameters.AddWithValue("
@Description", DBNull.Value);
        }
        else {
command.Parameters.AddWithValue("
@Description",
MultiLineTextBox_Desciption.Text); }
    }
```

```
    private void
button_Delete_Click(object sender,
EventArgs e)
    {


        DialogResult result =
MaterialMessageBox.Show($"The
story {this.Text} will be deleted
forever. Continue?", "Deleting the
story", MessageBoxButtons.OK);
        if (result == DialogResult.OK)
        {
            DeleteStory();
            this.Close();
        }
    }


    private void DeleteStory()
    {
        SqlCommand command;
        string query;



controller.Database.OpenConnection();


        //Deleting characters
        query = $"SELECT * FROM
characters WHERE id_story =
'{controller.StoryID}'";
        DataTable dt =
controller.Database.GetTable(query);
        if (dt.Rows.Count > 0)
```

```
        {
            DeleteExtentions("char");
            command = controller.MakeCommand($"DELETE FROM characters WHERE id_story = '{controller.StoryID}'");

command.ExecuteNonQuery();
        }

        //Deleting worlds
        query = $"SELECT * FROM worlds WHERE id_story = '{controller.StoryID}'";
        dt = controller.Database.GetTable(query);
        if (dt.Rows.Count > 0)
        {
            DeleteExtentions("world");
            command = controller.MakeCommand($"DELETE FROM worlds WHERE id_story = '{controller.StoryID}'");

command.ExecuteNonQuery();
        }

        //Deleting notes
        command = controller.MakeCommand($"DELETE FROM notes_gallery WHERE id_story = '{controller.StoryID}'");
        command.ExecuteNonQuery();

        //Deleting links
        command = controller.MakeCommand($"DELETE FROM links_gallery WHERE id_story = '{controller.StoryID}'");
        command.ExecuteNonQuery();

        //Deleting pictures
        command = controller.MakeCommand($"DELETE FROM pictures_gallery WHERE id_story = '{controller.StoryID}'");
        command.ExecuteNonQuery();

        // TODO: Add deleting everyting connected to writing
        command = controller.MakeCommand($"DELETE FROM drafts WHERE id_story = '{controller.StoryID}'");
        command.ExecuteNonQuery();

        //Deleting story
        command = controller.MakeCommand($"DELETE FROM stories WHERE id_story = '{controller.StoryID}'");
        command.ExecuteNonQuery();

controller.Database.CloseConnection();
```

```
            deleted = true;
        }

        //Deleting everything associeted with characters/worlds of the story
        private void DeleteExtentions(string type)
        {
            if (type == "char")
            {
                //Deleting all associeted fields in sections
                SqlCommand command = controller.MakeCommand($"DELETE FROM char_attribute WHERE id_field IN " +
                    $"(SELECT id_field FROM char_attribute WHERE id_csection IN " +
                    $"(SELECT id_csection FROM character_section WHERE character_id IN" +
                    $"(SELECT id_char FROM characters WHERE id_story = '{controller.StoryID}')))");

                command.ExecuteNonQuery();

                //Deleting all associeted sections
                command = controller.MakeCommand($"DELETE FROM character_section WHERE character_id IN " +
                    $"(SELECT id_char FROM characters WHERE id_story = '{controller.StoryID}')");

                command.ExecuteNonQuery();
            }
            else
            {
                //TODO: add deleting for world
            }
        }

        #endregion

        private void MultiLineTextBox_Desciption_Click(object sender, EventArgs e)
        {
            if (TextBox_Title.Text == "Description")
            {
                MultiLineTextBox_Desciption.Clear();
            }
        }

        private void uc_AllCharactersPage_CharacterPicked(object sender, EventArgs e)
        {
            uc_CharacterPage.Reset();
```

```
uc_CharacterPage.LoadControl(false);

uc_CharacterPage.BringToFront();

this.DrawerShowIconsWhenHidden =
false;
            this.Refresh();
        }

        private void
uc_CharacterPage_LeavePage(object
sender, EventArgs e)
        {
            //this.DrawerTabControl =
TabControl_Story;

this.DrawerShowIconsWhenHidden =
true;

uc_AllCharactersPage.BringToFront();
            uc_AllCharactersPage.Reset();
            this.Refresh();
        }

        private void
uc_AllCharactersPage_UserAddsCharac
cter(object sender, EventArgs e)
        {
            uc_CharacterPage.Reset();

uc_CharacterPage.LoadControl(true);

uc_CharacterPage.BringToFront();

this.DrawerShowIconsWhenHidden =
false;
            this.Refresh();
        }

        private void
uc_AllWorldsPage_WorldPicked(objec
t sender, EventArgs e)
        {
        }

        private void
uc_AllWorldsPage_UserAddsWorld(o
bject sender, EventArgs e)
        {

