Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Конструктор сторінок HTML на основі деревоподібної структури даних**»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Мединського Яреми

Керівник Бабич Т.А.

магістр комп’ютерних наук, асистент

«6» червня 2022 р.

Київ – 2022

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“10” жовтня 2021 року

Завдання для курсової студенту

Мединському Яремі

- Тема роботи — «**Конструктор сторінок HTML на основі деревоподібної структури даних**» керівник роботи Бабич Трохим Анатолійович, магістр комп’ютерних наук, асистент
- Строк подання студентом роботи — 7 червня 2022
- План роботи
    - Анотація
    - Вступ
    - Розділ 1. Дослідження та аналіз предметної області
        - Аналіз готових рішень
        - Flask-фреймворк для написання веб-застосунків на Python
        - Об’єктна модель гіпертекстового документу
        - Реалізація деревовидної структури на клієнтській стороні
    - Розділ 2. Проєктування та розробка системи
        - ER-модель даних для додатку
        - Серверна частина

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	10 жовтня 2021			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	10 жовтня 2021 – 2 листопада 2021			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	2 листопада 2021			
4.	Написання розділів роботи	2 листопада 2021 – 01 березня 2022			
5.	Проміжний контроль виконання роботи	01 лютого 2022			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	11 січня 2022 – 29 березня 2022			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25 січня 2022			
	Розділ 2 (аналітично-дослідницька частина)	01 березня 2022			
	Розділ 3 (проектно-рекомендаційна частина)	29 березня 2022			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	01 квітня 2022 – 06 травня 2022			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	6 червня 2022			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 10 жовтня 2020 р.
Науковий керівник Бабич Трохим Анатолійович
Виконавець курсової роботи Мединський Ярема

Зміст

Анотація

У даній роботі розглядаються особливості створення конструктора HTML-сторінок. Пояснюється вибір бібліотеки Jstree для динамічного відображення деревовидної структури HTML-документу.

Вступ

На ринку є багато готових рішень конструкторів HTML-сторінок такі як Wordpress, Wix та ін. Ці рішення дуже комплексні, тому я вирішив розібратися і зробити щось простіше, що було б зрозуміло людині, яка хоч трохи знає HTML та CSS.

Метою цієї роботи є дослідити побудову деревоподібних структур, і розібратися як за допомогою неї будувати HTML структуру.

Розділ 1. Дослідження та аналіз предметної області

1.1 Аналіз готових рішень

Аналізуючи ринок готових рішень я зробив висновок, що всі додатки для побудови HTML-сторінок можна поділити на 2 типи. Перші – це прості, інтуїтивно зрозумілі, де не потрібно знати ні HTML ні CSS, і все можна зробити методом Drag-and-drop(перетягування мишкою потрібних елементів). Другий – це більш складні, комплексні, де можна вставляти свій власний HTML та CSS, і навіть писати свою серверну частину для обробки запитів на сервер.

До перших можна віднести:

- Wix
- Tilda
- Weebly

До других:

- WordPress
- Webflow

Другі скоріше можна віднести до CMS(Система керування вмістом) з можливістю редагування елементів сайту вручну. З цього я можу зробити висновки, що зараз все більше і більше конструкторів сайтів створюється для людей далеких від програмування і верстки. Навіть WordPress, який раніше позиціонувався як CMS, створив систему плаґінів через яку сторонні розробники можуть додавати свій функціонал, який буде спрощувати розробку сайту. Досить популярний плаґін-конструктор для WordPress – Elementor. Він надає змогу будувати сторінки без необхідності розбиратися в HTML та CSS.

Я вирішив зробити систему для побудови сторінок без високорівневих надбудов. В моєму додатку треба вписувати HTML та CSS кожному елементу вручну. Це надає користувачу гнучкість в написанні сторінки.

1.2 Flask-фреймворк для написання веб-застосунків на Python

Flask — це мікрофреймворк для вебдодатків, створений з використанням Python. Його основу складає інструментарій Werkzeug та рушій шаблонів Jinja2. Поширюється відповідно до умов ліцензії BSD. 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

1.3 Об’єктна модель гіпертекстового документу

HTML(HyperText Markup Language) - стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. Веб-браузери надсилають HTTP запит і у відповідь отримують HTML документ від сервера за протоколами або відкривають з локального диска, далі переводять код у відображення, яке відображатиметься на екрані монітора. Елементи HTML є будівельними блоками сторінок HTML. За допомогою конструкцій HTML, зображення та інші об'єкти, такі як посилання, інтерактивні форми для запитів можуть бути використані в сторінці. HTML - це засіб для створення структурованих документів, які позначають структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Через те, що браузер перетворює всі теги в об’єкти, то ці теги не відображаються на сторінці.

