МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Веб-орієнтований генератор документів на основі шаблонів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студента групи Індн-61МП **Коврига О.О.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи Індн-61МП спеціальності "Інформатика" дистанційної форми навчання Ковриги Олександра Олександровича.

Тема: " **Веб-орієнтований генератор документів на основі шаблонів**"

Затверджена наказом по СумДУ

№_____ от _____2020 р.

Зміст пояснювальної записки: 1) Аналітичний огляд задачі; 2) Проектування інформаційної системи; 3) Практична реалізація інформаційної системи.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Кузіков Б.О.

Завдання приняв до виконання _____ Коврига О.О.

РЕФЕРАТ

Записка: 47 стор., 10 рис., 6 табл., 3 додаток, 14 джерел.

Об’єкт дослідження — процес проектування інформаційної системи генерації документів на основі шаблонів.

Мета роботи — розробити та програмно реалізувати інформаційну систему генерації документів на основі шаблонів з урахуванням особливостей реалізації програмних продуктів.

Методи дослідження — методи проектування інформаційних систем, формування та оптимізації структури бази даних, тестування програмного забезпечення.

Результати — розроблене інформаційне та програмне забезпечення системи генерації документів на основі шаблонів з урахуванням особливостей реалізації програмних продуктів. При цьому розглянуто питання побудови інформаційної моделі системи, моделі даних, розробки інтерфейсу користувача. Запропонована система реалізована за допомогою технологій HTML, JavaScript, Python.

ШАБЛОН, ДОКУМЕНТ, ГЕНЕРАЦІЯ
JAVASCRIPT, HTML, CSS, PYTHON,
FLASK, VUE, BOOTSTRAP, SQLITE
ВЕБ-САЙТ, ВЕБ-СЕРВЕР, БАЗА ДАНИХ

ЗМІСТ

ВСТУП.....5
1 АНАЛІТИЧНИЙ ОГЛЯД ЗАДАЧІ.....8
1.1 Призначення текстових шаблонів.....8
1.2 Огляд існуючих рішень для реалізації інтернет-магазину.....8
1.3 Постановка задачи.....10
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....11
2.1 Принципова схема роботи інформаційної системи.....11
2.2 Потоки даних інформаційної системи.....11
2.3 Сховища даних інформаційної системи.....12
2.4 Проектування бази даних.....14
2.5 Мережева організація інформаційної системи.....15
2.6 Проектування бекенду.....17
2.7 Проектування фронтенду.....18
3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....21
3.1 Обґрунтування засобів розробки.....21
3.2 Опис файлової структури та класів.....24
3.3 Інструкція користувача.....28
3.4 Інструкція адміністратора.....31
ВИСНОВКИ.....33
СПИСОК ЛІТЕРАТУРИ.....35
ДОДАТОК А.....36
ДОДАТОК Б.....46
ДОДАТОК В.....47

ВСТУП

Успіх будь-якого бізнесу багато в чому залежить від того, наскільки ефективно компанія працює із власною документацією. Звичайно, в сучасності багато компаній переходять до використання електронних систем документообігу. Наразі існують як складні та коштовні системи для виконання подібних задач, так і досить прості. Найпростішим та найбільш поширеним варіантом такої роботи з документами є звичайний обмін текстовими файлами з використанням офісних пакетів.

Визнаним лідером на ринку таких пакетів є компанія Microsoft із своїм продуктом Office. Найбільш використовуваною програмою з цього пакету є Microsoft Word – текстовий процесор із широкою функціональністю з оформлення тексту. Хоча широке використання цього програмного засобу має ряд власних переваг, треба мати на увазі і його недоліки. Інтерфейс програми створений таким чином, що для початку роботи із нею користувач може мати майже нульовий рівень підготовки. Тобто, інтерфейс є інтуїтивно зрозумілим. Практика показує, що такий підхід спонукає користувачів не отримувати розвинені навички роботи із програмним забезпеченням. І дійсно, користувач зазвичай не зацікавлений витрачати свій час на читання літератури, коли він відразу може отримати прийнятний результат.

Типові помилки, які допускають неосвічені користувачі при роботі з Microsoft Word:

- створення відступів «червоних» рядків за допомогою пробілів;
- призначення оформлення тексту натисканням на відповідні кнопки в панелі інструментів, замість використання гарячих клавіш;
- ручне призначення оформлення типовим блокам тексту, замість використання стилів;
- створення нового екземпляру документу через редагування старого тощо.

В рамках деякого підприємства робота в такому форматі може призводити до негативних наслідків. По-перше, робітник витрачає свій робочий час на виконання рутинних операцій, замість виконання своєї основної роботи. По-друге, невміло форматований документ на іншому комп`ютері може відображатися не так, як на тому, де він був створений, особливо якщо на них встановлені різні офісні пакети. Це може призвести до репутаційних втрат компанії або, навіть, до того, що цільовий адресант не зможе прочитати необхідний текст.

Вирішити ці проблеми в рамках підприємства можна двома шляхами. Перший – організація масового навчання робітників роботі із офісними пакетами. З точки зору кінцевого результату це найбільш ефективний спосіб подолати вищевказані проблеми, але й, в більшості випадків, економічно недоцільний.

Другий шлях – створити централізований сервіс в мережі підприємства, який би зберігав шаблони усіх необхідних документів, і за запитом генерував готовий екземпляр документу. Задача користувача в даному випадку полягає лише у тому, щоб вказати змінні дані, не чіпаючи саму форму документу.

Такий підхід дає ряд переваг. По-перше, розташування сервісу буде відомо усім робітникам підприємства. Коли будь-кому з них знадобиться створити документ, йому не доведеться витрачати час на пошук шаблону. По-друге, користувач буде вводити лише змінні дані, а значить – він не буде витрачати час на форматування та оформлення тексту і не буде допускати наведених вище помилок. І по-третє, за рахунок невеликого обсягу змінних даних, у порівнянні з вихідним документом, робітник зможе сконцентруватися на цих даних, що зменшує ймовірність змістових помилок.

В наявній роботі буде реалізована централізована система генерації документів на основі шаблонів для невеликого підприємства. При цьому будуть розглянуті такі аспекти, як аналіз існуючих рішень, постановки задачі для розробки, проектування програмного забезпечення та його фізична реалізація.

Також буде наведена демонстрація роботи системи та будуть зроблені висновки.

1 АНАЛІТИЧНИЙ ОГЛЯД ЗАДАЧІ

1.1 Призначення текстових шаблонів

Шаблон [1,2] - це вже оформлений (частково або повністю) документ, в який досить тільки ввести свої дані. До складу шаблону входить форматування (в т.ч. стилі), текст, об'єкти Word (зображення, діаграми, схеми та ін.), Стандартні блоки, а так же, призначені для користувача вкладки, команди і макроси.

Шаблони дозволяють налаштовувати всі необхідні параметри, які користувач хоче попередньо[2] застосувати до макету документів, стилі, форматування, вкладки і т.д. Потім він може легко створити новий документ на основі цього шаблону.

Доцільним є використання централізованого сховища шаблонів, куда користувач може зайти, вибрати необхідний йому документ, заповнити форму і отримати готовий екземпляр документу. Таку систему можна створити, використовуючи веб-технології.

1.2 Огляд існуючих рішень для реалізації інтернет-магазину

Генератор документів — це, перш за все, сайт, відповідно для забезпечення його працездатності необхідні веб-сервер і СУБД. В даному огляді вони не розглядаються детально; досить відзначити, що в якості веб-сервера стандартом де-факто є Apache і nginx. Що стосується СУБД, то їх вибір дуже великий [3]: починаючи від SQLite, яка підходить для невеликих проектів, і закінчуючи Oracle Database, яка застосовується в великих системах, особливо пов'язаних з фінансовими операціями.

Крім вищевказаних компонентів необхідна основа сайту - движок, який являє собою програмне забезпечення, яке працює з базою даних і відображає користувачеві необхідну інформацію.

Движки можуть бути класифіковані такими способами:

Безкоштовні та вільні. Такі програмні рішення дозволяють підприємцям використовувати системи генерації документів, не витрачаючи великі грошові суми на створення і підтримання їх інфраструктури, в чому і полягає їх основна перевага. Недоліки безкоштовних рішень - відносно висока поширеність в інтернеті, а, отже, і підвищений інтерес до них зловмисників; неадаптированность таких рішень до конкретних завдань;

Платні. Схожі з попередніми рішеннями, однак, найчастіше, мають більш високою якістю коду і мають технічну підтримаю. Такі рішення зазвичай орієнтовані на середні і великі проекти.

Унікальні. Зазвичай створюються великими компаніями на замовлення для виконання конкретних завдань.

Розглянемо деякі проекти, що реалізують функціонал генерації документів:

1С-Бітрікс з модулем «Генератор документів». 1С-Бітрікс - професійна система управління веб-проектами: сайтами компаній, інтернет-магазинами, соціальними мережами та спільнотами, корпоративними порталами, системами оренди веб-додатків і іншими проектами. «Генератор документів», який формує файли з розширенням .docx на основі створеного користувачем шаблону, а типові дані на зразок реквізитів, прізвищ, сум або таблиць з картинками автоматично підставляються в поля шаблону після запуску бізнес-процесу.

