МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка додатку для управління задачами та організації робочого часу»

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи ЦДБ-20-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Технології цифрового дизайну»

_____ Канюка Владислав Богданович
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Ясенова Ірина Сергіївна, к.т.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Ратушняк Тетяна Володимирівна, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «22» травня 2024 №15

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2024

Реєстрація № 38 31.05.2024 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 22.05.2024 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 11.06.2024 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії:

_____ М.М. Філоненко
(підпис)

_____ С.М. Грищенко
(підпис)

_____ В.О. Ніжегородцев
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	26		11.06.2024 Франчук В.М. Філоненко М.М. Грищенко С.М. Ніжегородцев В.О. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	7		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	3		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	41		

Оцінювання кваліфікаційної роботи науковим керівником - 39 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
80	B	4 (добре)	11.06.2024	Франчук В.М. Філоненко М.М. Грищенко С.М. Ніжегородцев В.О. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Технології цифрового дизайну»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«23» жовтня 2023

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Канюка Владислав Богданович
(прізвище, ім'я, по батькові)

Тема кваліфікаційної роботи «Розробка додатку для управління задачами та організації робочого часу»

1. Затверджена наказом ректора від «20» жовтня 2023 № 1411
2. Строк здачі закінченої роботи «10» травня 2024
3. Мета роботи розробка програмного забезпечення для управління часом, яке дозволить користувачам ефективно планувати свій робочий та особистий час, визначати пріоритетні завдання, ведення щоденних списків справ, а також аналізувати витрачений час на виконання конкретних завдань. Об'єкт дослідження
   Предмет дослідження сучасні види додатків для створення власного додатку для управління задачами та організації робочого часу
   Орієнтовна база дослідження
   Власій О. О., Винничук М. Д. Розробка мобільних додатків засобами блочного програмування: Навчально-методичний посібник. Прикарпатський національний університет імені Василя Стефаника, 2021. 130 с.
4. Вихідні дані до роботи інформація про існуючі додатки для управління задачами та планування робочого часу.
5. Перелік графічного матеріалу діаграми, малюнки, таблички
6. Зміст кваліфікаційної роботи вступ, розділ 1 аналіз предметної області, 1.1 поняття тайм-менеджменту, 1.2 розробка програмного забезпечення, 1.3 паттерн MVC, розділ 2 методологія дослідження, 2.1 інструментарій розробки, 2.2 проектування функціоналу системи, розділ 3 розробка програми і аналіз результатів, 3.1 розробка внутрішньої будови, 3.2 розробка інтерфейсу користувача.

7. Календарний план виконання роботи

№ п/п	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.10.2023	До 15.10.2023
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	29.12.2023	29.12.2023
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	16.02.2024	16.02.2024
	ІІІ розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2024	16.04.2024
	Висновки	30.04.2024	30.04.2024
	Доопрацювання роботи з урахуванням зауважень наукового керівника	10.05.2024	10.05.2024
	Отримання відгуку керівника	30.05.2024	30.05.2024
	Отримання зовнішньої рецензії	03.06.2024	03.06.2024

8. Дата видачі завдання «23» жовтня 2023

9. Керівник випускної кваліфікаційної роботи _____ (підпис) Ясенова (прізвище, і

10. Керівник освітньо-професійної програми _____ (підпис) Ратушн (прізвище, і

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Канюка (прізвище, і

АНОТАЦІЯ

Мета роботи полягає в тому щоб розробити додаток для управління завданнями та організаціями робочого часу, який буде зручним у використанні, функціональним та ефективним.

Завданням роботи є проаналізувати різні існуючі методи та інструменти для управління та плануванням робочого часу. Визначити потреби користувачів до додатку для управління завданнями та організації робочого часу. Реалізувати та провести тестування додатку також оцінити ефективність роботи додатку.

Об’єктом дослідження є такі відомі інструменти як Todoist, Microsoft To Do, Any.do.

Одержані результати це розроблений додаток, який буде зручним у використанні та ефективним також підвищення продуктивності користувачів цього додатку. Позитивні відгуки користувачів про додаток.

Кваліфікаційна робота містить 66 сторінок, 1 таблицю, 5 рисунків, 40 використаних джерел

Ключові слова: Pyhton, CSS, HTML

Рік виконання кваліфікаційної роботи 2024

Рік захисту роботи 2024

ABSTRACT.

The aim of the work is to develop an application for managing tasks and organising working time that is easy to use, functional and efficient.

The task is to analyse various existing methods and tools for managing and scheduling working time. Identify user needs for a task management and time management application. Implement and test the application and evaluate the effectiveness of the application.

The object of the study is such well-known tools as Todoist, Microsoft To Do, Any.do.

The results obtained are a developed application that will be easy to use and effective in increasing the productivity of users of this application. Positive feedback from users about the application.

The qualification work contains 66 pages, 1 table, 5 figures, 40 references.

Keywords: Python, CSS, HTML

Year of completion of the qualification work 2024

Year of defence 2024

СПИСОК УМОВНИХ СКОРОЧЕНЬ

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

MVC: Model-View-Controller

LIST: LISt Processing language

CWI: Centrum Wiskunde & Informatica

SGML: Standard Generalized Markup Language

MIME: Multipurpose Internet Mail Extensions

GUI: Graphical User Interface

CERN: European Council for Nuclear Research

ATM: automated teller machine

ABC: American Broadcasting Company

Cascade Framework: a framework created to facilitate the work of the web developer and web designer, speed up development and prevent the maximum number of errors.

PHP: Hypertext Preprocessor

ЗМІСТ

ВСТУП 10

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11

1.1 Поняття тайм-менеджменту 11

1.2 Розробка програмного забезпечення 13

1.3 Паттерн MVC 15

1.4 Огляд існуючих рішень 18

1.5 Постановка задачі 20

РОЗДІЛ 2 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ 21

2.1 Інструментарій розробки 21

2.2 Проектування функціоналу системи 49

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМИ І АНАЛІЗ РЕЗУЛЬТАТІВ 51

3.1 Розробка внутрішньої будови 51

3.2 Розробка інтерфейсу користувача 53

3.3 Тестування 58

ВИСНОВКИ 62

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ 67

ВСТУП

У сучасному ритмі життя ефективне управління часом стає ключовим аспектом успіху в різних сферах діяльності. Однак, зі зростанням обсягу завдань та розмаїттям діяльностей, управління часом може стати викликом для багатьох людей. Для вирішення цієї проблеми розробляються програмні засоби, що допомагають у плануванні та контролі за власним часом.

Мета даного дипломного проекту полягає у розробці програмного забезпечення для управління часом, яке дозволить користувачам ефективно планувати свій робочий та особистий час, визначати пріоритетні завдання, ведення щоденних списків справ, а також аналізувати витрачений час на виконання конкретних завдань. Програмний продукт буде містити функціонал для реєстрації завдань, встановлення термінів виконання, нагадувань про них та підсумкового аналізу витраченого часу.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Розробка інтерфейсу для зручного ведення списків справ та планування часу.
2. Реалізація функціоналу для встановлення пріоритетів та термінів виконання завдань.
3. Розробка механізму нагадувань та планування подій у календарі.
4. Реалізація засобів аналізу витраченого часу та відслідковування прогресу у виконанні завдань.

Цей дипломний проект спрямований на створення зручного та ефективного інструмента для управління часом, який допоможе користувачам більш ефективно розподіляти свій час та досягати поставлених цілей.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття тайм-менеджменту

Тайм-менеджмент — це процес планування та свідомого контролю часу, витраченого на конкретну діяльність, особливо для підвищення ефективності та продуктивності.

Управління часом передбачає вимоги, пов’язані з роботою, соціальним життям, сім’єю, хобі, особистими інтересами та зобов’язаннями. Ефективне використання часу дає людям більше можливостей для управління діяльністю.[1] Тайм-менеджменту може допомогти низка навичок, інструментів і методів, особливо під час виконання конкретних завдань, проектів і цілей із дотриманням терміну виконання.

Спочатку термін «тайм-менеджмент» охоплював лише ділову та робочу діяльність, але з часом цей термін розширився, включивши також особисту діяльність. Система управління часом – це розроблена комбінація процесів, інструментів, технік і методів. Управління часом зазвичай є необхідністю в управлінні будь-яким проектом, оскільки воно визначає час завершення проекту та обсяг.

Культурологічні погляди на тайм-менеджмент

Різниця в тому, як культура бачить час, може вплинути на спосіб управління часом. Наприклад, лінійне представлення часу — це спосіб уявлення про те, що час перетікає від одного моменту до наступного лінійним чином. Таке лінійне сприйняття часу є домінуючим в Америці разом з більшістю країн Північної Європи, таких як Німеччина, Швейцарія та Англія.[2] Люди в цих культурах, як правило, надають великого значення продуктивному управлінню часом і схильні уникати рішень або дій, які призвели б до марної втрати часу.[2]

Цей лінійний погляд на час корелює з тим, що ці культури є більш «монохронними» або вважають за краще робити лише одну справу за раз.

Іншим культурним переглядом часу є багатоактивний перегляд часу. У мультиактивних культурах більшість людей вважають, що чим більше справ або завдань виконується одночасно, тим краще. Це створює відчуття щастя.[2] Багатоактивні культури є «поліхронними» або вважають за краще виконувати кілька завдань одночасно. Цей багатоактивний режим перегляду часу поширений у більшості країн Південної Європи, таких як Іспанія, Португалія та Італія.[2] У цих культурах люди часто схильні витрачати час на те, що вони вважають більш важливими, наприклад, приділяти велике значення завершенню соціальних розмов.[2] У бізнес-середовищі вони часто звертають мало уваги на тривалість зустрічей, радше зосереджуються на проведенні зустрічей високої якості. Загалом культурна увага зосереджена на синергії та творчості, а не на ефективності.[3]

Кінцевий культурний час перегляду є циклічним переглядом часу. У циклічних культурах час не вважається ані лінійним, ані подієвим. Оскільки дні, місяці, роки, пори року та події відбуваються регулярно, повторювано, час розглядається як циклічний. З цієї точки зору час не вважається витраченим даремно, оскільки він завжди повернеться пізніше, отже, його кількість необмежена.[2] Цей циклічний погляд на час поширений у більшості країн Азії, включаючи Японію та Китай. У культурах із циклічними концепціями часу важливіше зосередитися на правильному виконанні завдань, тому більшість людей витрачатимуть більше часу на роздуми про рішення та вплив, який вони матимуть, перш ніж діяти за своїми планами.[3] Більшість людей у циклічних культурах схильні розуміти, що інші культури мають різні точки зору на час, і усвідомлюють це, діючи на глобальній арені.[4]

Нейропсихологія

Надмірна та хронічна нездатність ефективно керувати часом може бути наслідком синдрому дефіциту уваги та гіперактивності (СДУГ).[5] Діагностичні критерії включають відчуття недостатньої успішності, труднощі з організацією, проблеми з початком роботи, проблеми з керуванням багатьма одночасними проектами та проблеми з продовженням.[6]

1.2 Розробка програмного забезпечення

Розробка програмного забезпечення — це процес, який використовується для розробки, визначення, проектування, програмування, документування, тестування та виправлення помилок з метою створення та підтримки додатків, фреймворків або інших компонентів програмного забезпечення. Розробка програмного забезпечення передбачає написання та підтримку вихідного коду, але в ширшому сенсі вона включає всі процеси від концепції бажаного програмного забезпечення до остаточного прояву, як правило, у спланованому та структурованому процесі, який часто збігається з розробкою програмного забезпечення. Розробка програмного забезпечення також включає дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, переробку, технічне обслуговування або будь-яку іншу діяльність, яка призводить до створення програмних продуктів [1].

Методики

Одна методологія розробки системи не обов’язково підходить для використання в усіх проектах.

Кожна з доступних методологій найкраще підходить для конкретних видів проектів, виходячи з різних технічних, організаційних, проектних і командних міркувань [2].

Діяльності

Виявлення потреби

Джерел ідей для програмних продуктів досить багато. Ці ідеї можуть виникнути в результаті дослідження ринку, включаючи демографічні дані потенційних нових клієнтів, існуючих клієнтів, потенційних клієнтів, які відмовилися від продукту, іншого внутрішнього персоналу з розробки програмного забезпечення або креативної третьої сторони. Ідеї для програмних продуктів, як правило, спочатку оцінюються маркетинговим персоналом на предмет економічної доцільності, відповідності існуючим каналам розповсюдження, можливого впливу на існуючі лінійки продуктів, необхідних функцій і відповідності маркетинговим цілям компанії. На етапі маркетингової оцінки оцінюються припущення щодо витрат і часу. На початку першої фази приймається рішення про те, чи слід продовжувати проект на основі більш детальної інформації, отриманої спеціалістами з маркетингу та розробки [3].

Процес планування

Важливим завданням при створенні ПЗ є аналіз вимог. Клієнти зазвичай мають абстрактне уявлення про те, що вони хочуть як кінцевий результат, але не знають, що програмне забезпечення повинно робити. Кваліфіковані та досвідчені інженери програмного забезпечення визнають неповні, неоднозначні або навіть суперечливі вимоги на цьому етапі. Часта демонстрація коду в реальному часі може допомогти зменшити ризик того, що вимоги будуть неправильними.

«Хоча на етапі розробки вимог докладається багато зусиль, щоб гарантувати, що вимоги є повними та узгодженими, рідко це трапляється; залишаючи етап проектування програмного забезпечення найвпливовішим, коли йдеться про мінімізацію впливу нових або змінених вимог. Нестабільність вимог є складним, оскільки вони впливають на майбутні або вже реалізовані зусилля» [5].