HTML в браузері представляється у вигляді об’єктів. І це представлення називають Document Object Model (Об’єктна модель документу) надалі - DOM.

З точки зору об'єктно-орієнтованого програмування, DOM визначає класи, методи та атрибути цих методів для роботи із представленям документів у

вигляді дерева. Все це для того, аби надати можливість комп'ютерній програмі доступу та динамічної модифікації вмісту, структури та оформлення документа.

Тобто сервер присилає HTML документ на браузер, і вже браузер будує DOM, використовуючи який він може побудувати графічне представлення HTML-документа.

1.4 Реалізація деревовидної структури на клієнтській стороні

Для навігації по елементах було вибрано бібліотеку jstree яка базується на популярній бібліотеці jQuery.

jQuery — популярна JavaScript-бібліотека з відкритим кодом. jQuery є найпопулярнішою бібліотекою JavaScript, яка часто використовується в сьогоднішніх веб-застосунках. jQuery є безкоштовним програмним забезпеченням під ліцензією MIT. Синтаксис jQuery створений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плаґінів(як jstree). Використовуючи ці об'єкти, розробники плаґінів можуть створювати абстракції для низькорівневого коду браузера.

jsTree — це плаґін jquery , який використовується для інтерактивної побудови дерев. Він є абсолютно безкоштовним з відкритим вихідним кодом і поширюється за ліцензією MIT. jsTree легко розширюється, тематизується і налаштовується, він підтримує джерела даних HTML і JSON і завантаження AJAX .

jsTree належним чином функціонує в будь-якій моделі box-box (content-box або border-box), може бути завантажений як модуль AMD і має вбудовану мобільну тему для адаптивного дизайну, яку можна легко налаштувати. Він використовує

систему подій jQuery, тому зв’язування зворотних викликів для різних подій у дереві є знайомим і простим.

Деревовидна структура формується після завантаження HTML-сторінки, використовуючи AJAX запит на сервер. Серверна частина присилає дані у вигляді JSON(JavaScript Object Notation). Далі jstree автоматично будує деревовидну структуру на сторінці.

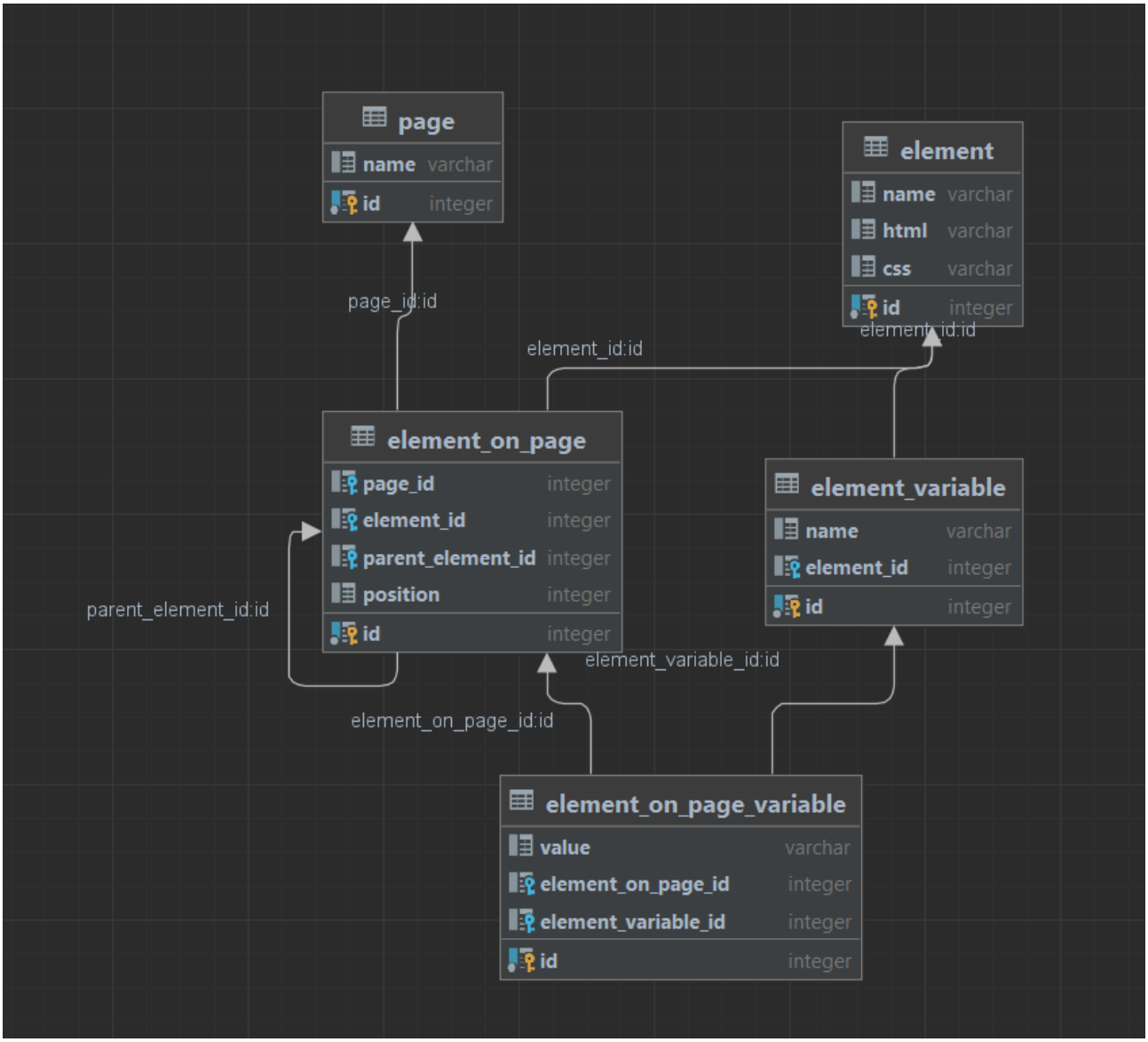
Завдяки DOM api та JS можна побудувати деревовидну структуру елементів без використання рекурсивного алгоритму, що робить алгоритм додавання інтуїтивно зрозумілим для інших програмістів.