Salesman.pro з модулем «Генератор документів». Salesman.pro - Система для ведення клієнтської бази та управління продажами. Містить аналітичні звіти, розсилки, завдання і календар. Гнучка настройка прав доступу, можливо додавання позаштатних співробітників з повною ізоляції від основної бази, можливість настройки полів форм. «Генератор документів» - це модуль CRM, який дозволяє створювати і / або вести облік різних документів, прикріплювати їх до Організаціям і Угодах.

Document.sumdu.edu.ua — унікальна система, яка працює в межах Сумського державного університету. Надає інформаційне забезпечення

працівникам, студентам та аспірантам цієї установи, дозволяє онлайн заповнити шаблон та завантажити готовий екземпляр документу.

1.3 Постановка задачи

На підставі вищевказаного зробимо постановку задачі. Необхідно розробити інформаційну систему, генерувати документи за шаблоном. Метою такої системи є реалізація наступних функцій:

- Зберігання шаблонів документів;
- Забезпечення доступу до них працівникам організації, де така система працює;
- Забезпечення можливості вказати деякі змінні данні, які автоматично будуть підставлятися у шаблон;
- Забезпечення можливості завантажити оброблений документ.

Для досягнення цієї мети в роботі необхідно вирішити задачі:

- розробити інформаційні моделі задачі
- вибрати найбільш зручну для реалізації архітектуру програмного забезпечення
- добрати інструментальні та апаратні засоби з застосуванням найбільш поширеного програмного забезпечення з локалізацією до української мови;

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Принципова схема роботи інформаційної системи

Виходячи з постанови задачі виділимо типовий робочий процес інформаційної системи. В ній зберігаються шаблони документів, які являють собою docx-файли. В цих файлах побудована структура документу, але замість змінних даних в них знаходяться позначки виду {{ variable }}, які суть є полями форми. Система, окрім файлів шаблонів, зберігає інформацію про назви документів, що прив`язуються до шаблонів, а також інформацію про структуру полів документів.

Користувач, який хоче отримати готовий екземпляр документу, знає його назву. Отже, першим кроком від робить пошуковий запит до системи, щоб знайти ідентифікатор необхідного шаблону. Тепер, коли відомий ідентифікатор, користувач робить другий запит, що узнати структуру питомого шаблону, а саме кількість, типи даних та назви його полів. Після цього користувач заповнює форму даними і робить третій запит – на збірку. Він передає інформаційній системі введені дані, а вона, в свою чергу, пов`язує їх із шаблонами, замінюючи мітки-змінні дійсними даними. Далі система формує екземпляр документу і повертає його користувачеві. На цьому робочій процес закінчується.

2.2 Потоки даних інформаційної системи

Виходячи із вказаної вище схеми, ми можемо зробити висновок, що основною сутністю інформаційної системи є шаблон документу.

На основі цієї сутності ми можемо виділити два основних потоки даних в системі: створення та збірка. Також виділимо дві основі групи осіб, що приймають участь у роботі інформаційної системи: користувач та адміністратор (мається на увазі не системний адміністратор, а той, хто керує інформаційними потоками).

Адміністратор створює шаблон документу, позначає в ньому змінні дані та їхні атрибути та реєструє шаблон в системі.

Користувач може виконувати пошук необхідного шаблону, вносити в нього змінні дані та створювати запити на збірку документу. Таким чином, основі процеси системи – це керування шаблонами, та їхнє використання. Відобразимо наведені процеси на DFD-діаграмі 1-го рівня на рисунку 2.1.

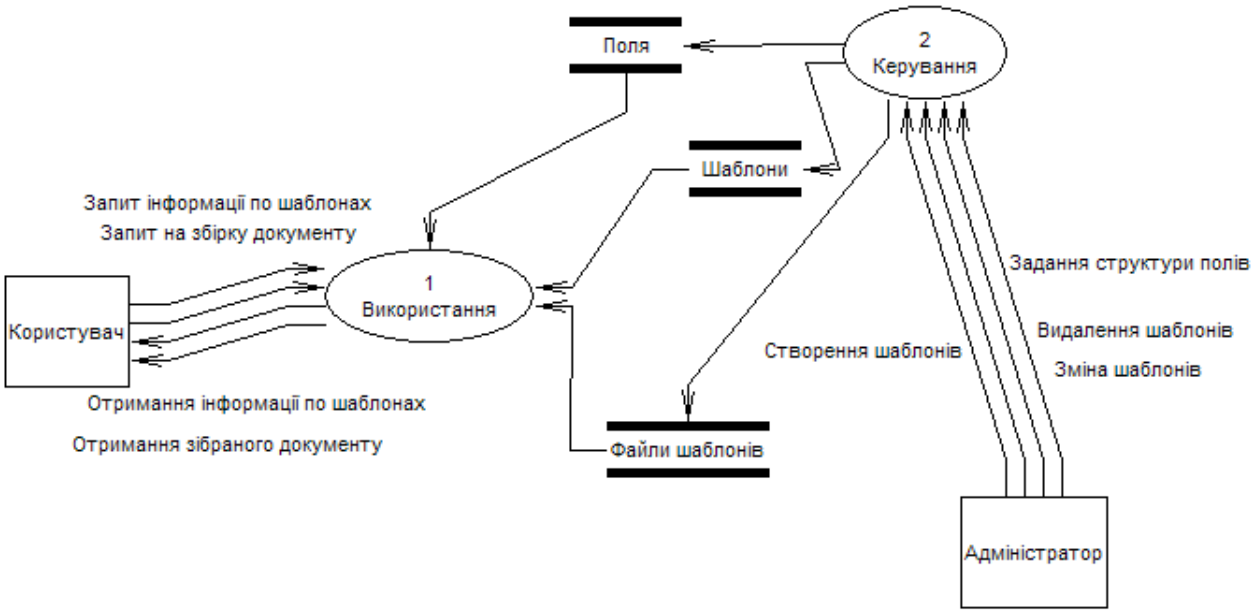

Рисунок 2.1 – DFD-діаграма 1-го рівня

2.3 Сховища даних інформаційної системи

На основі отриманих вище потоків даних виділимо 3 сховища даних: «Поля», «Шаблони», «Файли шаблонів». Нижче наведемо атрибути цих сховищ.

Шаблони (form)	
Назва	Пояснення
id	Унікальний ідентифікатор шаблону
name	Назва документу шаблону
filename	Назва файла, в якому зберігається шаблон

Таблиця 2.1 - Атрибути сховища «Шаблони»

Файли шаблонів – фізично є простим каталогом на диску	
Назва	Пояснення
filename	Назва файла, в якому зберігається шаблон

Таблиця 2.2 - Атрибути сховища «Файли шаблонів»

Поля(field)	
Назва	Пояснення
id	Унікальний ідентифікатор поля
Form_id	Ідентифікатор шаблону, до якого відноситься поле
name	Змістова назва поля
Hint	Коментарій або підказка для користувача
Field_type	Тип поля. Можливі варіанти: текст (text), число(number) та дата(date)
Var_name	Назва змінної у файлі шаблону, яка буде замінена на значення цього поля
Is_required	Чи є поле обов`язковим

Таблиця 2.3 - Атрибути сховища «Шаблони»

Відобразимо наведені сховища, їхні атрибути та зв`язки між ними на ER-діаграмі.