        }

        private void
uc_WorldPage_LeavePage(object
sender, EventArgs e)
        {
        }
    }
}
```

UC_AllCharactersPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class
UC_AllCharactersPage : UserControl
   {
      private Controller controller =
new Controller();

      public event EventHandler
CharacterPicked;

      public event EventHandler
UserAddsCharacter;

      public UC_AllCharactersPage()
      {
         InitializeComponent();

GenerateDynamicUserControl();
      }

      public void Reset()
      {
        GenerateDynamicUserControl();
      }

      //Creating a collection of
characters's cards
      private void
GenerateDynamicUserControl()
      {

flowLayoutPanel_Characters.Controls.
Clear();

         string query = $"SELECT
id_char, icon_char, name_char " +
            $"FROM characters WHERE
id_story = '{controller.StoryID}'
ORDER BY date_char DESC";
         DataTable dt =
controller.Database.GetTable(query);

         UC_ObjectCard[] charCard =
new UC_ObjectCard[dt.Rows.Count];

         foreach (DataRow row in
dt.Rows)
         {
            charCard[0] = new
UC_ObjectCard();
            charCard[0].ID =
(int)row["id_char"];
            if (row["icon_char"] !=
DBNull.Value)
            {
               charCard[0].Icon =
controller.ImageProcess.BinaryToImag
e((byte[])row["icon_char"]);
            }
            else { charCard[0].Icon =
Properties.Resources.charactericon; }

            charCard[0].Name =
row["name_char"].ToString();


flowLayoutPanel_Characters.Controls.
Add(charCard[0]);
            charCard[0].Click += new
System.EventHandler(this.UserControl
_Click);

         }
      }

      protected virtual void
UserControl_Click(object sender,
EventArgs e)
      {
```

```
            UC_ObjectCard card =
(UC_ObjectCard)sender;
            controller.CharacterID =
card.ID;
            CharacterPicked?.Invoke(this,
e);
        }

        private void
button_Add_Click(object sender,
EventArgs e)
        {

UserAddsCharacter?.Invoke(this, e);
        }
    }
}
```

UC_CharacterPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class
UC_CharacterPage : UserControl
    {
        private bool isNew;
        public event EventHandler
LeavePage;
        private Controller controller =
new Controller();

        public UC_CharacterPage()
        {
            InitializeComponent();
        }

        public void Reset()
        {
            pictureBox_Icon.Image = null;
            textBox_Name.Text = null;
            multiTextBox_Description.Text
= "Description";

flowLayoutPanel_Attributes.Controls.
Clear();

flowLayoutPanel_Gallery.Controls.Cle
ar();
        }

        public void LoadControl(bool
isNew)
        {
            this.isNew = isNew;

            if (isNew)
            {

controller.Database.OpenConnection();
                CreateCharacter();
                AddBasicSections();

controller.Database.CloseConnection();
            }

            LoadInformation();
        }

        private void
UC_CharacterPage_Load(object
sender, EventArgs e)
        {
        }

        private void CreateCharacter()
        {
```

```
        SqlCommand command =
controller.MakeCommand($"INSERT
INTO characters (date_char, id_story)
values ('{DateTime.Now}',
'{controller.StoryID}')");

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

        //Setting created character's ID
as current one
        string query = $"SELECT
SCOPE_IDENTITY() AS
[SCOPE_IDENTITY]";
        DataTable dt =
controller.Database.GetTable(query);

        controller.CharacterID =
Convert.ToInt32(dt.Rows[0]["SCOPE_
IDENTITY"]);
    }

    private void UpdateCharacter()
    {

controller.Database.OpenConnection();

        SqlCommand command =
controller.MakeCommand($"UPDATE
characters SET icon_char = @Icon,
name_char = @Name, " +
            $"summary_char =
@Description, date_char =
@DateTime WHERE id_char =
'{controller.CharacterID}'");

        CheckFields(command);