Розділ 2. Проєктування та розробка системи

2.1 ER-Модель даних додатку

При проєктуванні бази даних враховувалась ефективність у використанні пам’ять. В першу чергу – це перевикористання пам’яті, яка вже використовується. Тобто якщо один і то й же елемент використовується в різних місцях на сторінці, то не треба створювати два однакові елементи. Можна створити один елемент і подати посилання на нього в певному місці на сторінці.

Також реалізовано змінні, які можна заповнити один раз, і використовувати в будь-якому елементі на сторінці.



Для реалізації бази даних була використана SQLAlchemy.

SQLAlchemy - це ORM(Об’єктно реляційна проєкція). Це пришвидшило розробку і збільшило надійність система через те, що там програміст не пише SQL запити вручну. За нього SQL-запит формується автоматично системою ORM. Це автоматично виключає можливість SQL-ін’єкції та інших вразливостей, які будуть присутні в чистому SQL запиту без окремої оброки.

```
class Page(db.Model):
    __tablename__ = 'page'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String)
    element_on_page = db.relationship("ElementOnPage", lazy="dynamic", cascade="all,delete")
```

Рисунок 1. Батьківський клас

Для цього програмістові треба створити новий клас, як сутність бази даних, і унаслідувати його від класу Model. Після цього треба створити поля, як атрибути в базі даних, давши їм значення об’єкту класу Column з параметрами того поля, яке вам потрібно. Зв’язуються таблиці як і в простій базі даних через зовнішній ключ в класі дитини, але, окрім того, ще треба обов’язково вказати зв’язок в батьківському класі.

```
class ElementOnPage(db.Model):
    __tablename__ = 'element_on_page'

    id = db.Column(db.Integer, primary_key=True)
    page_id = db.Column(db.Integer, db.ForeignKey('page.id'))
    element_id = db.Column(db.Integer, db.ForeignKey('element.id'))
    parent_element_id = db.Column(db.Integer, db.ForeignKey('element_on_page.id'))
    children_element = relationship("ElementOnPage", lazy="dynamic", cascade="all, delete")
    element_on_page_variable = relationship("ElementOnPageVariable", lazy="dynamic")
    position = db.Column(db.Integer)
```

Рисунок 2. Клас дитини

В цьому зв’язку ще можна описати параметри зв’язку. Наприклад вказати, що робити при видалення батьківського елементу з дитячим елементом. У моєму випадку було використано каскадне видалення всіх дитячих елементів. Тобто

якщо видаляється батьківський елемент, то всі діти, які належали йому, теж видаляються.