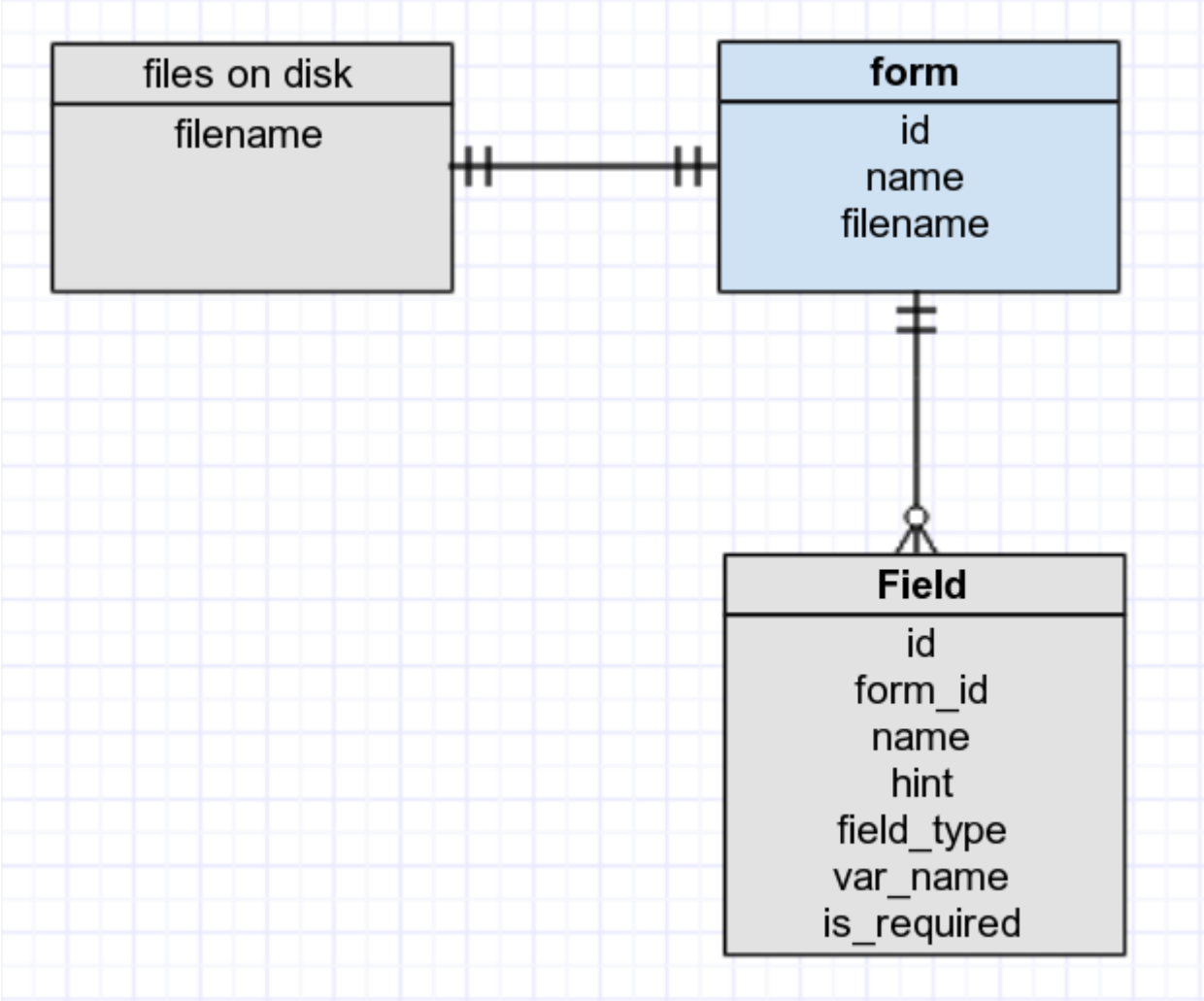

Рисунок 2.2 – ER-діаграма

2.4 Проектування бази даних

На основі виявлених сховищ даних створимо структуру зберігання цих даних в СУБД, і яка ляже в основу фізичної реалізації.

Наведемо структуру таблиць.

Form					
№	Ключ	Назва	Тип даних	Null	Unique
1	PK	Id	Integer	-	+
2		Name	Varchar(512)	-	-
3		fileName	Varchar(64)	-	-

Таблиця 2.4 – Структура таблиці Form

Field					
№	Ключ	Назва	Тип даних	Null	Unique
1	PK	Id	Integer	-	+
2		Form_id	Integer	-	-
3	FK -> form.id	Name	Varchar(128)	-	-
4		Hint	Varchar(256)	+	-
5		Field_type	Varchar(16)	-	-
6		Var_name	Varchar(64)	-	-
7		Is_required	boolean	-	-

Таблиця 2.5 – Структура таблиці Field

Оскільки файли шаблонів будуть зберігатися не в СУБД, то таблиця для цього сховища не створюється.

2.5 Мережева організація інформаційної системи

Інформаційно-програмний виріб організований по архітектурі «клієнт - сервер» В цьому випадку мережеве програмне забезпечення припускає не лише спільне використання ресурсів мережі, але і обробку на сервері по запитах користувачів. Програмне забезпечення в даному випадку складається з двох частин: сервера і клієнта. Програма клієнт виконується на локальному комп'ютері користувача, вона посилає запити програмі-серверу і приймає від неї необхідну інформацію. Програма-сервер працює на комп'ютері загального доступу, здійснює обробку запитів, що поступають до неї, і повертає клієнтові необхідні результати.

В термінології веб-розробки серверну частину інформаційної системи називаються бекендом, а клієнтську, яка виконується в браузері користувача – фронтендом. Бекенд виконує задачі зберігання, обробки та видачі даних за запитом. Фронтенд організує зручний доступ даних для користувача.

В нашому випадку бекенд виконує такі функції:

1. Видача користувачеві файлів фронтенду для відображення в браузері;

2. Отримання запитів на надання інформаціїї про шаблони та їхні структури;
3. Отримання запитів на створення екземляру документу, разом із змінними даними, а також видача цих екземплярів.

Функція №1 являє собою видачу статичного контенту і реалізується засобами веб-серверу. Функція №2 реалізується через витяг відомостей із бази даних, формування їх у форматі JSON і передачі результату фронтенду. Функція №3 реалізується через отримання даних та передачі їх особливому компоненту системи – шаблонізатору. Він, в свою чергу, виконує основне призначення ІС та повертає питомий результат користувачу.

Для ефективної організації робочого процесу була створена схема адрес, через які клієнт може отримати від бекенда тий чи інший ресурс. Бекенд реалізує принцип RESTful API, тобто можливість обміну чистими даними з клієнтом, минуючи HTML-обгортку. Розглянемо адреси доступу до ресурсів, які є актуальними в бекенді:

Метод HTTP	URI	Дія
GET	/	Видача клієнту файлу index.html, який є основним для фронтенд-компонента
GET	/static/*	Видача стилів та скриптів, необхідних для роботи фронтенда
GET	/api/search/<q>	Пошук усіх шаблонів, які містять рядок q. Видає клієнту JSON, в якому містяться ідентифікатори та повні назви шаблонів
GET	/api/forms/<id>	Видає JSON, в якому містяться відомості про поля шаблону q
POST	/api/forms/<id>/compile	Отримує список пар Ключ=Значення, де ключ назва змінної, значення – бажаний зміст, на який треба

		замінити цю змінну. За допомогою шаблонізатора проводить відповідну заміну у шаблоні id і результат надсилає клієнтові.

Таблиця 2.5 – Організація URI-ідентифікаторів

2.6 Проектування бекенду

Первинну обробку запитів виконує компонент під назвою Роутер. Він займається маршрутизацією запитів в залежності від вказаного URI, тобто ставить в залежність деякою URI деяку дію. Якщо клієнта на деякий статичний файл, то розтер просто віддає цей файл без змін. Якщо запит на деякий ресурс, який відноситься до API, то роутер виділяє із запиту ідентифікатор конкретного ресурса, звертається до відповідної моделі даних, а вона, в свою чергу, викликає методи ORM. ORM – компонент об`єктно-реляційного зв`язку. Він дає можливість пов`язати дані із СУБД із моделями даних, і там, і там зберігаючи усі зв`язки та обмеження. Також, опосередковано роутер пов`язується із шаблонізатором, задача якого показана вище.

Виходячи з вищевказаного побудуємо модель бекенду.

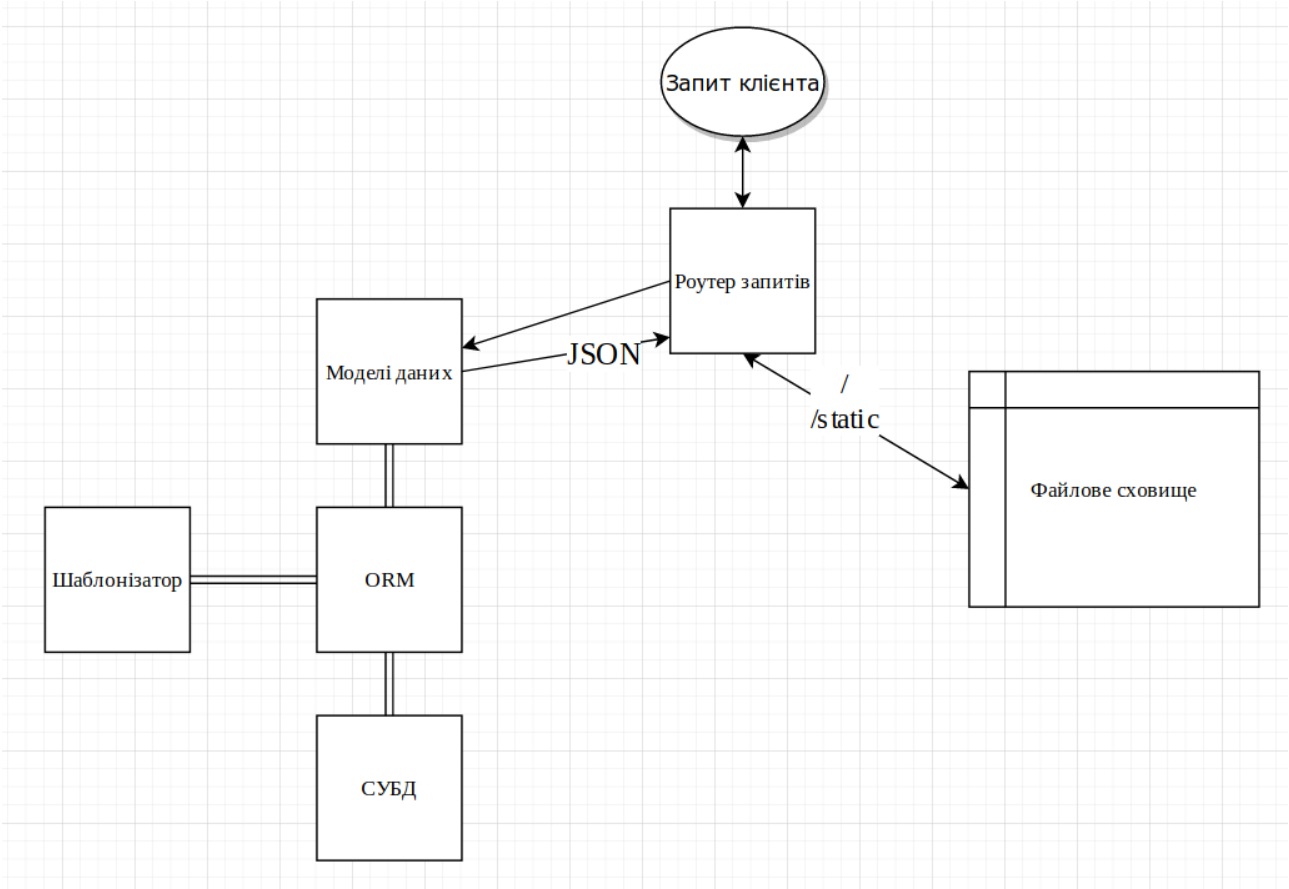

Рисунок 2.3 – Модель бекенду

2.7 Проектування фронтенду

Виходячи з постановки задачі, фронтенд буде мати наступні властивості. Він складається із однієї сторінки, на якій присутній рядок пошуку. Якщо користувач введе деяку назву, або частину назви, фронтенд негайно надішле запит бекендові на пошук. Якщо будуть знайдені будь-які шаблони, назви яких містять текст пошукового рядка, вони будуть негайно виведені під ним. Напроти кожної назви шаблону має буди відповідна кнопка редагування. Якщо її натиснути, відкриється модальне вікно, у якому будуть наявні поля вводу даних, що відповідають вказаному шаблону. Також, в цьому вікні мають виводитися підказки до полів, якщо такі наявні, а також кнопки для онулення введених даних та запиту на побудування документа. Після натискання на останню фронтенд відправить запит до бекенда і отримає файл екземпляра документа, який буде запропоновано завантажити.