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

        foreach(UC_SectionCard
section in
flowLayoutPanel_Attributes.Controls)
        {
            section.UpdateSection();
        }


controller.Database.CloseConnection();
    }

    private void
CheckFields(SqlCommand command)
    {
        if (pictureBox_Icon.Name ==
"default")
        {

command.Parameters.AddWithValue("
@Icon",
System.Data.SqlTypes.SqlBinary.Null)
;
        }
        else {
command.Parameters.AddWithValue("
@Icon",
controller.ImageProcess.ImageToBinar
y(pictureBox_Icon.Image)); }


command.Parameters.AddWithValue("
@Name", textBox_Name.Text);

command.Parameters.AddWithValue("
@DateTime", DateTime.Now);
```

```
        if
(multiTextBox_Description.Text ==
"Description")
        {

command.Parameters.AddWithValue("
@Description", DBNull.Value);
        }
        else {
command.Parameters.AddWithValue("
@Description",
multiTextBox_Description.Text); }
    }

    private void AddBasicSections()
    {

        SqlCommand command =
controller.MakeCommand($"INSERT
INTO character_section(character_id,
section_id)" +
            $"SELECT
'{controller.CharacterID}', id_section "
+
            $"FROM sections " +
            $"WHERE id_section
BETWEEN 1 AND 6 ");

        if
(command.ExecuteNonQuery() == 0)
        {

MaterialMessageBox.Show("Error
with adding basic sections!", "Fail!");
        }

        //Adding specific basic fields
based on section
        command =
controller.MakeCommand($"INSERT
INTO char_attribute(id_csection,
id_field) SELECT S.id_csection,
id_field " +
            $"FROM character_section
S, fields WHERE S.character_id =
'{controller.CharacterID}' and id_field
= " +
            $"CASE " +
            $"WHEN S.section_id = 1
and id_field BETWEEN 1 AND 7
THEN id_field " +
            $"WHEN S.section_id = 2
and id_field BETWEEN 8 AND 13
THEN id_field " +
            $"WHEN S.section_id = 3
and id_field BETWEEN 14 AND 19
THEN id_field " +
            $"WHEN S.section_id = 4
and id_field BETWEEN 20 AND 26
THEN id_field " +
            $"WHEN S.section_id = 5
and id_field BETWEEN 27 AND 30
THEN id_field " +
            $"END");

        if
(command.ExecuteNonQuery() == 0)
        {

MaterialMessageBox.Show("Error
with adding basic fields!", "Fail!");
        }
    }

    private void LoadInformation()
    {
        //Loading general information
about a character
        string query = $"SELECT
icon_char, name_char, summary_char
FROM characters WHERE id_char =
'{controller.CharacterID}'";
        DataTable dt =
controller.Database.GetTable(query);

        string text =
dt.Rows[0]["name_char"].ToString();
```

```
            textBox_Name.Text =
dt.Rows[0]["name_char"].ToString();

            if (dt.Rows[0]["icon_char"] !=
DBNull.Value)
            {
                pictureBox_Icon.Image =
controller.ImageProcess.BinaryToImag
e((byte[])dt.Rows[0]["icon_char"]);
            }
            else {
                pictureBox_Icon.Image =
Properties.Resources.charactericon;
                pictureBox_Icon.Name =
"default";
            }

            if
(dt.Rows[0]["summary_char"] !=
DBNull.Value)
            {

multiTextBox_Description.Text =
dt.Rows[0]["summary_char"].ToString
();
            }

            LoadSectionsFields();

        }

        private void LoadSectionsFields()
        {

flowLayoutPanel_Attributes.Controls.
Clear();

            controller.SetSectionsList(true);



            UC_SectionCard[] sections =
new
UC_SectionCard[controller.CSections
List.Count];

            foreach(Section section in
controller.CSectionsList)
            {
                sections[0] = new
UC_SectionCard(section);

flowLayoutPanel_Attributes.Controls.
Add(sections[0]);
            }

        }

        private void
pictureBox_Back_Click(object sender,
EventArgs e)
        {
            if (isNew)
            {
                DialogResult result =
MaterialMessageBox.Show($"Do you
want to save the character you
created?",
                    "Saving",
MessageBoxButtons.YesNo);
                if (result ==
DialogResult.No)
                {
                    DeleteCharacter();
                    LeavePage?.Invoke(this,
e);
                    return;
                }
            }
            //Checking if the title filled out
for saving/updating
            if (textBox_Name.TextLength
== 0)
            {
MaterialMessageBox.Show("Fill out
the Name!", "Error!",
MessageBoxButtons.OK);