Також для створення, видалення та взаємодії з цими даними можна створювати методи для цього.

```
def __init__(self, name):
    self.name = name
```

Рисунок 3. Конструктор класу Page

```
def save_to_db(self):
    db.session.add(self)
    db.session.commit()
    return self.id

def delete_from_db(self):
    db.session.delete(self)
    db.session.commit()
```

Рисунок 4. Методи зберігання та видалення об'єктів реляції

2.2 Серверна частина

Як вже зазначалось вище код для сервера було вирішено писати на Python, використовуючи фреймворк Flask. Flask може працювати як і з SSR(Server-side rendering), так і з REST(Representational State Transfer).

Для SSR необхідно мати один сервер, який власне й буде надсилати HTML до браузера.

REST – це підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Тобто це просто домовленість між клієнтською

частиною, та серверною в передаванні інформації. Чисте REST api використовує CSR(Client-side rendering). Для цього потрібно мати аж 2 сервери – сервер клієнтської частини(frontend), та сервер, де відбуваються логічні обрахунки(backend). Перевага цього способу в тому, що api backend серверу можна використовувати не тільки для веб-застосунків, а й для мобільних теж. А також цей підхід дозволяє гарно розподілити роботу між командами, які пишуть клієнтську частину застосунку, та командами, що пишуть серверну. Такий підхід добре працює, коли людей, які пишуть веб-застосунок двоє, або більше. В моєму випадку було логічніше обрати SSR, бо для цього треба всього один сервер.

Для того, щоб вказати endpoint(точку входу в програму) треба визначити функцію, і обернути її в декоратор.

```
@app.route("/home")
def home():
    pages = Page.get_all_pages()
    return render_template("home.html", pages=pages)
```

Ця функція може вертати рядок, JSON, або HTML файл. В даному випадку вертається HTML файл, який динамічно відмальовується використовуючи шаблонізатор Jinja2.

Jinja2 – це шаблонізатор створений для мови Python.

Він дозволяє зручно відмальовувати дані, які приходять з бази даних. З плюсів можна виділити те, що шаблон відмальовується в пісочниці, і якщо з кодом щось буде не те, то сервер не зламається. Через це веб-додаток повністю захищений від XSS-атак.

XSS-атака – це вставлення JavaScript коду в програму, задля зловмисного виконання. Зловмисник вчинивши цю атаку може красти кукі користувачів, в яких може знаходитись сесія до веб-додатку.

З функціоналу Jinja2 можна виділити спадковість шаблонів.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jstree/3.2.1/themes/default/style.min.css"/>
    {% block head %} {% endblock %}
</head>
<body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jstree/3.2.1/jstree.min.js"></script>
{% block body %} {% endblock %}
</body>
</html>
```

Рисунок 5. Базовий шаблон

Це базовий шаблон від якого наслідуються інші шаблони. Це зручно, бо можна підключати, і перепідключати бібліотеки в одному місці, і не бігати по всіх шаблонах.

```html
{% extends 'base.html' %}

{% block head %}
    <link rel="stylesheet" href="{{ url_for('static', filename='css/home.css') }}">
{% endblock %}

{% block body %}
    <h1>Pages</h1>
    <div class="pages" id="pages">
        {% for page in pages -%}
            <div id="page{{ page.id }}" class="page{{ page.id }}">
                <a href="/editor/{{ page.id }}" id="name-ref{{ page.id }}">{{ page.name }}</a>
                <label for="rename-input{{ page.id }}">New name</label>
                <input type="text" id="rename-input{{ page.id }}">
                <button onclick="renamePage({{ page.id }})">Save</button>
                <button onclick="deletePage({{ page.id }}, '{{ page.name }}')">Delete</button>
            </div>
        {% endfor %}
    </div>
    <div class="add-new-page-container">
        <label for="add-input">Name</label>
        <input type="text" id="add-input">
        <button onclick="addPage()">Add</button>
    </div>
    <script src="{{ url_for('static', filename='js/home.js') }}"></script>
{% endblock %}
```

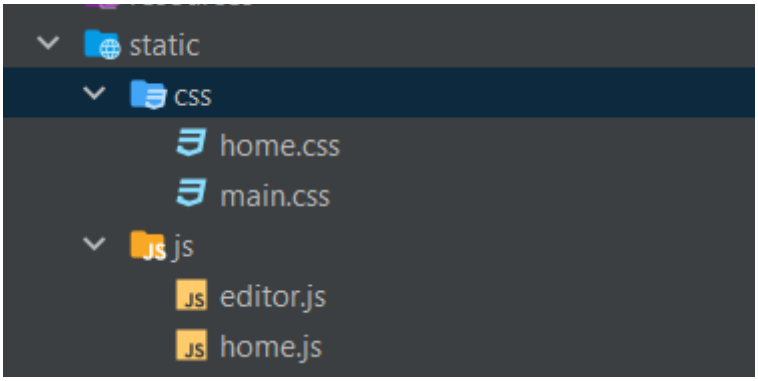
Рисунок 6. Домашня сторінка унаслідована від базової

Jinja2 може вимальовувати на сторінках як й окремі змінні, так і списки об’єктів. Для того, щоб ці змінні відрендерились достатньо просто передати їх в

параметрі метода render_template().