Розглянемо принципову схему головного вікна фронтенда.

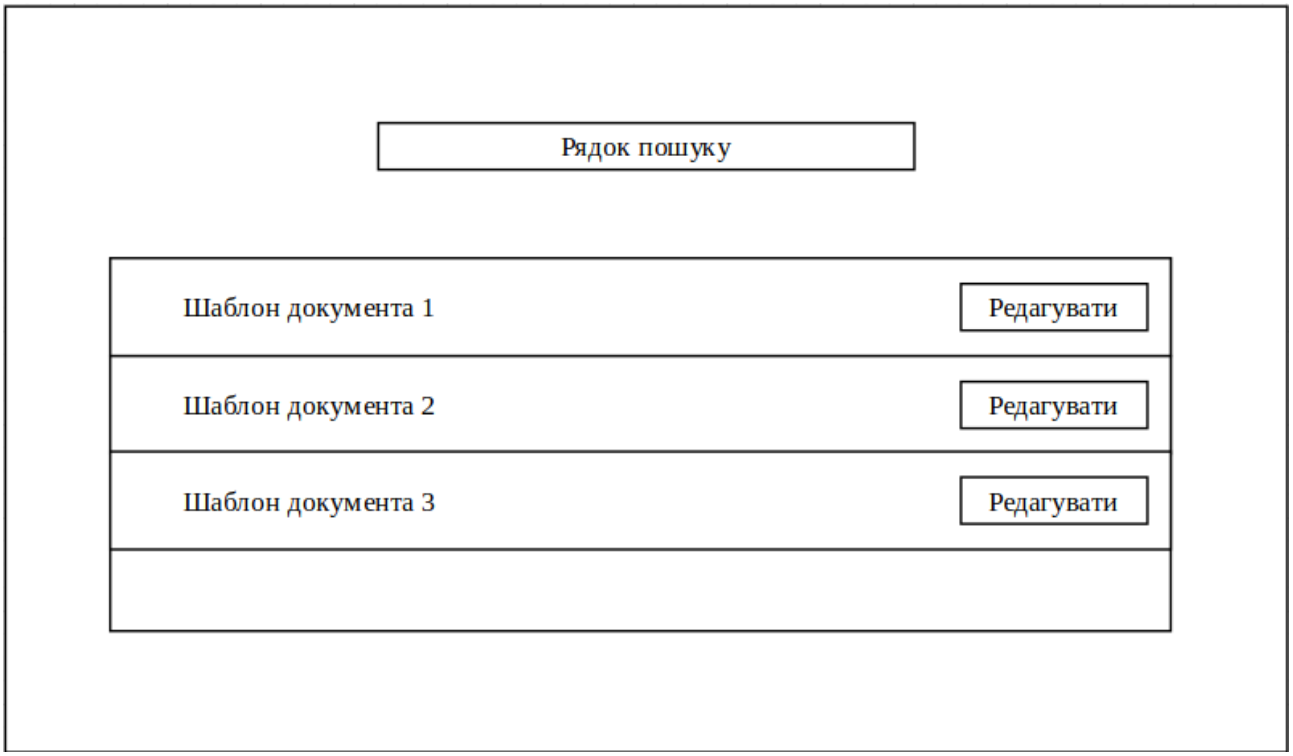

Рисунок 2.4 – Принципова схема головного вікна

Коли користувач ініціює пошук, він має побачити таке:

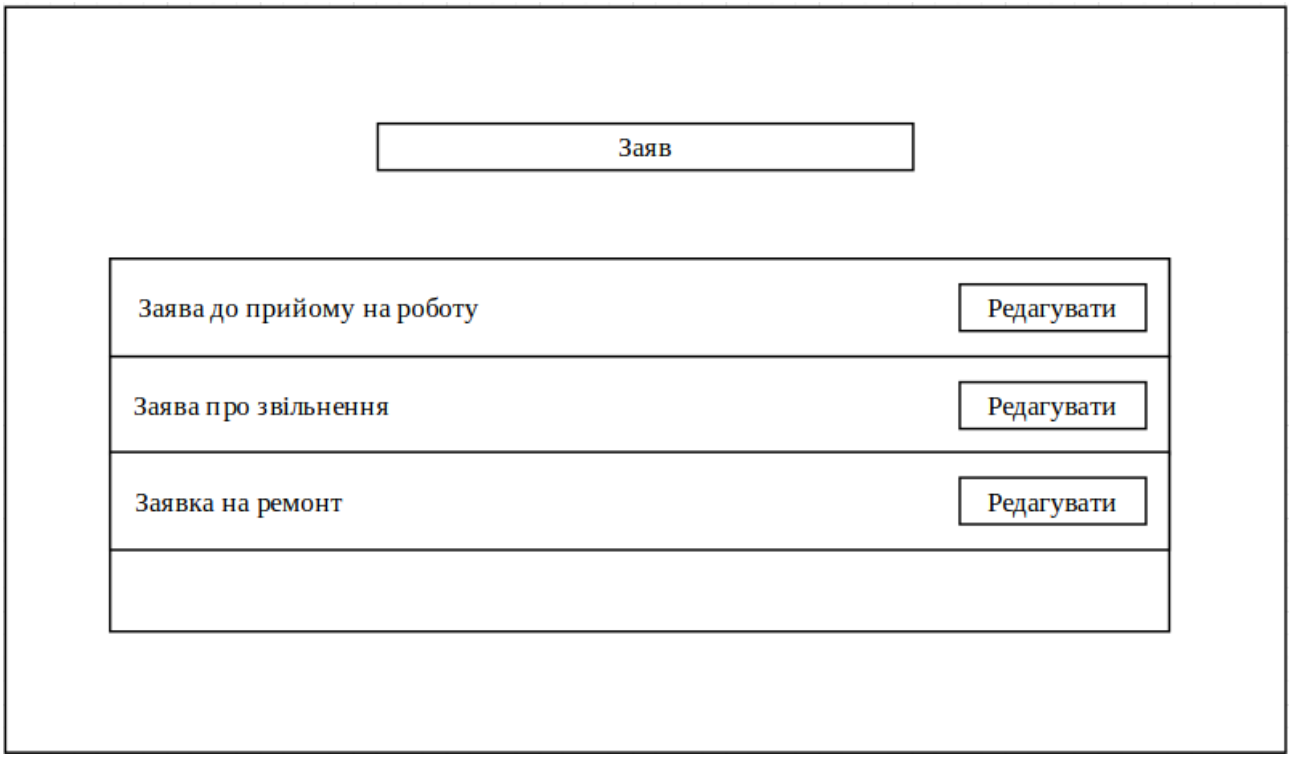

Рисунок 2.5 – Варіант вигляду головного вікна

Бачимо, що при запиті «Заяв», буде отримано усі шаблони, назви яких починаються на «Заяв».

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Обґрунтування засобів розробки

Для реалізації програмного продукту був обраний ряд ПО і технологій. При їх виборі були використані такі принципи:

- програмне забезпечення повинно бути безкоштовним;
- програмне забезпечення повинно дозволяти виконувати налагодження в домашніх умовах, тобто без необхідності щоразу вносити зміни в проект безпосередньо на web-сервері;
- незалежність від платформи;
- популярність технологій, і як результат – легкість знаходження інформації по ним.

З урахуванням наведених принципів був зроблений вибір. Перелічимо основні фреймворки та бібліотеки, для бекенду – це flask, SQLAlchemy, Alembic, docxtpl, SQLite. Для фронтенду – vue.js, bootstrap, axios, vue-form-generator.

Flask — мікрофреймворк для веб-додатків [7], створений з використанням Python. Його основу складає інструментарій Werkzeug та рушій шаблонів Jinja2. Flask називається мікрофреймворком, оскільки він не вимагає спеціальних засобів чи бібліотек. У ньому відсутній рівень абстракції для роботи з базою даних, перевірки форм або інші компоненти, які надають широковживані функції за допомогою сторонніх бібліотек. Однак, Flask має підтримку розширень, які забезпечують додаткові властивості таким чином, наче вони були доступні у Flask із самого початку. Існують розширення для встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку.