            }
```

```
            else
            {
                UpdateCharacter();
                LeavePage?.Invoke(this, e);
            }
        }

        private void DeleteCharacter()
        {

        }

        private void
pictureBox_Icon_Click(object sender,
EventArgs e)
        {
            OpenFileDialog Ofd = new
OpenFileDialog();
            if (Ofd.ShowDialog() ==
DialogResult.OK)
            {
                try
                {
                    var imageIcon =
Image.FromFile(Ofd.FileName);

pictureBox_Icon.SizeMode =
PictureBoxSizeMode.CenterImage;
                    pictureBox_Icon.Image =
imageIcon;
                    pictureBox_Icon.Name =
null;
                }
                catch (Exception ex)
                {

MessageBox.Show("Wrong file!",
"Error", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
                }

            }
        }

        private void
button_Settings_Click(object sender,
EventArgs e)
        {
            UI_Settings i_Settings = new
UI_Settings();
            i_Settings.Show();
        }

        private void
button_AddNote_Click(object sender,
EventArgs e)
        {
            UI_AddElement i_AddElement
= new UI_AddElement("note");
            i_AddElement.Show();
        }

        private void
buttton_DeletePic_Click(object sender,
EventArgs e)
        {

        }

        private void
button_EditNote_Click(object sender,
EventArgs e)
        {
            UI_EditElement i_EditElement
= new UI_EditElement("note");
            i_EditElement.Show();
        }

        private void
button_AddPic_Click(object sender,
EventArgs e)
        {

        }
    }
}
```

UC_AllWorldsPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class
UC_AllWorldsPage : UserControl
   {
      private Controller controller =
new Controller();

      public event EventHandler
WorldPicked;

      public event EventHandler
UserAddsWorld;

      public UC_AllWorldsPage()
      {
         InitializeComponent();
      }

      public void Reset()
      {

GenerateDynamicUserControl();
      }

      private void
GenerateDynamicUserControl()
      {

flowLayoutPanel_Worlds.Controls.Cle
ar();

         string query = $"SELECT
id_world, icon_world, name_world " +
            $"FROM world WHERE
id_world = '{controller.StoryID}'
ORDER BY date_world DESC";
         DataTable dt =
controller.Database.GetTable(query);

         UC_ObjectCard[] charCard =
new UC_ObjectCard[dt.Rows.Count];

         foreach (DataRow row in
dt.Rows)
         {
            charCard[0] = new
UC_ObjectCard();
            charCard[0].ID =
(int)row["id_world"];
            if (row["icon_world"] !=
DBNull.Value)
            {
               charCard[0].Icon =
controller.ImageProcess.BinaryToImag
e((byte[])row["icon_world"]);
            }
            else { charCard[0].Icon =
Properties.Resources.charactericon; }

            charCard[0].Name =
row["name_world"].ToString();


flowLayoutPanel_Worlds.Controls.Ad
d(charCard[0]);
            charCard[0].Click += new
System.EventHandler(this.UserControl
_Click);

         }
      }

      protected virtual void
UserControl_Click(object sender,
EventArgs e)
      {
```

```
            UC_ObjectCard card =
(UC_ObjectCard)sender;
            controller.WorldID = card.ID;
            WorldPicked?.Invoke(this, e);
        }

        private void
button_Add_Click(object sender,
EventArgs e)
        {
            UserAddsWorld?.Invoke(this,
e);
        }
    }
}
```

UC_WorldPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UC_WorldPage :
UserControl
    {
        private bool isNew;
        public event EventHandler
LeavePage;
        private Controller controller =
new Controller();

        public UC_WorldPage()
        {
            InitializeComponent();
        }

        public void Reset()
        {
            pictureBox_Icon.Image = null;
            textBox_Name.Text = null;
            multiTextBox_Description.Text
= "Description";

flowLayoutPanel_Attributes.Controls.
Clear();

flowLayoutPanel_Gallery.Controls.Cle
ar();
        }

        public void LoadControl(bool
isNew)
        {
            this.isNew = isNew;

            if (isNew)
            {

controller.Database.OpenConnection();
                CreateWorld();
                AddBasicSections();

controller.Database.CloseConnection();
            }

            LoadInformation();
        }

        private void CreateWorld()
        {
            SqlCommand command =
controller.MakeCommand($"INSERT
INTO worlds (date_world, id_world) "
+
                $"values ('{DateTime.Now}',
'{controller.StoryID}')");
```

```
        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

        //Setting created world's ID as
current one
        string query = $"SELECT
SCOPE_IDENTITY() AS
[SCOPE_IDENTITY]";
        DataTable dt =
controller.Database.GetTable(query);

        controller.WorldID =
Convert.ToInt32(dt.Rows[0]["SCOPE_
IDENTITY"]);
    }

    private void UpdateWorld()
    {

controller.Database.OpenConnection();