Робоча сила

Розробник програмного забезпечення — це особа або компанія, яка займається процесом розробки програмного забезпечення, включаючи дослідження, проектування, програмування, тестування та інші аспекти створення комп’ютерного програмного забезпечення. Інші назви посад для людей із подібним значенням включають програміста, програмного аналітика або програмного інженера. Компанії, що спеціалізуються на програмному забезпеченні, можна назвати будинками програмного забезпечення. У великій компанії можуть бути співробітники, одноосібна відповідальність яких складається тільки з однієї з дисциплін. У невеликих середовищах розробки кілька людей або одна людина можуть керувати повним процесом. Середовища спільної роботи, такі як програмне забезпечення з відкритим кодом, можуть об’єднати багато розробників.

Переглянути модель

Модель представлення — це структура, яка надає точки зору на систему та її середовище, які будуть використовуватися в процесі розробки програмного забезпечення. Це графічне представлення основної семантики перегляду.

Мета точок зору та поглядів полягає в тому, щоб дозволити інженерам-людинам зрозуміти дуже складні системи та організувати елементи проблеми навколо областей знань. У розробці фізично інтенсивних систем точки зору часто відповідають можливостям і обов'язкам в інженерній організації.[6]

1.3 Паттерн MVC

Model-View-Controller (зазвичай відомий як MVC) - це шаблон дизайну програмного забезпечення [1], який зазвичай використовується для розробки користувальницьких інтерфейсів, що розділяє відповідну логіку програми на три взаємопов'язані елементи. Це робиться для відокремлення внутрішнього

подання інформації від способів подання та прийняття інформації користувачем. [2] [3]

Традиційно використовуваний для графічних користувальницьких інтерфейсів (GUI), цей шаблон став популярним для проектування веб-додатків. [4] Популярні мови програмування, такі як JavaScript, Python, Perl, Object Pascal / Delphi, Ruby, PHP, Java, C #, Swift та Elixir, мають фреймворки MVC, які використовуються для розробки веб- чи мобільних додатків прямо зараз.

Модель

Центральний компонент візерунка. Це динамічна структура даних програми, незалежна від інтерфейсу користувача. [5] Він безпосередньо управляє даними, логікою та правилами програми.

Представлення

Будь-яке подання такої інформації, як діаграма або таблиця. Можливі декілька подань однієї і тієї ж інформації, наприклад, гістограма для управління та таблична таблиця для бухгалтерів.

Контролер

Приймає введення та перетворює його в команди для моделі чи подання. [6] На додаток до розподілу програми на ці компоненти, конструкція модель-вигляд-контролер визначає взаємодію між ними. [7]

- Модель відповідає за управління даними програми. Він отримує вхід користувача від контролера.
- Представлення робить презентацію моделі в певному форматі.
- Контролер реагує на введення користувачем і виконує взаємодію з об'єктами моделі даних. Контролер отримує вхідні дані, додатково перевіряє їх, а потім передає вхідні дані моделі.

Як і у випадку з іншими програмними моделями, MVC виражає "суть рішення" проблеми, дозволяючи її адаптувати для кожної системи. [8]

Конкретні конструкції MVC можуть суттєво відрізнятися від традиційного опису тут. [9]

Одним із основних уявлень про ранню розробку графічних користувальницьких інтерфейсів, MVC став одним із перших підходів до опису та реалізації програмних конструкцій з точки зору їх відповідальності. [10]

Тригве Ренскауг представив MVC в Smalltalk-79 під час відвідування дослідницького центру Xerox Palo Alto (PARC) [11] [12] у 1970-х. У 1980-х роках Джим Алтофф та інші впровадили версію MVC для бібліотеки класу Smalltalk-80. Лише пізніше стаття 1988 року в "Журналі об'єктних технологій" (JOT) висловила MVC як загальну концепцію. [13]

Згодом шаблон MVC еволюціонував, [14] породивши такі варіанти, як ієрархічна модель-вигляд-контролер (HMVC), модель-вигляд-адаптер (MVA), модель-вигляд-презентатор (MVP), модель-вигляд-виглядмодель (MVVM) та інші, що адаптували MVC до різних контекстів.

Використання шаблону MVC у веб-додатках зростало популярністю після введення в 1996 році WebObjects NeXT, який спочатку був написаний в Objective-C (який значною мірою запозичений у Smalltalk) і допоміг забезпечити застосування принципів MVC. Пізніше шаблон MVC став популярним серед розробників Java, коли WebObjects було перенесено на Java. Пізніші рамки для Java, такі як Spring (випущений у жовтні 2002 р.), Продовжували міцний зв’язок між Java та MVC. Введення фреймворків Django (липень 2005 р. Для Python) та Rails (грудень 2005 р. Ruby), обидва з яких робили сильний акцент на швидкому розгортанні, збільшив популярність MVC поза традиційним корпоративним середовищем, в якому він давно популярний. Мережеві рамки MVC тепер мають великі частки ринку порівняно з веб-наборами, що не належать до MVC.

Хоча спочатку він був розроблений для настільних обчислень, MVC широко застосовувався як дизайн додатків World Wide Web на основних мовах програмування. Було створено кілька веб-фреймворків, які забезпечують застосування шаблону. Ці основи програмного забезпечення різняться у своїх інтерпретаціях, головним чином у тому, як обов'язки MVC розподіляються між клієнтом та сервером [15].

Деякі веб-фреймворки MVC застосовують тонкий клієнтський підхід, який розміщує майже всю модель, вигляд та логіку контролера на сервері. При такому підході клієнт надсилає запити гіперпосилання або подання форми контролеру, а потім отримує повну та оновлену веб-сторінку (або інший документ) із подання; модель повністю існує на сервері. [15]

1.4 Огляд існуючих рішень

Todoist - це популярний інструмент для управління завданнями та списками справ. Він дозволяє користувачам створювати завдання, призначати їм терміни виконання, встановлювати пріоритети та категорії, а також спільно працювати над завданнями в команді. Todoist доступний для різних платформ, включаючи веб-версію, мобільні додатки та десктопні застосунки.

Переваги:

Інтуїтивний і простий у використанні інтерфейс.

Можливість призначати терміни виконання, встановлювати пріоритети та категорії для завдань.

Спільна робота над завданнями в команді.

Доступний для різних платформ.

Недоліки:

Деякі функції доступні лише у платній версії.

Обмеження на кількість одночасних проєктів у безкоштовній версії.

Microsoft To Do - це ще один популярний інструмент для управління завданнями від Microsoft. Він інтегрований з іншими сервісами Microsoft, такими як Outlook та Microsoft 365, що дозволяє легко організовувати завдання та спільно працювати над ними з колегами. Microsoft To Do також доступний на різних платформах.

Переваги:

Інтеграція з іншими сервісами Microsoft, що полегшує організацію завдань.

Спільна робота над завданнями в команді.

Доступний на різних платформах.

Недоліки:

Деякі функції можуть бути обмежені у безкоштовній версії.

Інтерфейс може бути складнішим порівняно з іншими інструментами.

Any.do - це інший популярний інструмент для управління завданнями та організації робочого часу. Він пропонує широкий спектр функцій, таких як створення завдань, нагадування про них, організація списків справ та багато іншого. Any.do також має можливість спільної роботи над завданнями в команді.

Переваги:

Широкий функціонал для управління завданнями та організації робочого часу.

Можливість спільної роботи над завданнями в команді.

Інтуїтивний і простий у використанні інтерфейс.

Недоліки:

Обмежені можливості у безкоштовній версії.

Деякі користувачі відзначають відсутність деяких додаткових функцій, які є у конкурентів.

1.5 Постановка задачі

Мета проекту: Створення веб-додатка, який дозволить користувачам ефективно управляти своїми завданнями, створювати списки справ, встановлювати пріоритети та терміни виконання, а також спільно працювати над завданнями в команді.

Функціонал програми:

1. Можливість створювати, редагувати та видаляти завдання.
2. Створення списків справ для організації завдань за категоріями або проєктами.
3. Встановлення пріоритетів та термінів виконання для кожного завдання.
4. Нагадування про наближення термінів виконання завдань.
5. Можливість позначати завдання як виконані.

Технічні вимоги:

1. Додаток повинен бути реалізований як веб-додаток із використанням фреймворку Flask.
2. Взаємодія з базою даних для збереження інформації про користувачів, завдання та списки справ.
3. Використання HTML, CSS та JavaScript для створення інтерфейсу користувача.
4. Забезпечення безпеки даних та захисту від несанкціонованого доступу.
5. Можливість розширення функціоналу за необхідності.

Загальна ідея проекту полягає у створенні простого та зручного інструменту для управління завданнями та організації робочого часу, який дозволить користувачам бути більш продуктивними та організованими.

РОЗДІЛ 2
МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

2.1 Інструментарій розробки

Python - інтерпретована мова програмування загального призначення високого рівня. Філософія дизайну Python підкреслює читабельність коду, використовуючи значні відступи. Його мовна структура та об’єктно-орієнтований підхід покликані допомогти програмістам писати чіткі логічні коди для великих та великих проектів.

Python динамічно збирає сміття. Він підтримує кілька парадигм програмування, включаючи структуроване (включаючи процедурне), об'єктно-орієнтоване та функціональне програмування. Оскільки Python має повну стандартну бібліотеку, її зазвичай описують як мову, що включає батареї.

Як спадкоємець мови програмування ABC, Гідо ван Россум почав займатись дослідженнями Python наприкінці 1980-х і вперше випустив цю мову під назвою Python 0.9.0 в 1991 році. [32] Python 2.0 був випущений у 2000 році, представивши нові функції, такі як розуміння списку та система збору сміття за допомогою підрахунку посилань, і був припинений у версії 2.7.18 у 2020 році. [33] Python 3.0 був випущений в 2008 році і є основною редакцією мови. Мова не повністю сумісна із зворотним характером, і більшість кодів Python 2 не можуть нормально працювати на Python 3.

Python завжди був однією з найпопулярніших мов програмування.

Python був задуманий наприкінці 1980-х років Гвідо ван Россумом з Нідерландських Centrum Wiskunde & Informatica (CWI). Він є наступником мови програмування ABC, натхненної SETC, яка може обробляти винятки та взаємодіяти з операційною системою Amoeba. Його реалізація розпочалась у грудні 1989 року. Як головний розробник проекту, Ван Россум брав на себе

повну відповідальність за проект до 12 липня 2018 року, коли оголосив про своє "вічне свято" титулом "дружнього диктатора життя" Пітона. Він знайшов своє відображення у своїй довгій історії спільнотою Python. Зобов'язання бути головним менеджером проекту. Зараз він займає керівну посаду в якості члена ради наглядових органів із п'яти осіб. У січні 2019 року активні розробники ядра Python обрали Бретта Кеннона, Ніка Коглана, Баррі Варшаву, Керол Віллінг та Ван Россума членами комітету з п'яти осіб. З тих пір Гвідо ван Россум відкликав свою кандидатуру до ради 2020 року.

Python 2.0 був випущений 16 жовтня 2000 року з багатьма основними новими можливостями, включаючи збирач сміття для виявлення циклів та підтримку Unicode.

Python 3.0 був випущений 3 грудня 2008 року. Це серйозний перегляд мови, який не повністю сумісний з іншою стороною. Багато його основних функцій було перенесено до версій Python 2.6.x та 2.7.x. До розподілу Python 3 входить утиліта 2to3, яка автоматично (принаймні частково) перетворює код Python 2 у Python 3.

Термін дії Python 2.7 спочатку був встановлений у 2015 році, а потім перенесений на 2020 рік через побоювання, що велика кількість існуючого коду не може бути легко перенесена на Python 3. Більше виправлень безпеки та інших удосконалень не буде випущено. Наприкінці життєвого циклу Python 2 підтримує лише Python 3.6.x та новіші версії.

Оскільки всі версії Python мають проблеми з безпекою, швидкість Python 3.9.2 та 3.8.8 була прискорена, що призвело до можливого віддаленого виконання коду та отруєння веб-кешу.

Python - мова програмування з багатьма парадигмами. Він повністю підтримує об'єктно-орієнтоване програмування та структуроване програмування, а багато його функцій підтримують функціональне

програмування та аспектно-орієнтоване програмування (включаючи дизайн метапрограми та мета-об'єкт (магічний метод)). Розширення підтримує багато інших парадигм, включаючи дизайн контрактів та логічне програмування.

Python використовує динамічне введення тексту та комбінацію підрахунку посилань та збирачів сміття.Сбирач сміття визначає цикл для управління пам'яттю. Він також має динамічну роздільну здатність імен (пізнє прив'язування), щоб асоціювати імена методів та змінних під час виконання програми.

Python призначений для надання певної підтримки функціонального програмування на основі Lisp. За допомогою функцій фільтрування, відображення та скорочення; перелічено розуміння генераторів, словників, наборів та виразів. Стандартна бібліотека має два модулі (itertools та functools) для реалізації функціональних інструментів, запозичених у Haskell та Standard ML.

Основні принципи мови викладені в документі Zen of Python (PEP 20), який включає такі сентенції:

- Красиво-краще, ніж потворно.
- Явне краще, ніж неявне.
- Просте - краще, ніж складне.
- Комплекс краще, ніж комплекс.
- Розрахунок читабельності.