Переваги Flask:

- Містить сервер для розробки та відлагоджувач;

- Управління запитами RESTful;
- Використовує шаблони Jinja2;
- 100% відповідність WSGI 1.0;
- Підтримка Unicode;
- Докладна документація;
- Наявність розширень для забезпечення бажаної поведінки.

SQLAlchemy - це програмне забезпечення з відкритим вихідним кодом для роботи з базами даних за допомогою мови SQL[8]. Воно реалізує технологію програмування ORM (Object-Relational Mapping), яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування. SQLAlchemy дозволяє описувати структури баз даних і способи взаємодії з ними прямо на мові Python. Найбільш видатною особливістю SQLAlchemy є можливість генерації SQL-коду на основі об`єктних моделей даних, заданих у проекті.

Alembic - це інструмент для міграції бази даних, який використовується в SQLAlchemy [9]. Міграція бази даних - це щось схоже на систему контролю версій для баз даних. При розробці програми поширена практика зміни схеми таблиці. Тут і приходить на допомогу Alembic. Він, як і інші подібні інструменти, дозволяє змінювати схему бази даних при розвитку програми. Він також стежить за змінами самої бази, так що можна рухатися туди і назад. Якщо не використовувати Alembic, то за всіма змінами доведеться стежити вручну і міняти схему за допомогою Alter. В поточному проекті використовуються розширення flask-migrate для інтеграції Alembic у Flask.

docxtpl – пакет для Python, який дозволяє використовувати jinja2-розмітку у документах docx [10].

Vue.js — JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних [11].

Переваги фреймворка Vue:

- Бібліотека досить проста і функціональна. Для того щоб розібратися в ній, потрібен мінімальний багаж знань.
- Вимоги до стека відсутні, тому Vue.JS можна використовувати на будь-якому проекті.
- Небагато важить.
- Досить висока швидкість розробки. Завдяки використанню будь-яких шаблонів і доступності документації, більшість виникаючих проблем вирішуються досить швидко.

Vue дозволяє створювати функціональні додатки, які відповідають усім сучасним стандартам, при мінімальному підключенні нових ресурсів і, власне, дешевше.

Axios – HTTP-клієнт, розроблений на JavaScript для браузерів та Node.js.

vue-form-generator – модуль для vue.js, який динамічно генерує форми на основі JSON.

Bootstrap — це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків [12].

Bootstrap — це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері.

Переваги Bootstrap[13]:

- Зменшення кількості часу, що витрачається на розробку
- адаптивність
- Кросбраузерну
- Легкість у використанні і швидкість в освоєнні
- зрозумілий код
- єдність стилів

SQLite — полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту SQL-92c[14]. Сирцевий код SQLite поширюється як суспільне надбання, тобто може використовуватися без обмежень та безоплатно з будь-якою метою. Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок.

3.2 Опис файлової структури та класів

В ході розробки була створена наступна структура проекту:

app/ — основна директорія проекту, яка містить реалізацію

api/ — директорія, що містить класи, які відповідають за маршрутизацію API-запитів

forms.py — функції, які описують, як обробляти запити до «/api»

models.py — моделі даних, тобто класи, що описують сутності шаблонів та полів, і доступ до них

routes.py — функції, які описують, як обробляти запит до «/»

static/ — статичні файли проекту. В нашому випадку – все, що відноситься до фронтенду

index.html — головна сторінка фронтенду;

libs — директорія з усіма необхідними бібліотеками та стилями для фронтенду;

main.js – головний скрипт фронтенду, створює і виконує екземпляр vue.js;

app.db – файл бази даних SQLite

config.py – конфігураційний файл проекту

forms/ - директорія, що містить файли шаблонів

forms-crud.py – головний файл проекту, який запускає WSGI-сервер

forms-env/ — середовище Python, що містить усі необхідні для роботи бібліотеки

migrations/ — репозиторій змін у базі даних, керується Alembic

Після запуску сервера із сторони бекенда, завантажуються фреймворк Flask, який, в свою чергу, завантажує forms-crud.py. Цей файл відвантажує класи та ресурси і сервер починає слухати HTTP-запити. Коли такий запит поступає, Flask визначає URI і в залежності від нього запускає відповідний обробник. Обробники запитів визначені у файлах routes.py та forms.py. Розглянемо деякі обробники:

```
#routes.py
@app.route('/')
@app.route('/index')
def index():
    return app.send_static_file('index.html')
```

Якщо клієнт дасть запит до «/», то буде виконана функція index(), яка на запит дасть відповідь у вигляді файлу index.html із директорії static (директорія за замовченням для всіх статичних файлов у Flask).

```
#forms.py
@bp.route('/forms/<int:id>', methods=['GET'])
def get_form(id):
    d = {}
    fields = Field.query.filter_by(form_id=id).all()
    for f in fields:
        d[f.id] = f.repr_schema()
    return jsonify({"fields":d})
```

Якщо буде запит за URI “/forms/<id>”, то Flask вилучить із запиту значення <id>, та запустить функцію get_form(id), передавши їй цей параметр. Ця функція за допомогою ORM-методів у моделі Field вилучає із бази даних усі записи, де form_id=id. Відразу ж формуються об`єкти типу Field, для яких вилучаються значення repr_schema(). Ці значення вносяться у словник d, який потім перетворюється на JSON-відповідь і відправляється клієнтові.

Розглянемо реалізацію моделі Field:

```
class Field(db.Model):
        id = db.Column(db.Integer, primary_key=True, nullable=False, unique=True)
            form_id = db.Column(db.Integer, db.ForeignKey('form.id'), nullable=False)
        name = db.Column(db.String(128), index=True, nullable=False)
        hint = db.Column(db.String(256))
        field_type = db.Column(db.String(16), nullable=False)
        var_name = db.Column(db.String(64), nullable=False)
        is_required = db.Column(db.Boolean, nullable=False)

        def repr_schema(self):
            schema = {
                "type": "input",
                "inputType": self.field_type,
                "label": self.name,
                "model": self.var_name,
```

```
            "inputName": self.var_name,
            "hint": self.hint,
            "required": self.is_required
        }
        return schema

    def __repr__(self):
            return '<Field {0}, {1}, {2}>'.format(self.form, self.name, self.field_type)
```

Бачимо, що змінні класу задаються у форматі SQLAlchemy. Також, досліджуючи цей клас, Alembic може сформувати відповідну таблицю. Метод repr_schema, як було вище вказано, необхідний для формування JSON-відповіді. Такий формат словнику необхідний для подальшої інтерпретації JSON модулем vue-form-generator, який створить форму з боку фронтенду для вводу даних користувачем.

Формування екземпляру документу проводиться у наступній функції:

```
@bp.route('/forms/<int:id>/compile', methods=['POST'])
def compile_form(id):
    form = Form.query.filter_by(id=id).first_or_404()
    filename = os.path.join(app.config['FORMS_DIR_PATH'], form.filename)
    doc = DocxTemplate(filename)
    doc.render(request.args.to_dict())

    stream = BytesIO()
    doc.get_docx().save(stream)
    stream.seek(0)
            return send_file(stream, as_attachment=True, attachment_filename='result.docx', mimetype='docx')
```

Спершу, метод по id шаблону знаходить файл, у якому той міститься. Потім, файл відкривається, із запиту вилучаються дані форми, які

підставляються у шаблон. Отриманий документ зберігається у пам`яті, після чого він віправляється клієнтові.

Всі основі вихідні файли наводяться у додатку А.

3.3 Інструкція користувача

Щоб працювати із системою користувач має зайти на відповідний веб-сайт, на головну сторінку. Він побачить наступне вікно:

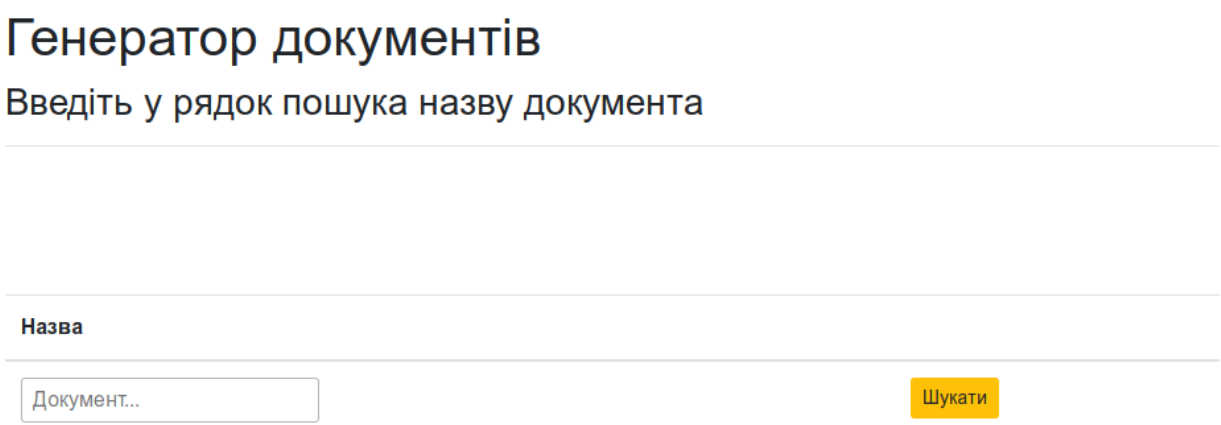

Рисунок 3.1 – Вигляд головного вікна додатку

Після цього користувач може у рядок пошуку ввести назву документа, або її частину. Якщо у користувача є проблеми із з`єднанням, він може натиснути кнопку «Шукати». На наступному скриншоті бачимо, що користувач ввів «Зая» і було знайдено документ «Заява про відгул».