        SqlCommand command =
controller.MakeCommand($"UPDATE
worlds SET icon_world = @Icon,
name_world = @Name, " +
            $"summary_world =
@Description, date_world =
@DateTime WHERE id_world =
'{controller.WorldID}'");

        CheckFields(command);

        if
(command.ExecuteNonQuery() != 1)
        {

MaterialMessageBox.Show("Error
with saving data!", "Fail!");
        }

        foreach (UC_SectionCard
section in
flowLayoutPanel_Attributes.Controls)
        {
            section.UpdateSection();
        }


controller.Database.CloseConnection();
    }

    private void
CheckFields(SqlCommand command)
    {
        if (pictureBox_Icon.Name ==
"default")
        {

command.Parameters.AddWithValue("
@Icon",
System.Data.SqlTypes.SqlBinary.Null)
;
        }
        else {
command.Parameters.AddWithValue("
@Icon",
controller.ImageProcess.ImageToBinar
y(pictureBox_Icon.Image)); }


command.Parameters.AddWithValue("
@Name", textBox_Name.Text);

command.Parameters.AddWithValue("
@DateTime", DateTime.Now);

        if
(multiTextBox_Description.Text ==
"Description")
        {

command.Parameters.AddWithValue("
@Description", DBNull.Value);
```

```
            }
            else {
command.Parameters.AddWithValue("
@Description",
multiTextBox_Description.Text); }
        }

        private void AddBasicSections()
        {

            //SqlCommand command =
controller.MakeCommand($"INSERT
INTO character_section(character_id,
section_id)" +
            //    $"SELECT
'{controller.CharacterID}', id_section "
+
            //    $"FROM sections " +
            //    $"WHERE id_section
BETWEEN 1 AND 6 ");

            //if
(command.ExecuteNonQuery() == 0)
            //{
            //
MaterialMessageBox.Show("Error
with adding basic sections!", "Fail!");
            //}

            ////Adding specific basic fields
based on section
            //command =
controller.MakeCommand($"INSERT
INTO char_attribute(id_csection,
id_field) SELECT S.id_csection,
id_field " +
            //    $"FROM character_section
S, fields WHERE S.character_id =
'{controller.CharacterID}' and id_field
= " +
            //    $"CASE " +
            //    $"WHEN S.section_id = 1
and id_field BETWEEN 1 AND 7
THEN id_field " +
            //    $"WHEN S.section_id = 2
and id_field BETWEEN 8 AND 13
THEN id_field " +
            //    $"WHEN S.section_id = 3
and id_field BETWEEN 14 AND 19
THEN id_field " +
            //    $"WHEN S.section_id = 4
and id_field BETWEEN 20 AND 26
THEN id_field " +
            //    $"WHEN S.section_id = 5
and id_field BETWEEN 27 AND 30
THEN id_field " +
            //    $"END");

            //if
(command.ExecuteNonQuery() == 0)
            //{
            //
MaterialMessageBox.Show("Error
with adding basic fields!", "Fail!");
            //}
        }

        private void LoadInformation()
        {
            //Loading general information
about world
            string query = $"SELECT
icon_world, name_world,
summary_world FROM worlds
WHERE id_world =
'{controller.WorldID}'";
            DataTable dt =
controller.Database.GetTable(query);

            textBox_Name.Text =
dt.Rows[0]["name_world"].ToString();

            if (dt.Rows[0]["icon_world"] !=
DBNull.Value)
            {
                pictureBox_Icon.Image =
controller.ImageProcess.BinaryToImag
e((byte[])dt.Rows[0]["icon_world"]);
```

```
        }
        else
        {
            pictureBox_Icon.Image =
Properties.Resources.charactericon;
            pictureBox_Icon.Name =
"default";
        }

        if
(dt.Rows[0]["summary_world"] !=
DBNull.Value)
        {

multiTextBox_Description.Text =
dt.Rows[0]["summary_world"].ToStrin
g();
        }

        LoadSectionsFields();

    }

    private void LoadSectionsFields()
    {

flowLayoutPanel_Attributes.Controls.
Clear();


controller.SetSectionsList(false);


        UC_SectionCard[] sections =
new
UC_SectionCard[controller.CSections
List.Count];

        foreach (Section section in
controller.CSectionsList)
        {
            sections[0] = new
UC_SectionCard(section);

flowLayoutPanel_Attributes.Controls.
Add(sections[0]);
        }

    }

    private void
pictureBox_Back_Click(object sender,
EventArgs e)
    {
        if (isNew)
        {
            DialogResult result =
MaterialMessageBox.Show($"Do you
want to save the world you created?",
                "Saving",
MessageBoxButtons.YesNo);
            if (result ==
DialogResult.No)
            {
                DeleteWorld();
                LeavePage?.Invoke(this,
e);
                return;
            }
        }
        //Checking if the title filled out
for saving/updating
        if (textBox_Name.TextLength
== 0)
        {