Python не має всіх своїх функцій, вбудованих у своє ядро, але розроблений таким чином, щоб бути дуже розширюваним (з модулями). Ця компактна модульність робить його особливо популярним як спосіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван Россума щодо малих основних мов, великих стандартних бібліотек та легко розширюваних перекладачів випливає з його невдоволення ABC, який підтримує протилежний

підхід. Python прагне спростити та зменшити плутанину граматики та граматики, дозволяючи розробникам обирати методи кодування. На відміну від девізу Perl "Існує кілька способів зробити це", Python висвітлює філософію дизайну "Для цього повинен бути один - бажано, лише один очевидний спосіб". [69] Алекс Мартеллі, співробітник Фонду програмного забезпечення Python і автор книги про Python, писав: "Опис чогось як" розумного "не вважається компліментом до культури Python".

Розробники Python прагнуть уникнути передчасної оптимізації та відмовляються виправляти некритичні частини посилальної реалізації CPython. Ці виправлення відбуваються за рахунок ясності, тим самим трохи покращуючи швидкість. Коли швидкість важлива, програмісти Python можуть переміщати критично важливі для часу функції до розширень, написаних мовами, такими як C, або використовувати компілятор PyPy, що встигає за часом. Також доступний Cython, який перетворює сценарії Python на C і здійснює виклики API рівня C безпосередньо до інтерпретатора Python.

Важливою метою розробників Python є збереження задоволення від використання. Це відображається в назві мови (данина поваги британській комедійній компанії Monty Python), а іноді і в цікавих підручниках та довідкових методах, таких як стандартний foo та bars.

Користувачів та шанувальників Python, особливо тих, хто вважається обізнаним або досвідченим, часто називають Pythonistas. Python має бути простою для читання мовою. Його формат візуально не бентежить, він часто використовує англійські ключові слова, тоді як інші мови використовують розділові знаки. На відміну від багатьох інших мов, він не використовує фігурні дужки для розділення блоків і допускає крапку з комою після операторів, але крапка з комою використовується рідко. Він має менше граматичних винятків та особливих випадків, ніж C або Pascal.

Python використовує пробіли замість фігурних дужок або ключових слів для розділення блоків. Збільшення відступу відбувається після певних операторів; зменшення відступу означає кінець поточного блоку. Тому візуальна структура програми точно відображає семантичну структуру програми. Цю особливість іноді називають `` зовнішнім " правилом, і деякі інші мови також мають це правило, але в більшості мов відступ не має семантичного значення. Рекомендований розмір відступу - чотири пробіли.

Оператори Python включають (серед іншого):

- Використовуйте знак рівності = оператор присвоєння.
- Оператор if, який використовується разом з else та elif (скорочення від else-if) для умовного виконання блоку коду.
- Оператор for, який фіксує кожен елемент у локальній змінній для використання вкладеним блоком для перегляду об'єкта, що ітерацію.
- Поки оператор, доки умова є істинним, виконується блок коду.
- Оператор try, який дозволяє захоплювати та обробляти витяги, що містяться у вкладених блоках коду (крім речень); він також гарантує, що незалежно від того, як блок виходить, код очищення в блоці завжди буде виконаний в кінцевому підсумку.
- Оператор підвищення застосовується для отримання зазначеного винятку або повторного підняття спійманого винятку.
- Оператори класів, які виконують блоки коду та приєднують свій локальний простір імен до класу для використання в об'єктно-орієнтованому програмуванні.
- Оператор Def, який визначає функцію або метод.
- Оператор with з Python 2.5, випущений у вересні 2006 р. [82], який охоплює блок коду в диспетчері контексту (наприклад, отримання блокування перед запуском кодового блоку, а потім звільнення блокування або відкриття

файлу. Потім закрийте його), дозволяють поведінку подібних ресурсів ініціалізувати (RAII) та замінюють загальне використання try / final.

- Оператор Break, вихід із циклу.
- Оператор continue пропускає цю ітерацію та переходить до наступного пункту.
- Оператор del видалить змінну, що означає, що посилання на ім'я значення буде видалено, а спроби використання цієї змінної призведуть до помилки. Ви можете перепризначити видалені змінні.
- Виконувати функції NOP через операторів. Синтаксично це необхідно для створення порожнього блоку коду.
- Заява про затвердження, що використовується для перевірки умов, що застосовуються під час налагодження.
- Оператор Yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану.
- Оператор return використовується для повернення значень із функцій.
- Оператор import використовується для імпорту модулів, функції чи змінні яких можна використовувати в поточній програмі. Існує три способи використання імпорту: імпорт <ім'я модуля> [як <псевдонім>] або <ім'я модуля> імпорт * або <ім'я модуля> для імпорту <визначення 1> [як <псевдонім 1>], <визначення 2 > [Як <псевдонім 2>], ...

Роль оператора присвоєння (=) полягає у прив'язуванні імені як посилання на окремий динамічно розподілений об'єкт. Потім ви можете будь-коли відхилити змінну до будь-якого об’єкта. У Python ім'я змінної є спільним власником посилання, і з ним не пов’язаний фіксований тип даних. Однак у цей час змінна посилається на об’єкт із типом. Це називається динамічним набором

тексту і на відміну від статично набраних мов програмування, де кожна змінна може містити лише певний тип значення.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і за словами Гвідо ван Россума, це ніколи не буде. Однак краща підтримка функцій, подібних до корутину, передбачена в 2.5, розширюючи тим самим генератор Python. Ледачий ітератор має максимум 2,5 генератора. Інформація передається в одному напрямку від генератора. За допомогою Python 2.5 ви можете передавати інформацію назад до функції генератора, тоді як за допомогою Python 3.3 ви можете передавати інформацію через кілька рівнів стека.

Деякі вирази Python подібні до таких, що зустрічаються в таких мовах, як C та Java, а інші - ні:

- Додавання, віднімання та множення однакові, але поведінка ділення інша. У Python є два типи розділів. Вони є базовим діленням (або цілочисельним поділом) // та плаваючою комою / діленням. Python також використовує оператори ** для просування.
- У Python 3.5 було введено нове твердження @infix. Він використовується бібліотеками, такими як NumPy, для множення матриць.
- У Python 3.8 введено синтаксис: =, який називається "моржовим оператором". Він присвоює значення змінним як частину більшого виразу. [91]
- У Python == порівнює з Java за значенням, Java порівнює числові значення за значенням, а об'єкти порівнює за посиланням. (Порівняння значень об'єктів у Java може бути здійснено за допомогою методу equals ().) Оператор is is is is може бути використаний для порівняння ідентифікаторів об'єктів (порівняння посилань). У Python можна порівняти порівняння, наприклад <= b <= c.

- Python використовує слово і, або не для своїх булевих операторів, а не для символів &&, ||! Використовується в Java та C.Python має тип виразу, що називається розумінням списку, а також більш загальний вираз, який називається генераторським виразом.
- • Анонімні функції реалізовані з використанням лямбда-виразів, однак вони обмежені тим фактом, що тіло може бути лише виразом.
- • Якщо с інакше у, умовний вираз у Python буде записаний як x (відмінний від порядку операнда оператора c? X: y), що є однаковим у багатьох інших мовах.
- • Python розрізняє списки та кортежі. Список записаний як [1, 2, 3], він був змінений і не може використовуватися як ключ словника (ключ словника повинен бути постійним у Python). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу є незмінними, він може використовуватися як ключ словника. Оператор + може використовуватися для об'єднання двох кортежів, що безпосередньо не змінює їх вміст, але створює новий кортеж, що містить елементи двох передбачених кортежів. Отже, враховуючи, що змінна t спочатку дорівнює (1,2,3), коли виконується t = t + (4, 5), спочатку обчисліть t + (4, 5), щоб отримати (1,2, 3) . 3, 4, 5), а потім призначити його назад до t, тим самим ефективно "змінюючи вміст t", одночасно відповідаючи інваріантній природі об'єкта кортежу. У чіткому контексті кортежі не потребують дужок.
- • Python має функцію декомпресії послідовності, де кілька виразів (кожен вираз обчислює весь вміст, який може бути призначений) (змінні, атрибути запису тощо) мають однакову форму з текстом, що становить кортеж, і, як правило, розміщуються зліва Одна сторона знака рівності у твердженні про присвоєння. Оператор очікує ітерабельний об’єкт праворуч від знака рівності. При сортуванні об’єкт видасть таку ж кількість значень, що й даний вираз, і

відфільтрує їх, присвоюючи кожне створене значення лівій стороні відповідного виразу.

- • Python має оператор% "рядковий формат". Це схоже на рядок printf у C, наприклад, "спам =% s яйце =% d"% ("бла", 2) оцінюється як "спам = бла-яйця = 2". У Python 3 та 2.6+ метод str () формату () доповнив це, наприклад "спам = {0} яйце = {1}". Формат ("бла", 2). Python 3.6 додав "f-рядки": blah = "blah"; яйця = 2; f'spam = {blah} яйця = {яйця}.
- • Рядки в Python можна поєднувати шляхом "додавання" (те саме твердження, що і додавання цілих чи значень з плаваючою комою). Наприклад, "спам" + "яйце" повертає "спамера". Навіть якщо ваш рядок містить цифри, вони все одно будуть додані як рядки замість цілих чисел. Наприклад, "2" + "2" повертає "22".
- • Python має різні типи рядкових літералів:
  - Рядки, укладені в одинарні або подвійні лапки. На відміну від Unix Shell, мови Perl та Perl впливають на Perl, а одинарні та подвійні лапки працюють однаково. Обидва типи рядків використовують зворотні скісні риски (\) як вихідні символи. Інтерполяція рядків стала "відформатованим рядковим літералом" у Python 3.6.Рядки з подвійними лапками, які починаються та закінчуються серією з трьох одинарних або подвійних лапок. Вони можуть охоплювати кілька рядків і функціонувати, як тут документи в оболонках, Perl і Ruby.
  - Початковий рядок, представлений префіксом рядкового літералу перед r. Ескейп-послідовності не інтерпретуються; тому необроблені рядки корисні в місцях, де поширені зворотні скісні риски літер, наприклад, регулярні вирази та шляхи у стилі Windows. Порівняйте "@ -citation" у C #.
- Python має індекси масивів та вирази масивів у списках, позначених як [ключ], [старт: зупинка] або [старт: зупинка: крок]. Індекс базується на нулі

та є від'ємним відносно кінця. Нарізання переміщує елементи від початкового індексу до (але не враховуючи) індексу зупинки. Третій варіант зрізу називається кроком або висотою, що дозволяє пропускати та обертати елементи. Індекс зрізу можна опустити, наприклад [:] повертає копію всього списку. Кожен елемент зрізу є неглибокою копією.

Python чітко розмежовує вирази та висловлювання, що відрізняється від таких мов, як Common Lisp, Scheme або Ruby. Це призводить до дублювання певних функцій.

Python зазвичай використовується в проектах зі штучного інтелекту та машинного навчання з такими бібліотеками, як TensorFlow, Keras, Pytorch та Scikit-learn. Як мова сценаріїв з модульною архітектурою, простим синтаксисом та розширеними засобами обробки текстів, Python зазвичай використовується для обробки природних мов. Python успішно впроваджений у багато програмних продуктів на мовах сценаріїв, включаючи програмне забезпечення з кінцевими елементами, таке як Abaqus, 3D-параметричні моделятори, такі як FreeCAD, 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, Nuke visual ефекти Composer, програми для двовимірних зображень (такі як GIMP, Inkscape, Scribus та Paint Shop Pro) та програми для приміток (наприклад, автор та група). Налагоджувач GNU використовує Python як хороший принтер для відображення складних структур, таких як контейнери C ++.

Ecpi виступає за те, що Python є найкращим вибором для сценаріїв в ArcGIS. Він також використовувався в декількох відеоіграх і був першою з трьох доступних мов програмування, що використовуються Google App Engine, а інші дві використовуються як Java та Go.

Багато операційних систем використовують Python як стандартний компонент. Він поставляється з більшістю дистрибутивів Linux, AmigaOS 4 (з

використанням Python 2.7), FreeBSD (як пакет), NetBSD, OpenBSD (як пакет) та macOS, і може використовуватися з командного рядка (терміналу). Багато дистрибутивів Linux використовують інсталятори, написані на Python: Ubuntu використовує інсталятор Ubiquity, тоді як Red Hat Linux та Fedora використовують інсталятор Anaconda. Gentoo Linux використовує Python у своїй системі управління пакетами Portage.

Python широко використовується для захисту інформації, включаючи розробку експлойт-програм.

Більшість програм Sugar Laptop XO, які зараз розробляються Sugar Labs, написані на Python. Проект одноплатної комп'ютерної програми Raspberry Pi прийняв Python як основну мову програмування користувача.

LibreOffice включає Python і має намір замінити Java на Python. Його постачальник скриптів Python є основною функцією у версії 4.0 від 7 лютого 2013 року.

Виходячи з обсягу та потужних особливостей мови Python, описаних вище, можна зробити висновок, що вона універсальна. Ось чому для виконання цієї роботи було обрано саме цю мову програмування.

PyCharm - це інтегроване середовище розробки (IDE), що використовується в комп'ютерному програмуванні, зокрема для мови Python. Він розроблений чеською компанією JetBrains. Він забезпечує аналіз коду, графічний налагоджувач, інтегрований тестер одиниць, інтеграцію із системами контролю версій (VCSes), а також підтримує веб-розробку з Django, а також аналіз даних з Anaconda.

PyCharm є крос-платформним, з версіями для Windows, macOS та Linux. Видання Community випускається під ліцензією Apache, а також існує Professional Edition з додатковими функціями - випущено під власною ліцензією.

Бета-версія була випущена в липні 2010 року, а версія 1.0 з'явилася через 3 місяці. Версія 2.0 була випущена 13 грудня 2011 року, версія 3.0 - 24 вересня 2013 року, а версія 4.0 - 19 листопада 2014 року

PyCharm Community Edition, версія з відкритим кодом PyCharm, стала доступною 22 жовтня 2013 р.