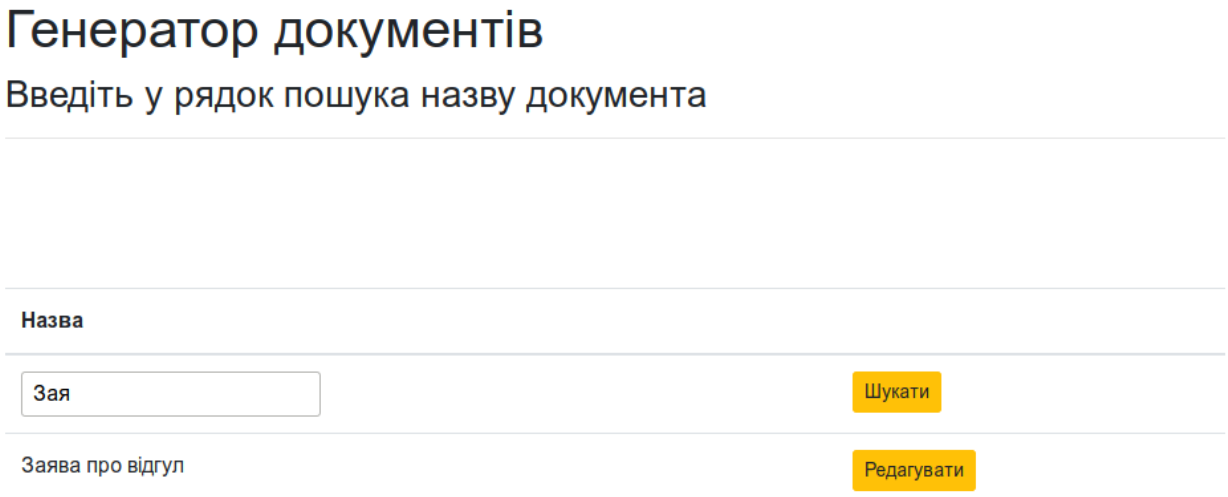

Рисунок 3.2 – Результат пошуку у головному вікні

Далі користувач має вибрати документ, який він хоче отримати і натиснути відповідну йому кнопку «Редагувати». З`явиться модальне вікно редагування форми. Далі користувач має вибрати документ, який він хоче отримати і натиснути відповідну йому кнопку «Редагувати». Далі користувач має вибрати документ, який він хоче отримати і натиснути відповідну йому кнопку

«Редагувати». З`явиться модальне вікно редагування форми.

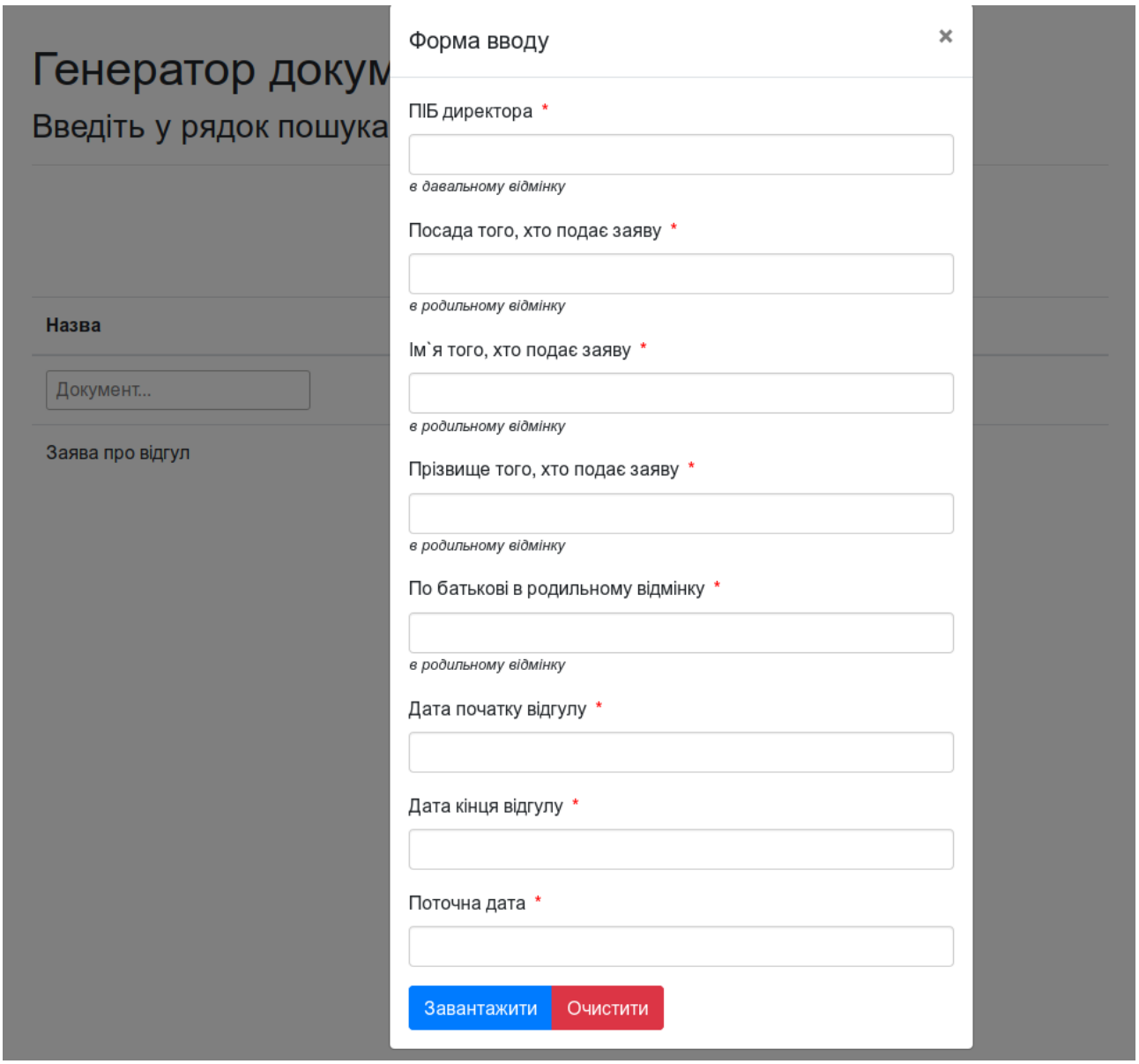

Рисунок 3.3 – Форма пошуку

Далі необхідно ввести бажані дані.

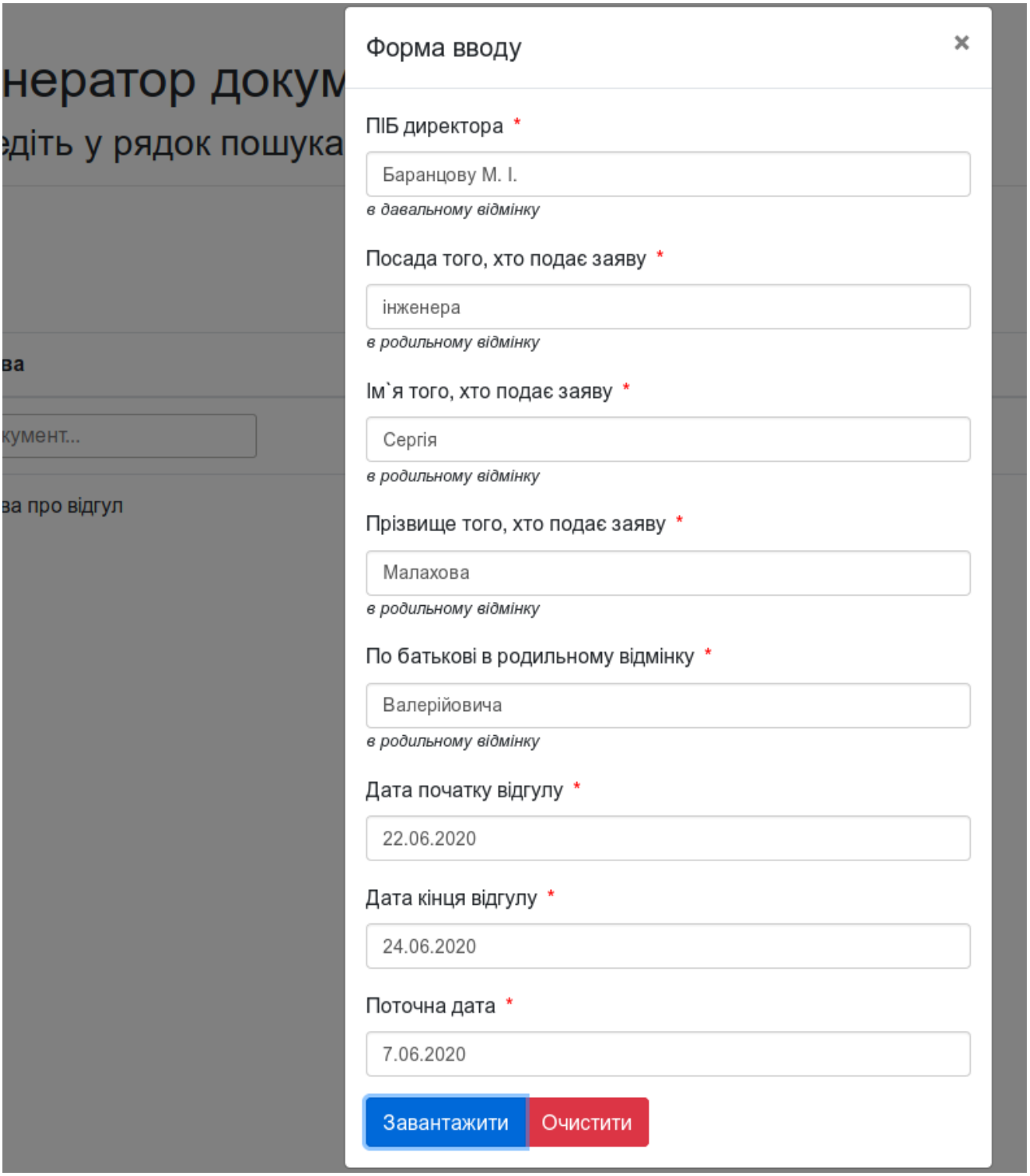

Рисунок 3.4 – Заповнена форма пошуку

Якщо треба очисти поля вводу, треба натиснути кнопку «Очистити».