MaterialMessageBox.Show("Fill out
the Name!", "Error!",
MessageBoxButtons.OK);

        }
        else
        {
            UpdateWorld();
            LeavePage?.Invoke(this, e);
        }
    }
```

```
        private void DeleteWorld()
        {

        }

        private void
pictureBox_Icon_Click(object sender,
EventArgs e)
        {
            OpenFileDialog Ofd = new
OpenFileDialog();
            if (Ofd.ShowDialog() ==
DialogResult.OK)
            {
                try
                {
                    var imageIcon =
Image.FromFile(Ofd.FileName);

pictureBox_Icon.SizeMode =
PictureBoxSizeMode.CenterImage;
                    pictureBox_Icon.Image =
imageIcon;
                    pictureBox_Icon.Name =
null;
                }
                catch (Exception ex)
                {

MessageBox.Show("Wrong file!",
"Error", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
                }

            }
        }

        private void
button_Settings_Click(object sender,
EventArgs e)
        {
            UI_Settings i_Settings = new
UI_Settings();
            i_Settings.Show();
        }


        private void
button_AddNote_Click(object sender,
EventArgs e)
        {
            UI_AddElement i_AddElement
= new UI_AddElement("note");
            i_AddElement.Show();
        }

        private void
buttton_DeletePic_Click(object sender,
EventArgs e)
        {

        }

        private void
button_EditNote_Click(object sender,
EventArgs e)
        {
            UI_EditElement i_EditElement
= new UI_EditElement("note");
            i_EditElement.Show();
        }

        private void
button_AddPic_Click(object sender,
EventArgs e)
        {

        }
    }
}
```

UC_PlotPage.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class UC_PlotPage :
UserControl
   {
      private Controller controller =
new Controller();

      public UC_PlotPage()
      {
         InitializeComponent();

         LoadInformation();
      }

      public void LoadInformation()
      {
         //Getting information
      }

      public void SaveChapter()
      {
         //Adding chapter to database
      }

      private void
comboBox_Chapters_SelectedIndexCh
anged(object sender, EventArgs e)
      {
         //Changing to appropriate
chapter
      }

      private void
comboBox_Drafts_SelectedIndexChan
ged(object sender, EventArgs e)
      {
         //Chnaging to needed draft
      }

      private void
button_Settings_Click(object sender,
EventArgs e)
      {
         //Opening settings
      }

      private void
button_AddChapter_Click(object
sender, EventArgs e)
      {
         //Adding chapter
      }

      private void
btn_AddTag_Click(object sender,
EventArgs e)
      {
         //Adding tag
      }

      private void
btn_AddScene_Click(object sender,
EventArgs e)
      {
         //Adding scene
      }

      private void
comboBox_Timelines_SelectedIndexC
hanged(object sender, EventArgs e)
      {
         //Changing to needed timeline
      }

      private void
btn_AddTimeline_Click(object sender,
EventArgs e)
      {
         //Addding timeline
      }
```

```
        private void
btn_AddEvent_Click(object sender,
EventArgs e)
        {
            //Adding event
        }
    }
}
```

UI_Inspiration.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
    public partial class UC_Inspiration :
UserControl
    {
        public UC_Inspiration()
        {
            InitializeComponent();
        }


        private void
comboBox_Type_SelectedIndexChang
ed(object sender, EventArgs e)
        {
            //Changing contents to links or
pictures
        }