Мова розмітки HyperText або HTML - це стандартна мова розмітки для документів, призначених для відображення у веб-браузері. Йому можуть допомогти такі технології, як каскадні таблиці стилів (CSS) та мови сценаріїв, такі як JavaScript. Веб-браузери отримують документи HTML з веб-сервера або з локального сховища та надають документи на мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично та включає ознаки зовнішнього вигляду документа.

Елементи HTML - це будівельні блоки HTML-сторінок. За допомогою конструкцій HTML зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у відтворену сторінку. HTML забезпечує засіб для створення структурованих документів, позначаючи структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML розмежовані тегами, записаними за допомогою кутових дужок. Такі теги, як <img /> та <input />, безпосередньо вводять вміст на сторінку. Інші теги, такі як <p>, оточують та надають інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не відображають теги HTML, але використовують їх для інтерпретації вмісту сторінки.

HTML може вбудовувати програми, написані мовою сценаріїв, наприклад JavaScript, що впливає на поведінку та вміст веб-сторінок. Включення CSS визначає вигляд та макет вмісту. Консорціум World Wide Web (W3C), колишній

супровідник HTML і поточний супровід стандартів CSS, заохочує використання CSS над явним презентаційним HTML з 1997 року.

У 1980 році фізик Тім Бернерс-Лі, підрядник CERN, запропонував та створив прототип INQUIRE - системи для використання дослідниками CERN та обміну документами. У 1989 році Бернерс-Лі написав записку, пропонуючи систему гіпертексту на основі Інтернету. Бернерс-Лі вказав HTML та написав програмне забезпечення для браузера та сервера наприкінці 1990 року. Того року Бернерс-Лі та інженер систем обробки даних CERN Роберт Кайо співпрацювали над спільним запитом на фінансування, але проект офіційно не був прийнятий ЦЕРНом. У своїх особистих замітках від 1990 року він перерахував "деякі з багатьох областей, в яких використовується гіпертекст", і поставив енциклопедію на перше місце. Першим загальнодоступним описом HTML був документ під назвою "Теги HTML", вперше згаданий в Інтернеті Тімом Бернерсом-Лі наприкінці 1991 року. Він описує 18 елементів, що містять початковий, відносно простий дизайн HTML. За винятком тегу гіперпосилання, на них сильно вплинув SGMLguid, власний стандартний формат загальної узагальненої розмітки (SGML), що базується на CERN. Одинадцять із цих елементів все ще існують у HTML 4.

HTML - це мова розмітки, яку веб-браузери використовують для інтерпретації та складання тексту, зображень та іншого матеріалу на візуальні чи звукові веб-сторінки. Характеристики за замовчуванням для кожного елемента розмітки HTML визначаються у браузері, і ці характеристики можуть бути змінені або покращені додатковим використанням CSS дизайнера веб-сторінок. Багато текстових елементів містяться у технічному звіті ISO TR TR TR 9537 Техніка використання SGML, який, у свою чергу, охоплює особливості мов раннього форматування тексту, таких як команда RUNOFF, розроблена на початку 1960-х для CTSS (Сумісний час -Sharing System) операційна система: ці

команди форматування були отримані з команд, що використовуються набірщиками для ручного форматування документів. Однак концепція SGML узагальненої розмітки базується на елементах (вкладених анотованих діапазонах з атрибутами), а не просто на ефектах друку, а також на поділі структури та розмітки; HTML поступово рухався в цьому напрямку за допомогою CSS.

Бернерс-Лі вважав HTML додатком SGML. Формально його було визначено Робочою групою з питань Інтернет-інженерії (IETF) із публікацією першої пропозиції щодо специфікації HTML в середині 1993 року "Мова розмітки гіпертексту (HTML)" Інтернет-проект Бернерса-Лі та Дана Конноллі, який включив визначення типу документа SGML для визначення граматики. Термін дії проекту закінчився через шість місяців, але відзначився визнанням користувацького тегу браузера NCSA Mosaic для вбудовування вбудованих зображень, що відображає філософію IETF щодо заснування стандартів на вдалих прототипах. Подібним чином, конкуруючий Інтернет-проект проекту "HTML + (формат розмітки гіпертексту)" Дейва Реггетта з кінця 1993 року пропонував стандартизувати вже впроваджені функції, такі як таблиці та заповнені форми.

Після закінчення строків дії HTML та HTML + на початку 1994 р. IETF створив робочу групу HTML, яка в 1995 р. Завершила "HTML 2.0", перша специфікація HTML, призначена для розгляду як стандарту, на якому мають базуватися майбутні реалізації.

Подальший розвиток під егідою IETF було зупинено конкуруючими інтересами. З 1996 року специфікації HTML підтримуються консорціумом World Wide Web (W3C) за участю комерційних постачальників програмного забезпечення. Однак у 2000 р. HTML також став міжнародним стандартом (ISO / IEC 15445: 2000). HTML 4.01 був опублікований наприкінці 1999 р., А

подальші помилки були опубліковані до 2001 р. У 2004 р. Розпочато розробку HTML5 у Робочій групі веб-технологій прикладних технологій (WHATWG), яка стала спільним результатом із W3C у 2008 р., А завершена та стандартизована 28 жовтня 2014 р.

Семантичний HTML - це спосіб написання HTML, який підкреслює значення закодованої інформації над її поданням (виглядом). HTML включає семантичну розмітку з самого початку, але також включає презентаційну розмітку, таку як теги <font>, <i> та . Існують також семантично нейтральні теги span і div. Починаючи з кінця 90-х, коли каскадні таблиці стилів починали працювати в більшості браузерів, веб-авторам пропонувалось уникати використання презентаційної розмітки HTML з метою розділення презентації та вмісту.

У обговоренні Семантичної Мережі 2001 року Тім Бернерс-Лі та інші навели приклади способів того, як інтелектуальні програмні "агенти" можуть одного разу автоматично сканувати Інтернет і знаходити, фільтрувати та співвідносити раніше не пов'язані, опубліковані факти на благо людей . Такі агенти не є звичними навіть зараз, але деякі ідеї веб-2.0, змішувачів та веб-сайтів порівняння цін можуть наближатися. Основна різниця між цими гібридами веб-додатків та семантичними агентами Бернерса-Лі полягає в тому, що поточне узагальнення та гібридизація інформації, як правило, розробляється веб-розробниками, які вже знають веб-розташування та семантику API конкретних даних, які вони бажають розім’яти, порівняти та об’єднати.

Важливим типом веб-агента, який виконує сканування та читання веб-сторінок автоматично, без попереднього знання того, що він може знайти, є веб-сканер або павук пошукової системи. Ці програмні агенти залежать від семантичної чіткості знайдених ними веб-сторінок, оскільки використовують різні техніки та алгоритми для читання та індексування мільйонів веб-сторінок

на день та надання веб-користувачам засобів пошуку, без яких корисність Всесвітньої павутини значно зменшиться.

Для того, щоб павуки пошукових систем могли оцінювати значення фрагментів тексту, які вони знаходять у документах HTML, а також для тих, хто створює змішувачі та інші гібриди, а також для більш автоматизованих агентів у міру їх розробки, існують семантичні структури в HTML потрібно широко і одноманітно застосовувати, щоб з'ясувати значення опублікованого тексту.

Презентаційні теги розмітки застаріли в поточних рекомендаціях HTML та XHTML. Більшість презентаційних функцій попередніх версій HTML більше не дозволяються, оскільки вони призводять до погіршення доступності, вищих витрат на обслуговування сайту та більших розмірів документів [86].

Хороший семантичний HTML також покращує доступність веб-документів (див. Також Рекомендації щодо доступності веб-вмісту). Наприклад, коли засіб зчитування з екрана чи аудіо браузер може правильно встановити структуру документа, він не втратить час для людей із вадами зору, читаючи повторну або неактуальну інформацію, коли він був правильно розмічений. Всесвітня павутина складається переважно з документів HTML, що передаються з веб-серверів у веб-браузери за допомогою протоколу передачі гіпертексту (HTTP). Однак HTTP використовується для обслуговування зображень, звуку та іншого вмісту, крім HTML. Щоб веб-браузер міг знати, як обробляти кожен отриманий документ, разом із документом передається інша інформація. Ці метадані зазвичай включають тип MIME (наприклад, text / html або application / xhtml + xml) та кодування символів.

У сучасних браузерах тип MIME, який надсилається разом із документом HTML, може вплинути на спосіб початкової інтерпретації документа. Очікується, що документ, надісланий із типом XHTML MIME, буде добре сформованим XML; синтаксичні помилки можуть призвести до того, що

браузер не зможе його відтворити. Той самий документ, надісланий із типом HTML MIME, може бути успішно відображений, оскільки деякі браузери більш поблажливі до HTML.

У рекомендаціях W3C зазначено, що документи XHTML 1.0, які відповідають інструкціям, викладеним у Додатку С до рекомендації, можуть бути позначені типом MIME. У XHTML 1.1 також зазначено, що документи XHTML 1.1 повинні бути позначені типом MIME.

Більшість графічних поштових клієнтів дозволяють використовувати підмножину HTML (часто неправильно визначену), щоб забезпечити форматування та семантичну розмітку, недоступну для простого тексту. Сюди може входити друкарська інформація, наприклад кольорові заголовки, наголошений текст і цитований текст, вбудовані зображення та схеми. Багато таких клієнтів включають як редактор графічного інтерфейсу для створення електронних повідомлень HTML, так і механізм візуалізації для їх відображення. Використання HTML в електронній пошті дехто критикує через проблеми сумісності, оскільки це може допомогти замаскувати фішинг-атаки, через проблеми з доступністю для незрячих або людей із вадами зору, оскільки це може заплутати спам-фільтри та тому, що розмір повідомлення більше, ніж звичайний текст. Додаток HTML (HTA; розширення файлу ".hta") - це програма Microsoft Windows, яка використовує HTML та динамічний HTML у браузері для забезпечення графічного інтерфейсу програми. Звичайний HTML-файл обмежується моделлю безпеки веб-браузера, він передається лише веб-серверам і маніпулює лише об'єктами веб-сторінки та файлами cookie. HTA працює як повністю довірена програма і, отже, має більше привілеїв, таких як створення / редагування / видалення файлів та записів у реєстрі Windows. Оскільки вони працюють поза моделлю безпеки браузера, HTA неможливо виконати через

HTTP, але їх потрібно завантажити (як і файл EXE) і виконати з локальної файлової системи.

Каскадні таблиці стилів (CSS) - це мова таблиці стилів, що використовується для опису презентації документа, написаного мовою розмітки, такою як HTML. CSS - це наріжна технологія Всесвітньої павутини, поряд із HTML та JavaScript.

CSS призначений для розділення презентації та вмісту, включаючи макет, кольори та шрифти. Це розділення може покращити доступність вмісту, забезпечити більшу гнучкість та контроль у специфікації характеристик презентації, дозволити спільному форматуванню декілька веб-сторінок, вказавши відповідний CSS в окремому файлі .css, що зменшує складність та повторюваність структурного вмісту, а також дозволяє файл .css, який буде кешовано, щоб покращити швидкість завантаження сторінки між сторінками, що ділять файл, та його форматування.

Поділ форматування та вмісту також робить можливим подання однієї і тієї ж сторінки розмітки в різних стилях для різних методів візуалізації, таких як екран, друк, голосом (через мовний браузер або зчитувач екрана) та на основі Брайля тактильні пристосування. CSS також має правила альтернативного форматування, якщо доступ до вмісту здійснюється на мобільному пристрої. Каскадне ім'я походить із зазначеної пріоритетної схеми, щоб визначити, яке правило стилю застосовується, якщо більше одного правила відповідає певному елементу. Ця схема каскадного пріоритету передбачувана.

Специфікації CSS підтримуються Консорціумом World Wide Web (W3C). Текст / css типу Інтернет-носія (тип MIME) зареєстровано для використання з CSS RFC 2318 (березень 1998 р.). W3C забезпечує безкоштовну службу перевірки CSS для документів CSS.

На додаток до HTML, інші мови розмітки підтримують використання CSS, включаючи XHTML, звичайний XML, SVG та XUL.

CSS має простий синтаксис і використовує ряд англійських ключових слів, щоб вказати імена різних властивостей стилю.

Таблиця стилів складається зі списку правил. Кожне правило або набір правил складається з одного або декількох селекторів та блоку оголошень.

У CSS селектори оголошують, до якої частини розмітки застосовується стиль, збігаючи теги та атрибути в самій розмітці.

Селектори можуть застосовувати такі документи:

- всі елементи певного типу, напр. заголовки другого рівня h2
- елементи, визначені атрибутом, зокрема:
    - id: унікальний ідентифікатор у документі, ідентифікований префіксом хешу, наприклад #id
    - клас: ідентифікатор, який може анотувати кілька елементів у документі, ідентифікований префіксом точки, наприклад .classname
- елементи залежно від того, як вони розміщені щодо інших у дереві документів.

Класи та ідентифікатори чутливі до регістру, починаються з літер і можуть містити буквено-цифрові символи, дефіси та підкреслення. Клас може застосовуватися до будь-якої кількості екземплярів будь-яких елементів. Ідентифікатор може застосовуватися лише до одного елемента.