Після вводу треба натиснути кнопку «Завантажити». Користувачеві буде запропоновано скачати файл.

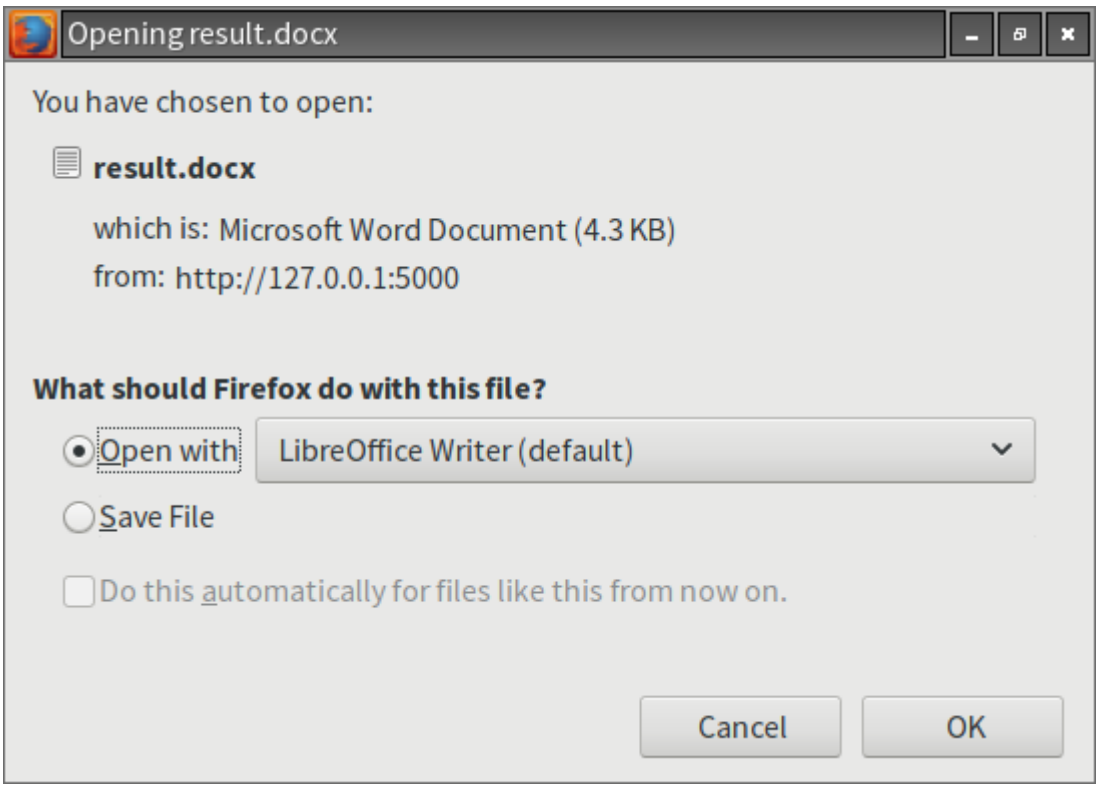

Рисунок 3.5 – Завантаження готового екземпляру документа

Приклади шаблону і результату підготовки документа наведені у додатку Б.

3.4 Інструкція адміністратора

Для роботи додатку необхідно наступне ПЗ:

Ubuntu Linux >=18.04

Пакети python3, python3-venv, python3-dev, nginx

Адміністратор має створити окремого користувача в системі для роботи додатку. В домашній каталог треба розмістити дистрибутив додатка forms-crud. Далі, треба виконати наступні команди:

```
$ cd forms-crus
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install -r requirements.txt
```

У файлі config.py треба налаштувати наступні параметри:

SQLALCHEMY_DATABASE_URI – розташування SQLite бази даних;

FORMS_DIR_PATH – розташування файлів шаблонів.

Далі адміністратор має створити конфігурацію nginx, враховуючи усі шляхи установки. Приклад конфігурації наданий у додатку В.

Для додання нових шаблонів адміністратор має розмістити самі шаблони у каталогу FORMS_DIR_PATH. Для створення форми вводу він має будь-яким SQLite-редактором відкрити файл бази даних SQLALCHEMY_DATABASE_URI і, користуючись структурою БД із попередніх розділів, внести форми за наступним прикладом:

```
insert into form values (1, 'Заява про відгул', 'test1.docx');
insert into field values (7, 1, 'ПІБ директора', 'в давальному відмінку', 'text', 'dir_nsp', 1);
insert into field values (8, 1, 'Посада того, хто подає заяву', 'в родильному відмінку', 'text', 'position', 1);
insert into field values (9, 1, 'Ім`я  того, хто подає заяву', 'в родильному відмінку', 'text', 'name', 1);
insert into field values (10, 1, 'Прізвище того, хто подає заяву', 'в родильному відмінку', 'text', 'surname', 1);
insert into field values (11, 1, 'По батькові в родильному відмінку', 'в родильному відмінку', 'text', 'patronymic', 1);
insert into field values (12, 1, 'Дата початку відгулу', '', 'date', 'date_start', 1);
insert into field values (13, 1, 'Дата кінця відгулу', '', 'date', 'date_end', 1);
insert into field values (14, 1, 'Поточна дата', '', 'date', 'date_current', 1);
```

Запуск додатка проводиться за допомогою команди flask run.

ВИСНОВКИ

Генератор документів на основі шаблонів – це веб-сервіс, який допомогає створювати неосвіченим користувачам добре сформатовані документи. Це досягається за рахунок того, что користувач не оформлює весь документ цілком – замість цього він може використати розроблену в цьому проекті інформаційну систему. В такому випадку користувачеві буде досить ввести тільки змінні дані документу, які будуть підставлені в заздалегідь відформатований шаблон.

В рамках даного дипломного проекту була розроблена така система, при цього був використаний ряд сучасних веб-технологій. А саме: бекенд-фреймворк Flask та фронтенд фрейворки Vue.js та Bootstrap. Була спроектована база даних, та реалізована за допомогою фреймворка SQLAlchemy та СУБД SQLite.

Позитивними якостями розробленого веб-сервісу є:

1. Зручний інтерфейс користувача;
2. Відсутність реєстрації клієнтів магазину, що забезпечує їхню анонімність та економить їхній час.
3. Сортування товарів по групах, що пришвидшує пошук потрібного товару.

Негативними якостями Інтернет-магазину є:

1. Відсутність панелі адміністратора;
2. Неможливість використання складних виразів у шаблонах, замість простих змінних.

Однак, слід враховувати, що для реалізації обох цих пунктів треба реалізувати автоматичне вилучення виразів із файлу шаблону і використання більш продвинутих технік розробки. При цьому всьому розроблена система здатна покрити потреби невеликого підприємства для обслуговування електронного документооберту невеликих масштабів.

Враховуючи вище сказане можна зробити висновок, що всі поставленні завдання в роботі виконані в повному обсязі.