        private void
btn_Add_Click(object sender,
EventArgs e)
        {
        }
    }
}
```

UI_Settings.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UI_Settings :
MaterialForm
    {
        private Controller controller =
new Controller();

        public UI_Settings()
        {
            InitializeComponent();

            var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);
```

```
            LoadSections();
        }

        private void LoadSections()
        {

flowLayoutPanel_Attributes.Controls.
Clear();


            UC_SectionEditCard[] sections
= new
UC_SectionEditCard[controller.CSecti
onsList.Count];

            foreach (Section section in
controller.CSectionsList)
            {
                sections[0] = new
UC_SectionEditCard(section);

flowLayoutPanel_Attributes.Controls.
Add(sections[0]);
            }
        }

        private void Update()
        {

controller.Database.OpenConnection();

            foreach (UC_SectionEditCard
section in
flowLayoutPanel_Attributes.Controls)
            {
                section.UpdateSection();
            }


controller.Database.CloseConnection();
        }

        private void
button_AddSection_Click(object
sender, EventArgs e)
        {
            UI_AddElement i_AddElement
= new UI_AddElement("section");
            i_AddElement.Show();
        }

        private void
UI_Settings_Load(object sender,
EventArgs e)
        {

        }

        private void
button_AddField_Click(object sender,
EventArgs e)
        {
            UI_AddField i_AddField = new
UI_AddField();
            i_AddField.Show();
        }

        private void
UI_Settings_FormClosing(object
sender, FormClosingEventArgs e)
        {

        }
    }
}
```

UC_SettingsChapter.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class
UI_SettingsChapters : MaterialForm
   {
      private Controller controller =
new Controller();

      public UI_SettingsChapters()
      {
         InitializeComponent();
         var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
         materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);

      }
      private void LoadChapters()
      {

      }

      private void Update()
      {
      }

      private void
button_Delete_Click(object sender,
EventArgs e)
      {

      }

      private void
UI_SettingsChapters_FormClosing(obj
ect sender, FormClosingEventArgs e)
      {

      }
   }
}
```

UC_SectionCard.cs:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class UC_SectionCard
: UserControl
   {
      private Controller controller =
new Controller();
      bool showFields = false;
      private Section sectionInfo = new
Section();

      public UC_SectionCard(Section
section)
      {
         InitializeComponent();

         flowLayoutPanel_Fields.Height
= 0;
```

```
            this.BackColor =
System.Drawing.Color.FromArgb(56,
52, 52);

            sectionInfo = section;
            button_Section.Text =
section.Name;
            SetField();
        }

        public void SetField()
        {

flowLayoutPanel_Fields.Controls.Clear
();
            foreach(Field field in
sectionInfo.fieldsList)
            {
                MaterialTextBox
textbox_Field = new
MaterialTextBox();
                textbox_Field.Name =
field.ID.ToString();
                textbox_Field.Hint =
field.Name;
                textbox_Field.Width = 300;
                if (field.Text != null)
                {
                    textbox_Field.Text =
field.Text;
                }

flowLayoutPanel_Fields.Controls.Add(
textbox_Field);
            }
        }

        public void UpdateSection()
        {
            foreach(Control field in
flowLayoutPanel_Fields.Controls)
            {
                if (field.Text.Length != 0)
                {
                    SqlCommand command =
controller.MakeCommand($"UPDATE
char_attribute SET text_attribute =
'{field.Text}' " +
                        $"WHERE id_cattribute
= '{field.Name}'");

                    if
(command.ExecuteNonQuery() != 1)
                    {

MaterialMessageBox.Show("Error
with saving fields!", "Fail!");
                    }
                }
            }
        }

        private void
button_Section_Click(object sender,
EventArgs e)
        {
            if(showFields)
            {

flowLayoutPanel_Fields.Height = 0;
                button_Section.Icon =
Properties.Resources.expandarrow;
                showFields = false;
            }
            else
            {

flowLayoutPanel_Fields.Height =
sectionInfo.fieldsList.Count * 55;
                button_Section.Icon =
Properties.Resources.collapsearrow;
                showFields = true;
            }
            this.Refresh();
        }
    }
}
```

UC_SectionEditCard.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class
UC_SectionEditCard : UserControl
   {
      private Controller controller =
new Controller();
      private Section sectionInfo = new
Section();

      public
UC_SectionEditCard(Section section)
      {
         InitializeComponent();

         sectionInfo = section;
         textBox_Name.Text =
section.Name;
         SetField();
      }

      public void SetField()
      {

flowLayoutPanel_Fields.Controls.Clear
();
         foreach (Field field in
sectionInfo.fieldsList)
         {
            MaterialTextBox
textbox_Field = new
MaterialTextBox();
            textbox_Field.Name =
field.ID.ToString();
            textbox_Field.Text =
field.Name;
            textbox_Field.Width = 300;

flowLayoutPanel_Fields.Controls.Add(
textbox_Field);
         }
      }

      public void UpdateSection()
      {
        //Updating section
      }
   }
}
```

UC_StoryCard.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class UC_StoryCard :
UserControl
   {
      public UC_StoryCard()
      {
         InitializeComponent();

         foreach (Control c in
this.Controls)
         {
```

```
            c.Click += (sender, e) => {
this.OnClick(e); };
        }

        label_DateTime.Text = "";
    }

    private Image cover;
    private string title;
    private string datetime;

    public Image Cover
    {
        get { return cover; }
        set { cover = value;
pictureBox_Cover.Image = value;
pictureBox_Cover.SizeMode =
PictureBoxSizeMode.Zoom; }
    }

    public string Title
    {
        get { return title; }
        set { title = value;
label_Title.Text = value; }
    }

    public string DateTime
    {
        get { return datetime; }
        set { datetime = value;
label_DateTime.Text = value; }
    }

    public int ID { get; set; }

  }
}
```