Псевдокласи використовуються в селекторах CSS, щоб дозволити форматування на основі інформації, яка не міститься в дереві документів. Одним із прикладів широко використовуваного псевдокласу є: hover, який визначає вміст лише тоді, коли користувач "вказує" на видимий елемент, як правило, утримуючи над ним курсор миші. Він додається до селектора, як у:

hover або #elementid: hover. Псевдо-клас класифікує елементи документа, такі як: link або: відвідується, тоді як псевдо-елемент робить виділення, яке може складатися з часткових елементів, таких як :: first-line або :: first-letter.

Селектори можуть поєднуватися різними способами для досягнення великої специфічності та гнучкості. Кілька селекторів можуть бути об’єднані в рознесений список, щоб вказати елементи за розташуванням, типом елемента, ідентифікатором, класом або будь-якою їх комбінацією. Порядок селекторів важливий. Наприклад, div .myClass {color: red;} застосовується до всіх елементів класу myClass, що знаходяться всередині елементів div, тоді як .myClass div {color: red;} застосовується до всіх елементів div, що знаходяться всередині елементів класу myClass. Цього не слід плутати з об'єднаними ідентифікаторами, такими як div.myClass {color: red;}, який застосовується до елементів div класу myClass.

Блок оголошень складається зі списку оголошень у фігурних дужках. Кожна декларація сама складається з властивості, двокрапки (:) та значення. Якщо в блоці є кілька оголошень, для відокремлення кожного оголошення потрібно вставити крапку з комою (;). Може бути використана необов’язкова крапка з комою після останньої (або одиничної) декларації. Властивості вказані в стандарті CSS. Кожна властивість має набір можливих значень. Деякі властивості можуть впливати на будь-який тип елементів, а інші стосуються лише певних груп елементів.

Значеннями можуть бути ключові слова, такі як "центр" або "успадкувати", або числові значення, такі як 200px (200 пікселів), 50vw (50 відсотків ширини області перегляду) або 80% (80 відсотків ширини батьківського елемента). Значення кольору можна вказати за допомогою ключових слів (наприклад, "червоний"), шістнадцяткових значень (наприклад, # FF0000, також скорочено # F00), значень RGB за шкалою від 0 до 255

(наприклад, rgb (255, 0, 0)), значень RGBA які вказують як кольорову, так і альфа-прозорість (наприклад, rgba (255, 0, 0, 0.8)), або значення HSL або HSLA (наприклад, hsl (000, 100%, 50%), hsla (000, 100%, 50%, 80 %)).

Ненульові числові значення, що представляють лінійні міри, повинні включати одиницю довжини, яка є або алфавітним кодом, або абревіатурою, як у 200px або 50vw; або знак відсотка, як у 80%. Деякі одиниці - см (сантиметр); дюйми (дюйми); мм (міліметр); шт (піка); і pt (точка) - абсолютні, що означає, що розмір, що відображається, не залежить від структури сторінки; інші - em (em); ex (ex) та px (pixel) - відносні, що означає, що такі фактори, як розмір шрифту батьківського елемента, можуть впливати на відтворене вимірювання. Ці вісім одиниць були властивістю CSS 1 і зберігались у всіх наступних переробках. Запропонований модуль значень та одиниць CSS Рівень 3, якщо буде прийнятий як рекомендація W3C, надасть ще сім одиниць довжини: ch; Q; rem; vh; vmax; vmin; та vw.

До CSS майже всі презентаційні атрибути HTML-документів містилися в HTML-розмітці. Всі кольори шрифту, стилі тла, вирівнювання елементів, межі та розміри мали бути чітко описані, часто неодноразово, в HTML. CSS дозволяє авторам перенести більшу частину цієї інформації в інший файл, таблицю стилів, в результаті чого HTML стає значно простішим.

Наприклад, заголовки (елементи h1), підзаголовки (h2), підзаголовки (h3) тощо визначаються структурно за допомогою HTML. У друку та на екрані вибір шрифту, розміру, кольору та акценту для цих елементів є презентаційним. До CSS автори документів, які хотіли присвоїти такі типографічні характеристики, скажімо, всім заголовкам h2, повинні були повторювати розмітку презентації HTML для кожного входження цього типу заголовка. Це зробило документи більш складними, більшими, більш схильними до помилок і важкими в обслуговуванні. CSS дозволяє відокремлювати презентацію від структури. CSS

може визначати колір, шрифт, вирівнювання тексту, розмір, межі, інтервали, макет та багато інших типографічних характеристик, і може робити це самостійно для екранних та друкованих подань. CSS також визначає невізуальні стилі, такі як швидкість читання та акценти для звукових читачів тексту. Зараз W3C не підтримує використання всієї презентаційної розмітки HTML.

Наприклад, під HTML до CSS елемент заголовка, визначений червоним текстом, буде записаний як:

<h1> Глава 1. </h1>

Використовуючи CSS, один і той же елемент можна закодувати, використовуючи властивості стилю замість презентаційних атрибутів HTML:

<h1 style = "color: red;"> Глава 1. </h1>

Переваги цього можуть бути не одразу ясними, але сила CSS стає більш очевидною, коли властивості стилю розміщуються у внутрішньому елементі стилю або, ще краще, у зовнішньому файлі CSS. Наприклад, припустимо, документ містить елемент стилю:

```
< style >
   h1 {
      колір: червоний;
   }
</style>
```

Тоді всі елементи h1 у документі автоматично стануть червоними, не вимагаючи явного коду. Якщо згодом автор хотів зробити елементи h1 синіми, це можна зробити, змінивши елемент стилю на:

```
< style >
   h1 {
      колір: синій;
   }
```

</style>

а не шляхом копіткого перегляду документа та зміни кольору для кожного окремого елемента h1.

Стилі також можна помістити у зовнішній файл CSS, як описано нижче, і завантажити, використовуючи синтаксис, подібний до:

<link href = "path / to / file.css" rel = "stylesheet" type = "text / css">

Це додатково відокремлює стиль від документа HTML і дає змогу переробляти стилі декількох документів, просто редагуючи спільний зовнішній файл CSS.

Інформація про CSS може бути надана з різних джерел. Такими джерелами можуть бути веб-браузер, користувач та автор. Інформацію від автора можна класифікувати вбудовано, тип носія, важливість, специфічність селектора, порядок правил, успадкування та визначення властивостей. Інформація про стиль CSS може бути в окремому документі або вбудована в документ HTML. Можна імпортувати декілька таблиць стилів. Залежно від використовуваного вихідного пристрою можуть застосовуватися різні стилі; наприклад, екранна версія може сильно відрізнятися від друкованої версії, так що автори можуть адаптувати презентацію відповідно до кожного носія.

Таблиця стилів з найвищим пріоритетом контролює відображення вмісту. Декларації, не встановлені у джерелі з найвищим пріоритетом, передаються джерелу з нижчим пріоритетом, наприклад стилю агента користувача. Процес називається каскадним.

Однією з цілей CSS є надання користувачам більшого контролю над презентацією. Хтось, кому важко читати червоні курсивні заголовки, може застосувати іншу таблицю стилів. Залежно від браузера та веб-сайту, користувач може вибирати з різних таблиць стилів, наданих дизайнерами, або може видаляти всі додані стилі та переглядати сайт, використовуючи стиль

браузера за замовчуванням, або може замінювати лише стиль червоного курсиву, не змінюючи атрибути.

Спадкування є ключовою особливістю CSS; він покладається на взаємозв'язок предок-нащадок. Спадкування - це механізм, за допомогою якого властивості застосовуються не лише до зазначеного елемента, але і до його нащадків. Спадщина спирається на дерево документів, яке є ієрархією елементів XHTML на сторінці на основі вкладеності. Нащадкові елементи можуть успадковувати значення властивостей CSS від будь-якого елемента-предка, що їх включає. Як правило, нащадні елементи успадковують властивості, пов’язані з текстом, але властивості, пов’язані з вікнами, не успадковуються. Властивості, які можна успадкувати, - це колір, шрифт, інтервал між літерами, висота рядка, стиль списку, вирівнювання тексту, відступ тексту, перетворення тексту, видимість, пробіли та пробіли між словами. Властивості, які неможливо успадкувати, це фон, межа, відображення, плаваюче та очищення, висота та ширина, поле, мінімальна та максимальна висота та -ширина, контур, переповнення, відступ, положення, оформлення тексту, вертикальне вирівнювання та z -індекс.

Спадкування може бути використана, щоб уникнути неодноразового декларування певних властивостей у таблиці стилів, дозволяючи коротший CSS.

Спадкування в CSS - це не те саме, що успадкування в мовах програмування на основі класів, де можна визначити клас B як "як клас A, але з модифікаціями". За допомогою CSS можна стилізувати елемент з "класом A, але з модифікаціями". Однак неможливо визначити такий клас CSS B, який потім можна було б використовувати для стилізації декількох елементів без необхідності повторювати модифікації.

CSS вперше був запропонований Håkon Wium Lie 10 жовтня 1994 р. У той час Лі працювала з Тімом Бернерс-Лі у ЦЕРНі. Кілька інших мов таблиць

стилів для Інтернету було запропоновано приблизно в той же час, і обговорення публічних списків розсилки та всередині Консорціуму всесвітньої павутини призвели до того, що перша Рекомендація W3C CSS (CSS1) [24] була опублікована в 1996 році. Зокрема, пропозиція Берт Бос мав вплив; він став співавтором CSS1 і вважається співавтором CSS.

Таблиці стилів існують в тій чи іншій формі з часів заснування Стандартної узагальненої мови розмітки (SGML) у 1980-х роках, а CSS був розроблений для надання таблиць стилів для Інтернету. Однією з вимог до мови веб-таблиць стилів було те, щоб таблиці стилів надходили з різних джерел у мережі. Тому існуючі мови таблиць стилів, такі як DSSSL та FOSI, не підходили. CSS, з іншого боку, дозволяє стилю документа впливати на декілька таблиць стилів шляхом "каскадного" стилю.

По мірі зростання HTML він охоплював все більше різноманітних стилістичних можливостей, щоб задовольнити запити веб-розробників. Ця еволюція дала дизайнеру більше контролю над зовнішнім виглядом сайту за рахунок більш складного HTML. Варіації реалізації веб-браузера, такі як ViolaWWW та WorldWideWeb, ускладнювали послідовний зовнішній вигляд сайту, а користувачі мали менший контроль над тим, як відображався веб-вміст. Браузер / редактор, розроблений Тімом Бернерсом-Лі, мав таблиці стилів, які були жорстко закодовані в програмі. Тому таблиці стилів не можуть бути пов'язані з документами в Інтернеті. Роберт Кайо, також із CERN, хотів відокремити структуру від презентації, щоб різні таблиці стилів могли описувати різні презентації для друку, презентації на екрані та редактори.

Поліпшення можливостей веб-презентацій було темою, яка цікавила багатьох у веб-спільноті, і дев'ять різних мов таблиць стилів було запропоновано в списку розсилки у стилі www. З цих дев’яти пропозицій дві особливо вплинули на те, що стало CSS: каскадні таблиці стилів HTML та

пропозиція таблиці стилів на основі потоку (SSP). Два браузери служили тестами для початкових пропозицій; Лі працював з Івом Лафоном над впровадженням CSS у браузері Arena Дейва Реггетта. Берт Бос реалізував власну пропозицію SSP у браузері Argo. Після цього Lie і Bos спільно працювали над розробкою стандарту CSS ("H" було вилучено з назви, оскільки ці таблиці стилів також можна застосовувати до інших мов розмітки, крім HTML).

Пропозиція Лі була представлена на конференції "Мозаїка та Інтернет" (згодом названа WWW2) в Чикаго, штат Іллінойс, у 1994 році, і знову разом з Бертом Босом у 1995 році. [23] Приблизно в цей час W3C вже був створений і зацікавився розробкою CSS. З цією метою він організував семінар-практикум під головуванням Стівена Пембертона. Це призвело до того, що W3C додав роботу над CSS до результатів роботи редакційної комісії HTML-перегляду (ERB). Лі і Бос були основним технічним персоналом у цьому аспекті проекту, в якому брали участь також додаткові члени, в тому числі Томас Рірдон з Microsoft. У серпні 1996 року компанія Netscape Communication Corporation представила альтернативну мову таблиць стилів, що називається JavaScript Sheets Style (JSSS). Специфікація ніколи не була закінчена, і вона застаріла. До кінця 1996 р. CSS був готовий стати офіційним, а Рекомендація CSS рівня 1 була опублікована в грудні.

Розробка HTML, CSS та DOM велася в одній групі - редакційній комісії HTML (ERB). На початку 1997 року ЄБР було розділено на три робочі групи: Робоча група HTML, головою якої став Дан Конноллі з W3C; Робоча група DOM під головуванням Лорен Вуд із SoftQuad; та робоча група CSS під головуванням Кріса Ліллі з W3C.

Робоча група CSS розпочала вирішення питань, які не були вирішені на рівні CSS 1, в результаті чого рівень CSS рівня 2 був створений 4 листопада

1997 р. Він був опублікований як рекомендація W3C 12 травня 1998 р. Рівень CSS 3, який був розпочате в 1998 році, все ще знаходиться в стадії розробки станом на 2014 рік.

У 2005 році Робочі групи CSS вирішили суворіше виконувати вимоги до стандартів. Це означало, що вже опубліковані стандарти, такі як CSS 2.1, CSS 3 Selectors та CSS 3 Text, були повернуті з Рекомендацій кандидатів на рівень робочого проекту.