СПИСОК ЛІТЕРАТУРИ

1. https://officelegko.com/2017/09/26/shablonyi-v-microsoft-word/
2. https://guidepc.ru/applications/microsoft-word/kak-sozdat-shablon-v-microsoft-word/
3. Популярные Web-сервисы: практика использования. Айверсон Уилл. КУДИЦ – ОБРАЗ, 2005.
4. Россум Г. Язык программирования python, 2010.
5. Б.А. Курс по библиотеке Tkinter языка python, 2008.
6. Б.А. Язык программирования python Ч1-Ч3, 2004.
7. https://uk.wikipedia.org/wiki/Flask
8. https://ru.wikibooks.org/wiki/SQLAlchemy
9. https://pythonru.com/uroki/16-migracii-bazy-dannyh-s-pomoshhju-alembic
10. https://pypi.org/project/docxtpl/
11. https://jetruby.com/ru/blog/vue-js-preimuschestva-i-nedostatki/
12. https://uk.wikipedia.org/wiki/Bootstrap
13. https://timeweb.com/ru/community/articles/plyusy-i-minusy-bootstrap-1
14. https://uk.wikipedia.org/wiki/SQLite

ДОДАТОК А

Лістинг файлу models.py.

```
from app import db


class Form(db.Model):
    id = db.Column(db.Integer, primary_key=True, nullable=False, unique=True)
    #version_id = db.Column(db.Integer, nullable=False, default=1)
    name = db.Column(db.String(512), index=True, nullable=False)
    filename = db.Column(db.String(64), index=True, nullable=False, unique=True)
    fields = db.relationship('Field', backref='form', lazy='dynamic')

    def __repr__(self):
        return '<Form {}>'.format(self.name)


class Field(db.Model):
    id = db.Column(db.Integer, primary_key=True, nullable=False, unique=True)
    form_id = db.Column(db.Integer, db.ForeignKey('form.id'), nullable=False)
    name = db.Column(db.String(128), index=True, nullable=False)
    hint = db.Column(db.String(256))
    field_type = db.Column(db.String(16), nullable=False)
    var_name = db.Column(db.String(64), nullable=False)
    is_required = db.Column(db.Boolean, nullable=False)

    def repr_schema(self):
        schema = {
            "type": "input",
            "inputType": self.field_type,
            "label": self.name,
            "model": self.var_name,
```

```
            "inputName": self.var_name,
            "hint": self.hint,
            "required": self.is_required
        }
        return schema

    def __repr__(self):
            return '<Field {0}, {1}, {2}>'.format(self.form, self.name, self.field_type)
```

Лістинг файлу routes.py

```
from app import db



class Form(db.Model):

    id = db.Column(db.Integer, primary_key=True, nullable=False, unique=True)

    #version_id = db.Column(db.Integer, nullable=False, default=1)

    name = db.Column(db.String(512), index=True, nullable=False)

    filename = db.Column(db.String(64), index=True, nullable=False, unique=True)

    fields = db.relationship('Field', backref='form', lazy='dynamic')



    def __repr__(self):

        return '<Form {}>'.format(self.name)
```

```
class Field(db.Model):

    id = db.Column(db.Integer, primary_key=True, nullable=False, unique=True)

    form_id = db.Column(db.Integer, db.ForeignKey('form.id'), nullable=False)

    name = db.Column(db.String(128), index=True, nullable=False)

    hint = db.Column(db.String(256))

    field_type = db.Column(db.String(16), nullable=False)

    var_name = db.Column(db.String(64), nullable=False)

    is_required = db.Column(db.Boolean, nullable=False)



    def repr_schema(self):

        schema = {

            "type": "input",

            "inputType": self.field_type,

            "label": self.name,

            "model": self.var_name,
```

```
            "inputName": self.var_name,
            "hint": self.hint,
            "required": self.is_required
        }
        return schema

    def __repr__(self):
        return '<Field {0}, {1}, {2}>'.format(self.form, self.name, self.field_type)
```

Лістинг файлу api/forms.py

```
import os
from io import BytesIO

from flask import request, jsonify, send_file
from app.api import bp
from app import app
from app.models import Form, Field

from docxtpl import DocxTemplate

@bp.route('/search/<string:q>', methods=['GET'])
def search_form(q):
    l = []
    search = "%{}%".format(q)
```

```
    forms = Form.query.filter(Form.name.like(search)).all()
    for f in forms:
        d = {}
        d['id'] = f.id
        d['name'] = f.name
        l.append(d)
    return jsonify(l)

@bp.route('/forms/<int:id>', methods=['GET'])
def get_form(id):
    d = {}
    fields = Field.query.filter_by(form_id=id).all()
    for f in fields:
        d[f.id] = f.repr_schema()
    return jsonify({"fields":d})


@bp.route('/forms/<int:id>/compile', methods=['GET','POST'])
def compile_form(id):
    form = Form.query.filter_by(id=id).first_or_404()
    filename = os.path.join(app.config['FORMS_DIR_PATH'], form.filename)
    doc = DocxTemplate(filename)
    doc.render(request.args.to_dict())

    stream = BytesIO()
    doc.get_docx().save(stream)
    stream.seek(0)
                    return send_file(stream, as_attachment=True,
attachment_filename='result.docx', mimetype='docx')
```

Лістинг файлу index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
```

```
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
            <meta name="viewport" content="width=device-width,initial-scale=1.0">
        <link rel="icon" href="favicon.ico">

                                <link rel="stylesheet" type="text/css" href="static/libs/styles/bootstrap.min.css">
                                <link rel="stylesheet" type="text/css" href="static/libs/styles/bootstrap-vue.css">
                                <link rel="stylesheet" type="text/css" href="static/libs/styles/vfg.css">

        <script type="text/javascript" src="static/libs/vue.min.js"></script>
                                        <script type="text/javascript" src="static/libs/bootstrap-vue.js"></script>
            <script type="text/javascript" src="static/libs/bootstrap-vue-icons.js"></script>
        <script type="text/javascript" src="static/libs/vfg.js"></script>
        <script type="text/javascript" src="static/libs/axios.js"></script>

        <title>Генератор документів</title>
      </head>
    <style>
    #app {
     margin-top: 60px
    }
    </style>
      <body>
        <noscript>
            <strong>We're sorry but client doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
        </noscript>
        <div id="app"><div class="container">
        <div class="row">
```

```
            <div class="col-sm-10">
              <h1>Генератор документів</h1>
              <h3>Введіть у рядок пошука назву документа</h1>
              <hr><br><br>
              <br><br>
              <table class="table table-hover">
                <thead>
                  <tr>
                    <th scope="col">Назва</th>
                    <th></th>
                  </tr>
                </thead>
                <tbody>
                  <tr>
                                        <td><input   type="text"   v-model="search"
placeholder="Документ..."/> </td>
                    <td>
                      <div class="btn-group" role="group">
                        <button
                              type="button"
                              class="btn btn-warning btn-sm"
                              @click="getForms">
                           Шукати
                        </button>
                      </div>
                    </td>
                  </tr>
                  <tr v-for="(form, index) in forms" :key="index">
                    <td>{{ form.name }}</td>
                    <td>
                      <div class="btn-group" role="group">
                        <button
                              type="button"
                              class="btn btn-warning btn-sm"
```

```
                        v-b-modal.form-edit-modal
                        @click="editForm(form)">
                      Редагувати
                    </button>
                   </div>
                  </td>
                 </tr>
                </tbody>
               </table>
              </div>
             </div>
             <b-modal ref="editFormModal"
                  id="form-edit-modal"
                  title="Форма вводу"
                  hide-footer>
                  <b-form :action="compileAddress" @reset="onReset" class="w-
100">
                                                                  <vue-form-
generator                  :schema="schema"                  :model="model"
:options="formOptions"></vue-form-generator>
               <b-button-group>
                   <b-button type="submit" variant="primary">Завантажити</b-
button>
                <b-button type="reset" variant="danger">Очистити</b-button>
               </b-button-group>
              </b-form>
             </b-modal>
            </div>
            </div></div>
             <script type="text/javascript" src="static/main.js"></script>
            </body>
           </html>
```

Лістинг файлу main,js

```
var vm = new Vue({
    el: "#app",

    components: {
        "vue-form-generator": VueFormGenerator.component
    },

    data() {
        return {
            search: '',
            forms: [],
            model: {},
            schema: {},
            formOptions: {
                validateAfterLoad: true,
                validateAfterChanged: true
            },
            compileAddress: ""
        };
    },


    methods: {
        getForms() {
            console.log(this.search);
            const path = '/api/search/Зая';
            axios.get(path)
                .then((res) => {
                    console.log(this.search);
                    this.forms = res.data;
                })
```

```
            .catch((error) => {
                // eslint-disable-next-line
                console.error(error);
            });
        console.log(this.$refs);
    },
    editForm(form) {
        const path = `/api/forms/${form.id}`;
        axios.get(path)
            .then((res) => {
                this.schema = res.data;
                console.log(res.data);
            })
            .catch((error) => {
                // eslint-disable-next-line
                console.error(error);
            });
        this.compileAddress = `/api/forms/${form.id}/compile`;
        //this.$refs.editFormModal.show();
    },
    onSubmit(evt) {
        evt.preventDefault();
        const path = `/api/forms/${this.currentFormID}/compile`;
        console.log(this.model);
        //evt.preventDefault();
        //this.$refs.formSubmit.submit();
    },
    onReset(evt) {
        evt.preventDefault();
        this.model = {};
    }
},
created() {
    this.getForms();
```

```
    },


    computed: {

    }

});
```

ДОДАТОК Б

Приклад шаблону:

Директору сумської філії ПАТ
“Укртелеком”
{{ dir_nsp }}
від {{ position }}
{{ surname }} {{ name }}
{{ patronymic }}

Заява

Прошу надати мені відгул з {{ date_start }} по {{ date_end }}.

{{ date_current }} (підпис)

Приклад результату обробки:

Директору сумської філії ПАТ
“Укртелеком”
Баранцову М. І.
від інженера
Малахова Сергія
Валерійовича

Заява

Прошу надати мені відгул з 22.06.2020 по 24.06.2020.

7.06.2020 (підпис)

ДОДАТОК В

Приклад конфігурації серверу nginx.

```
server {
    listen 80;
    server_name _;
    location / {
        return 301 https://$host$request_uri;
    }
}
server {
    listen 443 ssl;
    server_name _;



    ssl_certificate /home/ubuntu/forms-crud/certs/cert.pem;
    ssl_certificate_key /home/ubuntu/forms-crud/certs/key.pem;
    access_log /var/log/forms-crud_access.log;
    error_log /var/log/forms-crud_error.log;

    location / {
        proxy_pass http://localhost:5000;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /static {
        alias /home/ubuntu/forms-crud/static;
        expires 30d;
    }
}
```