```
@app.route("/editor/<int:_id>")
def editor(_id):
    page = Page.get_page(_id)
    return render_template("editor.html", id=_id, name=page.name, variables=page.get_variables())
```

Всі шаблони зберігаються в папці templates. Все інше – JavaScript та CSS код зберігаються в папці static, що говорить фреймворку, про те, що ці файли не будуть змінюватись під час роботи програми.



В даній курсовій роботі використовується три типи функцій для взаємодії з клієнтською частиною:

1) Які вертають HTML файл клієнту
2) Які вертають дані в JSON форматі на клієнтську частину
3) Які змінюють дані в базі даних

Якщо про перший тип було сказано вище, він просто вертає відмальований HTML файл. То два інших не були розібрані.

Другий тип займає досить важливе місце в програмі, без нього просто не відправлялися би дані на клієнтську частину.

Через те, що ми працюємо з HTML-конструктором, то нам приходиться використовувати деревоподібну структуру даних, але через використання реляційної бази даних треба використовувати алгоритм перетворення зі списку всіх елементів, до деревоподібної структури. Для цього прийдеться використати алгоритм із циклом та рекурсією одночасно.

```
def _json_for_tree(self):
    element = Element.get_element(self.element_id)
    children = []
    if self.children_element is not None:
        for child in self.children_element.all():
            self.set_of_nodes_with_parents.add(child.id)
            child_text = child._json_for_tree()
            children.append(child_text)
        return {
            "id": self.id,
            "text": element.name,
            "children": children
        }
    return {
        "id": self.id,
        "text": element.name
    }
```

Алгоритм будує деревоподібну структуру. Спочатку об’єкт на якому викликається метод – це об’єкт кореневого елементу, і потім цикл проходиться по всіх об’єктах елементів, і викликає у всіх цей самий метод.

Для того, щоб правильно відмальовувались змінні, і щоб зберігалась можливість їх редагування на клієнтській частині, було вирішено додати ще один метод, який будує JSON представлення елементу.

```
def json_data(self):
    element = Element.get_element(self.element_id)

    return {
        "id": self.id,
        "name": element.name,
        "html": element.html,
        "css": element.css,
        "parent_id": self.parent_element_id
    }
```

Рисунок 7.Невідмальоване представлення

```python
def json_with_converted_html(self):
    element = Element.get_element(self.element_id)
    for variable in self.element_on_page_variable.all():
        element.html = element.html.replace('{' + variable.name + '}', variable.value)

    return {
        "id": self.id,
        "name": element.name,
        "html": element.html,
        "css": element.css,
        "parent_id": self.parent_element_id
    }
```

Рисунок 8. Відмальоване представлення

і метод вертає дані для редагування на клієнтській частині.

Другий метод рендерить(відмальовує) готовий HTML код, який буде відображатись у користувача як результат. Це відбувається за рахунок заміни змінних їх значеннями.

Висновки

Було проаналізовано готові рішення в побудуванні HTML-сторінок. У цій роботі було використано фреймворк Flask та бібліотеку SQLAlchemy в якості ORM. Було детально розглянуто принцип роботи сервера цього веб-додатку. Було досліджено методи динамічної побудови HTML-сторінок. Написано застосунок, який динамічно будує HTML-сторінки.

Список використаних джерел

1. Документація по JQuery - https://api.jquery.com
2. Документація по JSTree - https://www.jstree.com/docs
3. Документація по Flask - https://flask.palletsprojects.com/en/2.1.x
4. Документація по SQLAlchemy - https://docs.sqlalchemy.org/en/14/orm
5. Lean D. Everything you need to know about tree data structures [Електронний ресурс]/ Режим доступу до ресурсу: https://www.freecodecamp.org/news/all-you-need-to-know-about-tree-data-structures-bceacb85490c