Фреймворки CSS - це заздалегідь підготовлені бібліотеки, які призначені для спрощення стилю веб-сторінок, що відповідають стандартам, з використанням мови каскадних таблиць стилів. Структури CSS включають Blueprint, Bootstrap, Cascade Framework, Foundation та Materialize. Як і бібліотеки мови програмування та сценаріїв, фреймворки CSS зазвичай включаються як зовнішні .css-аркуші, на які посилається HTML-код <head>. Вони надають ряд готових варіантів проектування та розміщення веб-сторінки. Незважаючи на те, що багато з цих фреймворків опубліковано, деякі автори використовують їх здебільшого для швидкого створення прототипів або для вивчення, і вважають за краще "виготовляти" CSS, який підходить для кожного опублікованого сайту, без проектування, обслуговування та завантаження, що має багато невикористаних функцій в стилі сайту.

У міру збільшення розміру ресурсів CSS, що використовуються в проекті, команді розробників часто потрібно прийняти рішення про загальну методологію проектування, щоб тримати їх організованими. Цілями є простота розробки, простота співпраці під час розробки та продуктивність розгорнутих таблиць стилів у браузері. Популярні методології включають OOCSS (об'єктно-орієнтований CSS), ACSS (атомний CSS), oCSS (органічний каскадний аркуш стилів), SMACSS (масштабована та модульна архітектура для CSS) та BEM (блок, елемент, модифікатор).

2.2 Проектування функціоналу системи

Діаграма варіантів використання (Use Case Diagram) є важливим інструментом при проектуванні і розробці програмного забезпечення, оскільки вона дозволяє моделювати функціональність системи з точки зору її користувачів. Ця діаграма визначає, як користувачі будуть взаємодіяти з системою і як система відповідатиме на їхні дії.

Основна мета використання діаграм варіантів використання - це визначення і моделювання функціональних вимог до системи. Вона дозволяє уточнити, як система повинна працювати з точки зору користувача і які функції вона повинна виконувати. Це допомагає команді розробників краще зрозуміти потреби користувачів і відповідно до цього проектувати і розробляти систему.

Діаграма варіантів використання також допомагає визначити головні актори (користувачі або інші системи), які будуть взаємодіяти з системою, і визначити основні сценарії взаємодії між ними. Це дозволяє виявити можливі проблеми та недоліки в дизайні системи ще на ранніх етапах розробки і внести відповідні зміни.

Загалом, використання діаграм варіантів використання є важливим етапом при проектуванні програмного забезпечення, оскільки вона допомагає зрозуміти потреби користувачів, уточнити функціональні вимоги і покращити якість інтерфейсу системи.

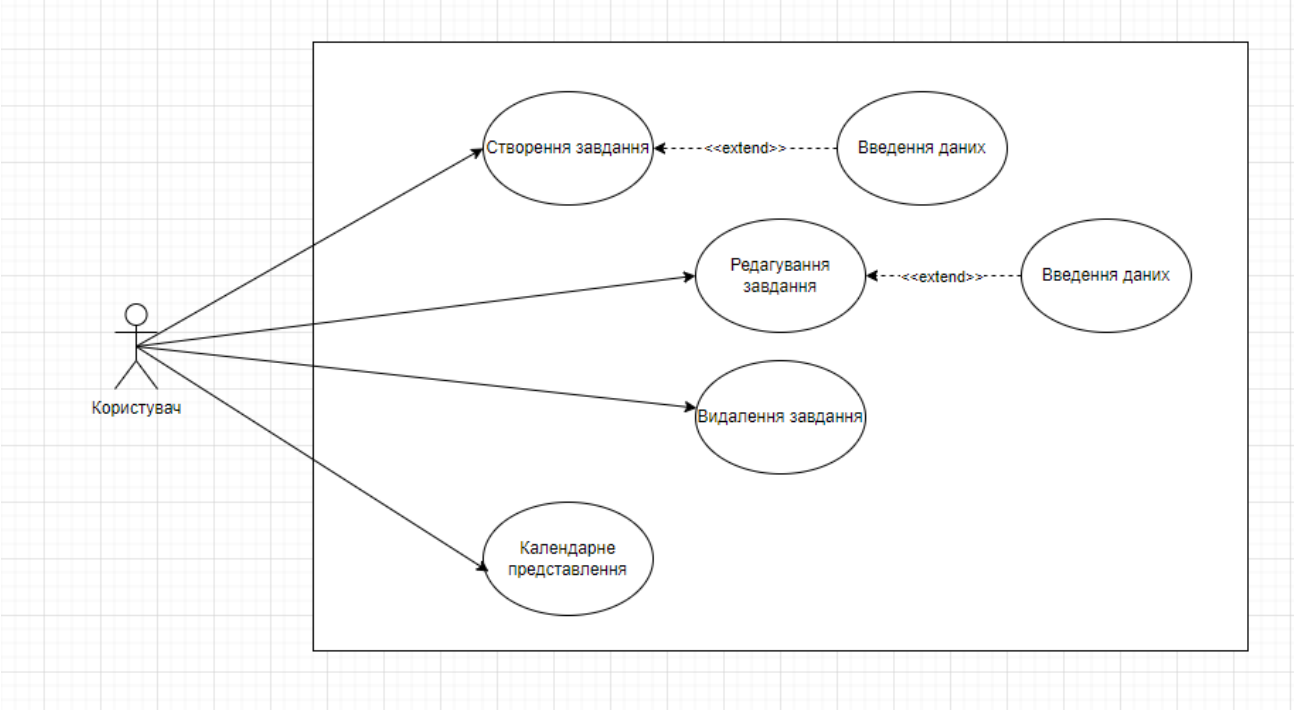


Рисунок 2.1 — Діаграма варіантів використання

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМИ І АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Розробка внутрішньої будови

Діаграма класів (Class Diagram) є важливим інструментом при проектуванні і розробці програмного забезпечення, оскільки вона дозволяє моделювати структуру системи з точки зору класів, які утворюють цю систему. Ця діаграма визначає класи, їх атрибути і методи, а також взаємозв'язки між класами.

Основна мета використання діаграм класів - це визначення структури системи і взаємозв'язків між її частинами. Вона дозволяє уточнити, які класи існують в системі, які дані вони зберігають і які операції вони виконують. Це допомагає команді розробників краще зрозуміти архітектуру системи і відповідно до цього проектувати і розробляти її.

Діаграма класів також допомагає виявити можливі проблеми та недоліки в дизайні системи ще на ранніх етапах розробки і внести відповідні зміни. Вона дозволяє визначити, які класи повинні бути додані або видалені, які атрибути і методи повинні бути модифіковані, щоб покращити якість системи.

Загалом, використання діаграм класів є важливим етапом при проектуванні програмного забезпечення, оскільки вона допомагає зрозуміти структуру системи, уточнити взаємозв'язки між класами і покращити якість її архітектури.

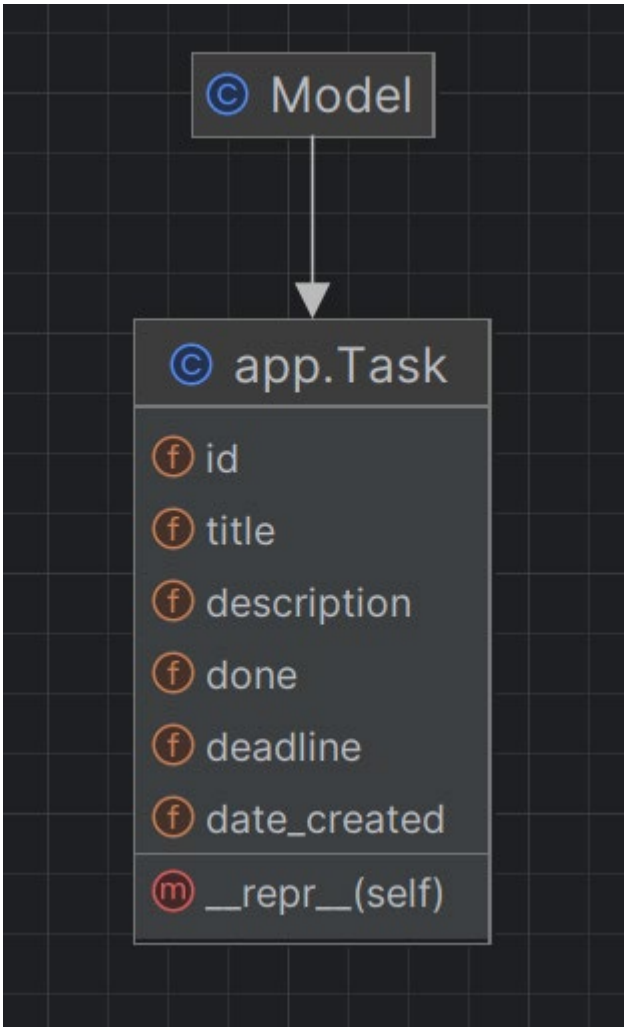


Рисунок 3.1 — Діаграма класів системи

3.2 Розробка інтерфейсу користувача

Графічний інтерфейс користувача, або GUI, представляє собою специфічну форму інтерфейсу, яка дає можливість користувачам взаємодіяти з електронними пристроями за допомогою графічних елементів, таких як піктограми та аудіоіндикатори, замість текстових інтерфейсів користувача, які передбачають введення команд через клавіатуру комп'ютера.

Взаємодія в GUI зазвичай здійснюється шляхом безпосереднього взаємодії з графічними елементами, а не шляхом введення текстових команд. Інтерфейси командного рядка (CLI) були поширені раніше, і GUI були введені відповідно до спрощення використання пристроїв та програм, оскільки CLI вимагали введення текстових команд.

GUI застосовуються не лише на комп'ютерах, але й на різних кишенькових пристроях, таких як MP3-плеєри, портативні медіапрогравачі, ігрові пристрої, смартфони і різноманітні побутові, офісні та промислові пристрої для управління. GUI часто не використовуються у випадках, коли дисплей має низьку роздільну здатність, як у відеоіграх (де використовується інтерфейс зі складнішою внутрішньою структурою), або коли пристрої мають інші типи дисплеїв, як от об'ємні дисплеї.

Розробка візуальної композиції та тимчасової поведінки GUI є ключовою частиною програмування прикладних програм, які спрямовані на взаємодію людини з комп'ютером. Вона спрямована на полегшення використання для користувачів, покращення ефективності та дотримання принципів юзабіліті. Дизайн, орієнтований на користувача, використовується для того, щоб забезпечити адаптованість візуального інтерфейсу до потреб користувачів. Модель адаптивної реалізації контенту веб сторінки дозволяє створювати гнучкі структури, в яких інтерфейс незалежний від функцій програми та опосередковано пов'язаний із ними, що дозволяє легко налаштувати графічний

інтерфейс. Це робить можливим вибір або створення іншої обгортки вебсторінки і полегшує зміну інтерфейсу з урахуванням потреб користувача.

У більшості випадків, великі віджети, такі як вікна, надають рамку або контейнер для відображення основного вмісту презентації, такого як веб-сторінка, електронна пошта або малюнок. Менші віджети служать для введення користувачем і взаємодії з інформацією.

GUI може бути спеціально розробленим для конкретних програм та використовувати різноманітні види екранів і пристроїв для управління, такі як ATM, POS-термінали в ресторанах, каси самообслуговування у роздрібних магазинах, інформаційні кіоски в громадських місцях та багато інших. GUI також використовуються в автомобільних навігаційних системах та мультимедійних центрах, а також на сенсорних екранах кишенькових пристроїв та ігрових систем.