UC_ObjectCard.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
  public partial class UC_ObjectCard :
UserControl
  {
    public UC_ObjectCard()
    {
      InitializeComponent();

      foreach (Control c in
this.Controls)
      {
        c.Click += (sender, e) => {
this.OnClick(e); };
      }
    }

    private Image icon;
    private string name;

    public Image Icon
    {
      get { return icon; }
      set { icon = value;
pictureBox_Icon.Image = value;
pictureBox_Icon.SizeMode =
PictureBoxSizeMode.Zoom; }
    }

    public string Name
    {
      get { return name; }
      set { name = value;
label_Name.Text = value; }
    }

    public int ID { get; set; }
  }
```

```
}
```

UC_ChapterCard.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class UC_ChapterCard
: UserControl
   {
      public UC_ChapterCard()
      {
         InitializeComponent();
      }
   }
}
```

UI_AddChapter.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class UI_AddChapter :
Form
   {
      public UI_AddChapter()
      {
         InitializeComponent();
      }
```

```
   }
}
```

UI_AddElement.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Writer
{
   public partial class UI_AddChapter :
Form
   {
      public UI_AddChapter()
      {
         InitializeComponent();
      }
   }
}
```

UI_AddField.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
   public partial class UI_AddField :
MaterialForm
   {
```

```
        private Controller controller =
new Controller();
        public UI_AddField()
        {
            InitializeComponent();

            var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);
        }

        private void
btn_Add_Click(object sender,
EventArgs e)
        {

        }

        private void
btn_Cancel_Click(object sender,
EventArgs e)
        {

        }
    }
}
```

UI_AddTag.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UI_AddTag :
MaterialForm
    {
        private Controller controller =
new Controller();

        public UI_AddTag()
        {
            InitializeComponent();
            var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);
        }

        private void
btn_Cancel_Click(object sender,
EventArgs e)
        {

        }

        private void
btn_Add_Click(object sender,
EventArgs e)
        {
```

```
        }
    }

}
```

UI_AddElement.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class
UI_DeleteElement : MaterialForm
    {
        private Controller controller =
new Controller();
        public UI_DeleteElement()
        {
            InitializeComponent();
            var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);
        }

        private void
btn_Delete_Click(object sender,
EventArgs e)
        {

        }

        private void
btn_Cancel_Click(object sender,
EventArgs e)
        {

        }
    }
}
```

UI_DeleteField.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UI_DeleteField :
MaterialForm
    {
        private Controller controller =
new Controller();

        public UI_DeleteField()
        {
            InitializeComponent();
            var materialSkinManager =
MaterialSkinManager.Instance;
```

```
materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);
        }

        private void
btn_Delete_Click(object sender,
EventArgs e)
        {

        }

        private void
btn_Cancel_Click(object sender,
EventArgs e)
        {

        }
    }
}
```

UI_EditElement.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MaterialSkin;
using MaterialSkin.Controls;

namespace Writer
{
    public partial class UI_EditElement :
MaterialForm
    {
        private Controller controller =
new Controller();
        public UI_EditElement(string
type)
        {
            InitializeComponent();
            var materialSkinManager =
MaterialSkinManager.Instance;

materialSkinManager.AddFormToMan
age(this);
            materialSkinManager.Theme =
MaterialSkinManager.Themes.DARK;

materialSkinManager.ColorScheme =
new ColorScheme(Primary.Amber800,
Primary.Grey800, Primary.Grey400,
Accent.Amber700,
TextShade.WHITE);

            switch (type)
            {
                case "tag":
                    break;
                case "event":
                    break;
                case "scene":
                    break;
                case "timeline":
                    break;
                case "note":
                    break;
            }
        }

        private void
button_Delete_Click(object sender,
EventArgs e)
        {

        }
```

```
        private void
btn_Save_Click(object sender,
EventArgs e)
        {

        }

        private void
btn_Cancel_Click(object sender,
EventArgs e)
        {

        }
    }
}
```