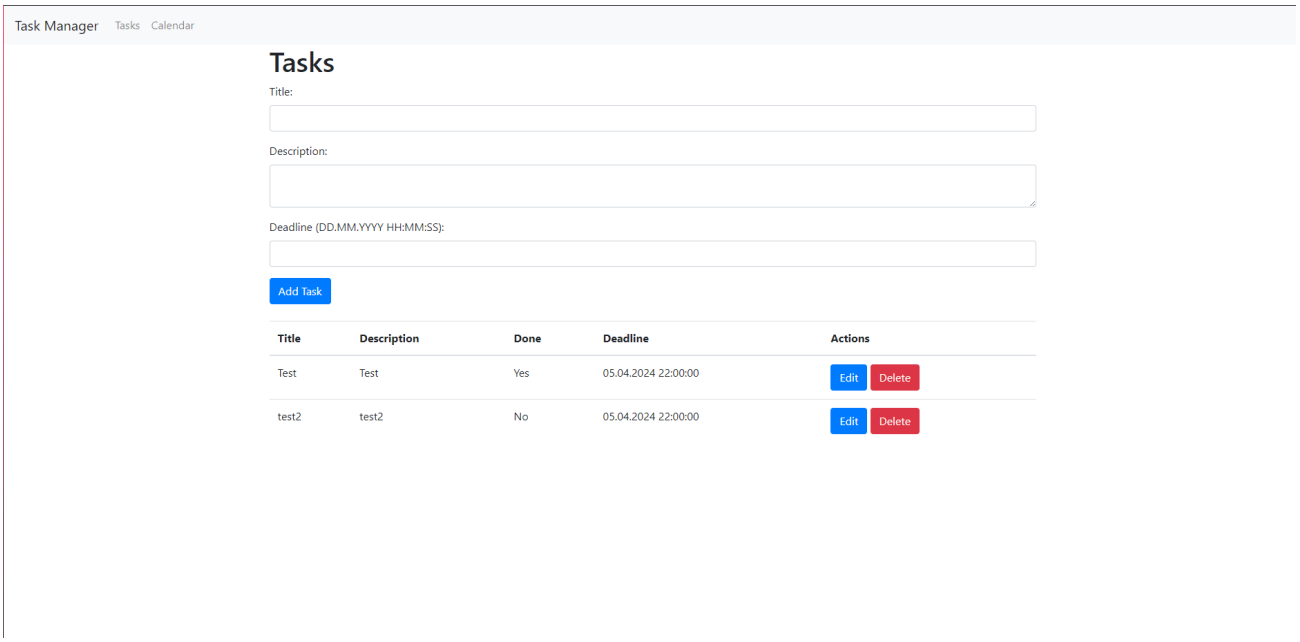


Рисунок 3.2 — Головна сторінка

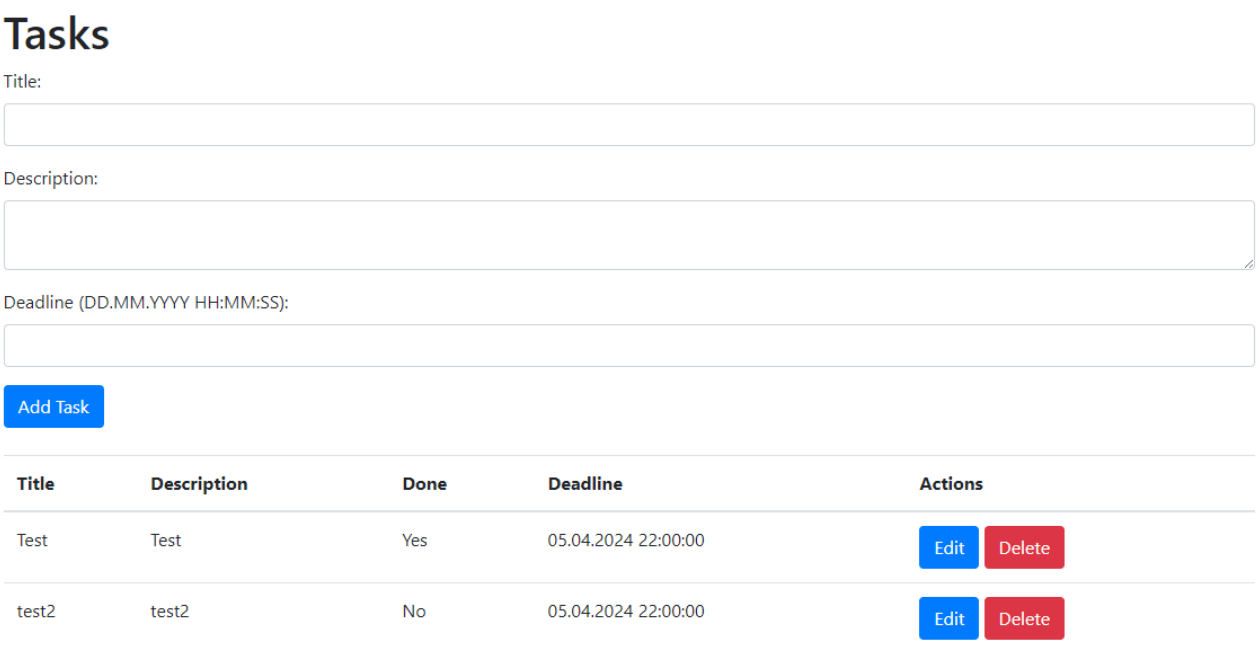


Рисунок 3.3 — Завдання

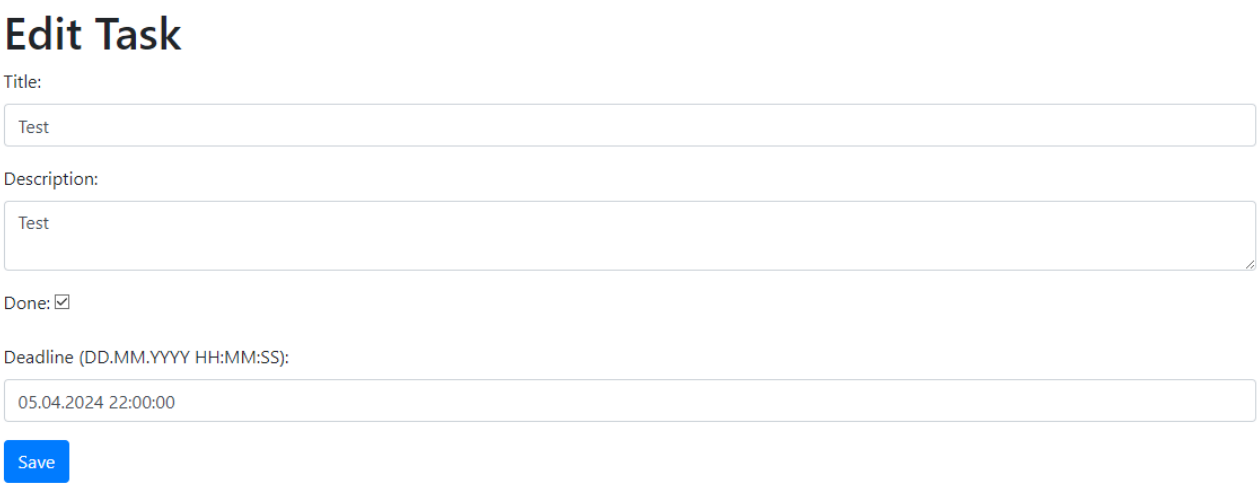


Рисунок 3.4 — Редагування завдання

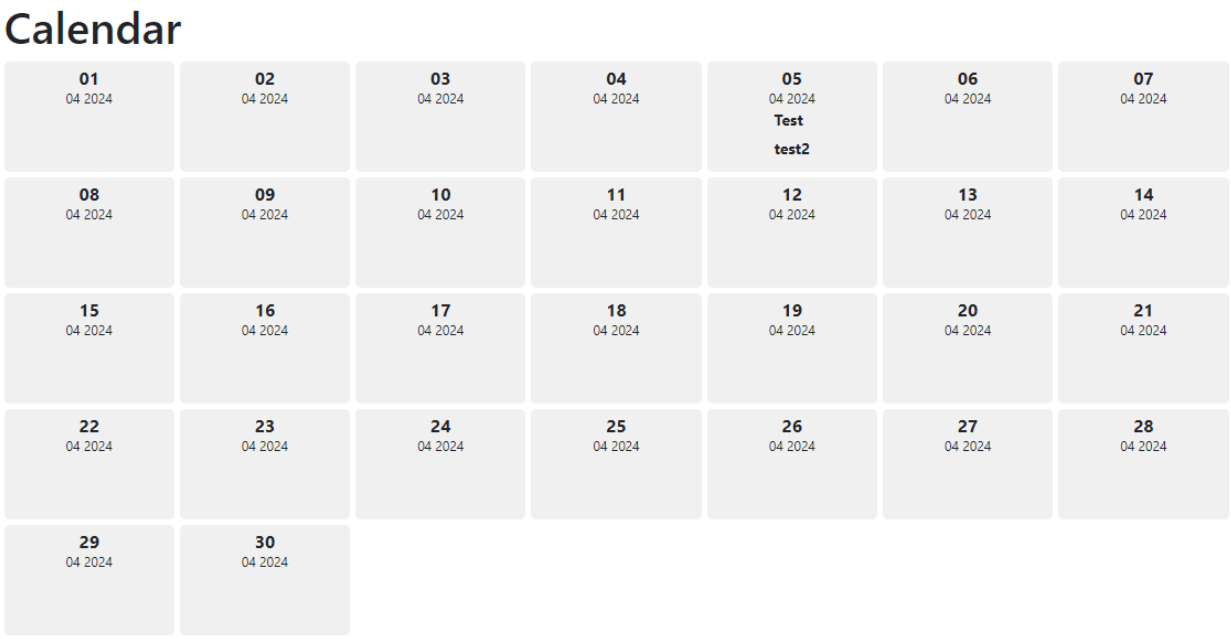


Рисунок 3.5 — Календарне представлення

3.3 Тестування

Тестування програмного забезпечення – це акт перевірки артефактів і поведінки програмного забезпечення, що тестується, шляхом перевірки та верифікації. Тестування програмного забезпечення також може забезпечити об’єктивне, незалежне уявлення про програмне забезпечення, щоб дозволити бізнесу оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають, але не обов’язково обмежуються:

- аналіз вимог до продукту щодо повноти та правильності в різних контекстах, таких як галузева перспектива, бізнес-перспектива, доцільність та життєздатність впровадження, зручність використання, продуктивність, безпека, міркування інфраструктури тощо.
- перегляд архітектури продукту та загального дизайну продукту
- робота з розробниками продуктів над удосконаленням техніки кодування, шаблонів проектування, тестів, які можна написати як частину коду на основі різних методів, таких як граничні умови тощо.
- виконання програми або програми з метою перевірки поведінки
- перевірка інфраструктури розгортання та пов’язаних скриптів та автоматизації
- брати участь у виробничій діяльності, використовуючи методи моніторингу та спостереження

Тестування програмного забезпечення може надати користувачам або спонсорам об’єктивну, незалежну інформацію про якість програмного забезпечення та ризик його збою.

Несправності та збої

Помилки програмного забезпечення виникають через наступний процес: Програміст робить помилку (помилку), що призводить до помилки (дефекту, помилки) у вихідному коді програмного забезпечення. Якщо ця помилка

виконана, у певних ситуаціях система дасть неправильні результати, що призведе до збою.

Не всі несправності обов'язково призведуть до збоїв. Наприклад, помилки в мертвому коді ніколи не призведуть до збоїв. Несправність, яка не виявила збоїв, може призвести до збою при зміні середовища. Приклади цих змін у середовищі включають програмне забезпечення, яке запускається на новій апаратній платформі комп’ютера, зміни вихідних даних або взаємодію з іншим програмним забезпеченням. Одна несправність може призвести до широкого спектру симптомів відмови.

Не всі помилки програмного забезпечення викликані помилками кодування. Одним із поширених джерел дорогих дефектів є прогалини у вимогах, тобто нерозпізнані вимоги, які призводять до помилок, пропущених розробником програми. Прогалини вимог часто можуть бути нефункціональними вимогами, такими як тестованість, масштабованість, ремонтопридатність, продуктивність та інші вимоги. безпеки.

Статичне, динамічне та пасивне тестування

У тестуванні програмного забезпечення існує багато підходів. Огляди, покрокові інструкції або перевірки називаються статичним тестуванням, тоді як виконання запрограмованого коду з заданим набором тестових випадків називається динамічним тестуванням.

Статичне тестування часто є неявним, як-от коректура, а також коли інструменти програмування/текстові редактори перевіряють структуру вихідного коду, а компілятори (попередні компілятори) перевіряють синтаксис і потік даних як статичний аналіз програми. Динамічне тестування відбувається під час запуску самої програми. Динамічне тестування може розпочатися до того, як програма буде на 100% завершена, щоб перевірити окремі розділи коду та застосувати до дискретних функцій або модулів. Типовими методами для них

є або використання заглушок/драйверів, або виконання з середовища налагодження.

Статичне тестування передбачає перевірку, тоді як динамічне також передбачає перевірку.

Пасивне тестування означає перевірку поведінки системи без будь-якої взаємодії з програмним продуктом. На відміну від активного тестування, тестувальники не надають жодних тестових даних, а переглядають системні журнали та трасування. Вони шукають моделі та специфічну поведінку, щоб приймати якісь рішення. Це пов’язано з перевіркою часу виконання в автономному режимі та аналізом журналу.

Дослідницький підхід

Дослідницьке тестування — це підхід до тестування програмного забезпечення, який коротко описується як одночасне навчання, розробка тесту та виконання тесту. Джем Канер, який ввів цей термін у 1984 році,:2 визначає дослідницьке тестування як «стиль тестування програмного забезпечення, який підкреслює особисту свободу та відповідальність окремого тестувальника за постійну оптимізацію якості своєї роботи, розглядаючи тест- пов’язане навчання, дизайн тесту, виконання тесту та інтерпретація результатів тесту як взаємодопоміжні дії, які виконуються паралельно протягом усього проекту».

«Коробковий» підхід

Методи тестування програмного забезпечення традиційно поділяються на тестування білого та чорного ящиків. Ці два підходи використовуються для опису точки зору, яку дотримується тестувальник під час розробки тестових випадків. До методології тестування програмного забезпечення також можна застосувати гібридний підхід, який називається тестуванням сірого ящика. Оскільки концепція тестування сірого ящика, яка розробляє тести на основі

конкретних елементів дизайну, стає все більш популярною, це «довільне розходження» між тестуванням чорного та білого ящиків дещо зникло.

Таблиця 3.1

Тестування додатку

№	Тест-кейс	Очікуваний результат	Результат
1	Відкриття додатку	Завантаження головної сторінки додатку	Завантаження головної сторінки додатку
2	Відображення списку завдань	Перегляд усіх доступних завдань	Перегляд усіх доступних завдань
3	Додавання нового завдання	Додання нового завдання до списку	Нове завдання успішно додано до списку
4	Видалення завдання	Успішне видалення завдання зі списку	Завдання успішно видалено зі списку
5	Редагування завдання	Успішне редагування завдання	Завдання успішно відредаговано
6	Відміна редагування завдання	Скасування внесених змін до завдання	Зміни до завдання успішно скасовані
7	Позначення завдання як виконане	Позначення завдання як виконане та його зникнення зі списку	Завдання успішно позначене як виконане і відсутнє у списку

ВИСНОВКИ

У ході роботи було проведено аналіз предметної області, де визначено основні поняття та принципи тайм-менеджменту. Було розглянуто основні етапи розробки програмного забезпечення та важливість використання паттерну MVC для побудови ефективних програмних рішень.

Огляд існуючих рішень на ринку дозволив виявити переваги та недоліки наявних програмних засобів для управління часом. Це стало основою для постановки задачі та формулювання вимог до розроблюваного програмного продукту.

У другому розділі роботи було визначено методологію дослідження, вибрано інструментарій розробки та проведено проектування функціоналу системи. Ці кроки допомогли створити загальний план розробки та визначити основні функції програми.

У третьому розділі описано процес розробки програми, зокрема розробку внутрішньої будови, інтерфейсу користувача та процес тестування. Результатом цього етапу є створена програма, яка відповідає вимогам та вирішує поставлені завдання.

Загальним висновком можна сказати, що розроблений програмний продукт є ефективним інструментом для управління часом, який може бути використаний для підвищення продуктивності та організації робочого та особистого часу користувачів. Програма відповідає сучасним вимогам до програмного забезпечення та може бути використана в різних сферах діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stella Cottrell (2013). The Study Skills Handbook by Stella Cottrell (University of Leeds). Palgrave Macmillan. pp. 123+. ISBN 978-1-137-28926-1.[permanent dead link]
2. ДСТУ 8604:2015 Інтерфейс людина-машина. Основні принципи безпеки, маркування та ідентифікація. Принципи включення (IEC 60447:2004, IDT). Чинний від 2016-07-01. Вид. офіц.
3. Communications, Richard Lewis (internationally renowned linguist). "How Different Cultures Understand Time". Business Insider. Retrieved 2018-12-04.
4. Pant, Bhaskar (2016-05-23). "How various cultures perceive deadlines varies". Harvard Business Review. Retrieved 2018-12-04.
5. Duranti, Giancarlo; Di Prata, Olvers (2009). "Everything is about time: does it have the same meaning all over the world?".
6. "NIMH – Attention Deficit Hyperactivity Disorder". www.nimh.nih.gov. Archived from the original on 2016-12-29. Retrieved 2018-01-05.
7. Hallowell, Edward M.; Ratey, John J. (1994). Driven To Distraction: Recognizing and Coping with Attention Deficit Disorder from Childhood Through Adulthood. Touchstone. pp. 73–76. ISBN 9780684801285. Retrieved 2013-07-30.
8. "The 80/20 Rule And How It Can Change Your Life". Forbes. Archived from the original on 2017-11-17. Retrieved 2017-09-16.
9. Ferriss, Timothy. (2007). The 4-hour workweek: escape 9-5, live anywhere, and join the new rich (1st ed.). New York: Crown Publishers. ISBN 978-0-307-35313-9. OCLC 76262350.

10. Dwight D. Eisenhower (August 19, 1954). Address at the Second Assembly of the World Council of Churches. Archived from the original on 2015-04-02. Evanston, Illinois. (retrieved 31 March 2015.)
11. ДСТУ 8604:2015 Інтерфейс людина-машина. Основні принципи безпеки, маркування та ідентифікація. Принципи включення (IEC 60447:2004, IDT). Чинний від 2016-07-01. Вид. офіц.
12. Background on the Eisenhower quote and citations to how it was picked up in media references afterwards are detailed in: O'Toole, Garson (9 May 2014). "What Is Important Is Seldom Urgent and What Is Urgent Is Seldom Important". Quote Investigator. Archived from the original on 11 April 2015.
13. Fowler, Nina (September 5, 2012). "App of the week: Eisenhower, the to-do list to keep you on task". Venture Village.
14. Drake Baer (April 10, 2014), "Dwight Eisenhower Nailed A Major Insight About Productivity" Archived 2015-04-02 at the Wayback Machine, Business Insider, (accessed 31 March 2015)
15. McKay; Brett; Kate (October 23, 2013). "The Eisenhower Decision Matrix: How to Distinguish Between Urgent and Important Tasks and Make Real Progress in Your Life". A Man's Life, Personal Development. Archived from the original on 2014-03-22. Retrieved 2014-03-22.
16. ISO/IEC TS 33074:2020 Information technology – Process assessment, (Process capability assessment model for service management)
17. "The Eisenhower Method". fluent-time-management.com. Archived from the original on 2014-03-03.
18. Morgenstern, Julie (2004). Time Management from the Inside Out: The Foolproof System for Taking Control of Your Schedule—and Your Life (2nd ed.). New York: Henry Holt/Owl Books. p. 285. ISBN 0-8050-7590-9.

19. Mackenzie, Alec (1972). The Time Trap (3rd ed.). AMACOM - A Division of American Management Association. pp. 41–42. ISBN 081447926X.
20. LeBoeuf, Michael (1979). Working Smart. Warner Books. pp. 52–54. ISBN 0446952737.
21. Nightingale, Earl (1960). "Session 11. Today's Greatest Adventure". Lead the Field (unabridged audio program). Nightingale-Conant. Archived from the original on 2013-01-08.
22. Lakein, Alan (1973). How to Get Control of Your Time and Your Life. New York: P.H. Wyden. ISBN 0-451-13430-3.
23. Аровіна М. Перспективи застосування Mobile Apps на регіональному інформаційному ринку України. Схід. 2016. № 3. С. 5-10.
24. Ахо А. В., Хопкрофт Дж. Е., Ульман Дж. Структури даних і алгоритми. Київ: Проф-інформ, 2011. 400 с.
25. Гнатієнко Г. М., Снитюк В. Є. Експертні технології прийняття рішень: Монографія. Київ: ТОВ «Маклаут», 2008. 444 с.
26. Голомбінськи К., Хаген Р. Додай повітря! Основи візуального дизайну для графіки веб і мультимедіа. Пер. з англ. Н. А. Ріміцан. Київ: Рекорд-К, 2013. 272 с
27. Івасенко А. Г., Грідасов А. Ю., Павленко В. А. Інформаційні системи і технології в економіці і управлінні. Київ: Просвіта, 2007. 160 с.
28. Кращі планувальники: 10 додатків для тайм-менеджменту. MOYO. URL: https://www.moyo.ua/ua/news/luchshie_planirovshchiki_10_prilozheniy_dlya_taymmenedzhmenta.html
29. Пароджанов В. Вчись писати, читати та розуміти алгоритми. Київ: Просвіта, 2016. 458
30. Трофімов В. В. Інформаційні системи і технології в економіці і управлінні. Київ: Вища освіта, 2007. 480 с.

31. Трофімов В. В., Павловська Т. А. Алгоритмізація та програмування: підручник для академічного бакалавріату. Харків: Вид-во Юрайт, 2016. 137 с. 89
32. Хорець В. О., Шушура О. М. Мобільний додаток «Органайзер студента». II Всеукраїнська науково-практична інтернет-конференція молодих вчених та здобувачів вищої освіти «Сучасна молодь в світі інформаційних технологій». Херсон, 2021. С. 42-44.
33. Hay L. Researching UX: Analytics by SitePoint Pty. Ltd, 2017. 106 p.
34. Wigdor D., Brave D. W. NUI World: Designing Natural User Interfaces for Touch and Gesture. Morgan Kaufmann, 2011. 264 p.
35. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги. Чинний від 2017-07-01. Вид. Офіц.
36. Пастернак, Ю.В. Морозов. Вісник «Комп'ютерні системи та мережі»: НУ«Львівська політехніка», Львів, 2011-№717.-С. 108-113
37. М'якшило О.М. CASE-технології у проектуванні інформаційних систем: електронний навчальний посібник для студ. вещих навч. закладів / О.М. М'якшило, Л.Г. Загоровська,– К.: НУХТ, 2017. – 190 с.
38. ДСТУ ISO/IEC/IEEE 29119-1:2017. Інженерія систем і програмних засобів. Тестування програмних засобів. Частина 1. Поняття та визначення (ISO/IEC/IEEE 29119-1:2013, IDT).
39. ДСТУ 2941:1994. Системи оброблення інформації. Розроблення систем. Терміни та визначення
40. Власій О. О., Винничук М. Д. Розробка мобільних додатків засобами блочного програмування: Навчально методичний посібник. Прикарпатський національний університет імені Василя Стефаника, 2021. 130 с.

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
from calendar import monthrange

from flask import Flask, jsonify, request, render_template, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timedelta

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tasks.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text, nullable=True)
    done = db.Column(db.Boolean, default=False)
    deadline = db.Column(db.DateTime, nullable=True)
    date_created = db.Column(db.DateTime, default=datetime.utcnow)

    def __repr__(self):
        return f"Task('{self.title}', '{self.description}', '{self.done}',
'{self.deadline}', '{self.date_created}')"
```

```
with app.app_context():
    db.create_all()

@app.route('/')
def index():
    return redirect(url_for('get_tasks'))

@app.route('/tasks', methods=['GET'])
def get_tasks():
    tasks = Task.query.all()
    return render_template('tasks.html', tasks=tasks)

@app.route('/tasks', methods=['POST'])
def create_task():
    data = request.form
    deadline = datetime.strptime(data['deadline'], '%d.%m.%Y %H:%M:%S') if data['deadline'] else None
    new_task = Task(title=data['title'], description=data.get('description', ''), done=False, deadline=deadline)
    db.session.add(new_task)
    db.session.commit()
    return redirect(url_for('get_tasks'))

@app.route('/tasks/<int:task_id>', methods=['GET', 'PUT', 'DELETE', 'POST'])
def manage_task(task_id):
    task = Task.query.get(task_id)
    if not task:
```

```
        return jsonify({'error': 'Task not found'}), 404

    if request.method == 'GET':
        return render_template('edit_task.html', task=task)
    elif request.method == 'POST':
        data = request.form
        task.title = data.get('title', task.title)
        task.description = data.get('description', task.description)
        task.done = True if data.get('done') == 'on' else False
        task.deadline = datetime.strptime(data['deadline'], '%d.%m.%Y %H:%M:%S') if data['deadline'] else None
        db.session.commit()
        return redirect(url_for('get_tasks'))
    elif request.method == 'PUT':
        data = request.form
        task.title = data.get('title', task.title)
        task.description = data.get('description', task.description)
        task.done = True if data.get('done') == 'on' else False
        task.deadline = datetime.strptime(data['deadline'], '%d.%m.%Y %H:%M:%S') if data['deadline'] else None
        db.session.commit()
        return redirect(url_for('get_tasks'))
    elif request.method == 'DELETE':
        db.session.delete(task)
        db.session.commit()
        return redirect(url_for('get_tasks'))
```

```
@app.route('/calendar')
def calendar():
    # Отримуємо першу та останню дату поточного місяця
    today = datetime.now()
    first_day = today.replace(day=1)
    last_day = today.replace(day=monthrange(today.year, today.month)[1])

    # Генеруємо список дат поточного місяця
    current_date = first_day
    dates_in_month = []
    while current_date <= last_day:
        dates_in_month.append(current_date)
        current_date += timedelta(days=1)

    # Створюємо словник, де ключ - це дата, а значення - список завдань на цю дату
    calendar = {}
    for date in dates_in_month:
        tasks = Task.query.filter(Task.deadline >= date, Task.deadline < date + timedelta(days=1)).all()
        calendar[date.strftime('%Y-%m-%d')] = tasks

    return render_template('calendar.html', calendar=calendar)



if __name__ == '__main__':
```

```
    app.run(debug=True)
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Task Manager</title>
    <style>
        {%  block styles %}
        {% endblock %}
    </style>
    <link                                                    rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">Task Manager</a>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link" href="/tasks">Tasks</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/calendar">Calendar</a>
                </li>
            </ul>
        </div>
```

```
        </nav>
        <div class="container">
            {% block content %}
            {% endblock %}
        </div>


    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/css/bootstrap.min.css">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/js/bootstrap.bundle.min.js"></
script>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/daterangepicker/daterangepicker.css">
    <script
src="https://cdn.jsdelivr.net/npm/daterangepicker/daterangepicker.min.js"></script>
        <script>
        $(document).ready(function() {
            $('#deadline').daterangepicker({
                singleDatePicker: true,
                timePicker: true,
                timePicker24Hour: true,
                locale: {
                    format: 'DD.MM.YYYY HH:mm:ss'
                }
            });
```

```
    });
</script>
</body>
</html>
{% extends 'base.html' %}
{% block styles %}
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            text-align: center;
        }
        .calendar {
            display: grid;
            grid-template-columns: repeat(7, 1fr);
            gap: 5px;
        }
        .day {
            background-color: #f0f0f0;
            padding: 5px;
            border-radius: 5px;
            height: 100px; /* Фіксована висота клітинки */
            display: flex;
            flex-direction: column;
            justify-content: flex-start;
            align-items: center;
        }
```

```
            .day h3 {
                margin: 0;
                font-size: 16px;
                font-weight: bold;
            }
            .day span {
                font-size: 12px;
            }
            .tasks {
                list-style-type: none;
                padding: 0;
                margin: 0;
                text-align: left;
                overflow-y: auto; /* Додано прокрутку, якщо завдань більше, ніж поміщається */
                max-height: 60px; /* Максимальна висота списку завдань */
            }
            .task {
                font-size: 14px;
                margin-bottom: 5px;
            }
        {% endblock %}

        {% block content %}
            <h1>Calendar</h1>
            <div class="month-year">
                <h2>{{ current_month_name }} {{ current_year }}</h2>
```

```
        </div>
        <div class="calendar">
            {% for date, tasks in calendar.items() %}
            <div class="day">
                <h3>{{ date.split('-')[2] }}</h3> <!-- Відображення лише числа дати
-->
                <span>{{ date.split('-')[1] }} {{ date.split('-')[0] }}</span> <!--
Відображення місяця і року -->
                <ul class="tasks">
                    {% for task in tasks %}
                    <li class="task">
                        <strong>{{ task.title }}</strong>
                    </li>
                    {% endfor %}
                </ul>
            </div>
            {% endfor %}
        </div>
            {% endblock %}
```