**Архітектура проекту**

Проект «Система керування завданнями» - це веб-додаток, який дозволяє користувачам створювати, читати, оновлювати та видаляти завдання. Додаток також дозволяє користувачам створювати списки завдань і додавати завдання до списків.

Система має наступні основні можливості:

* Створення завдань. Користувачі можуть створювати нові завдання, вказуючи назву, опис, дату виконання та статус завдання.
* Читання завдань. Користувачі можуть переглядати список всіх завдань, а також список завдань, які належать до певного списку.
* Оновлення завдань. Користувачі можуть змінювати назву, опис, дату виконання або статус завдання.
* Видалення завдань. Користувачі можуть видаляти завдання.
* Створення списків завдань. Користувачі можуть створювати нові списки завдань, вказуючи назву та опис списку.
* Додання завдань до списків. Користувачі можуть додавати завдання до списків.

Архітектура проекту «Система керування завданнями» є трирівневою, з використанням архітектури Model-View-Controller (MVC):

* Перший рівень – це рівень представлення, який відповідає за взаємодію з користувачем. Він складається з веб-сайту, який написаний за допомогою фреймворка ASP.NET Core. Веб-сайт складається з контролерів, які відповідають за обробку запитів користувачів, і шаблонів, які відповідають за відображення даних користувачеві.
* Другий рівень – це рівень ділової логіки, який відповідає за обробку даних і бізнес-правил. Він складається з пакетів todoList.Business і todoList.DataAccess. Пакет todoList.Business містить інтерфейси і сервіси, які відповідають за бізнес-логіку системи. Пакет todoList.DataAccess містить репозиторії, які відповідають за доступ до бази даних.
* Третій рівень – це рівень бази даних, який зберігає дані системи. Він складається з таблиць, які містять інформацію про завдання, списки завдань та користувачів.

**Контролери**

Контролери є найближчими до користувача компонентами системи. Вони отримують HTTP-запити від користувача, обробляють їх і повертають відповідь. Відповідь може бути в текстовому форматі, HTML-сторінкою, або іншим типом даних.

Контролери мають доступ до даних, які зберігаються в базі даних. Вони можуть використовувати ці дані для обробки запитів користувачів.

Наприклад, контролер, який відповідає за перегляд списку завдань, отримує HTTP-запит від користувача, який містить ідентифікатор списку. Контролер використовує цей ідентифікатор для отримання даних про список з бази даних. Потім контролер повертає HTML-сторінку, яка відображає дані про список.

**Шаблони**

Шаблони відповідають за відображення даних користувачеві. Вони отримують дані від контролера і використовують їх для створення HTML-сторінки.

Шаблони можуть використовувати HTML, CSS і JavaScript для створення інтерфейсу користувача.

Наприклад, шаблон, який відображає список завдань, отримує від контролера дані про список. Шаблон використовує ці дані для створення сторінки, яка містить інформацію про завдання.

**Інтерфейси**

Інтерфейси описують бізнес-логіку системи. Вони визначають методи, які повинні реалізувати сервіси. Інтерфейси роблять систему більш гнучкою. Вони дозволяють змінювати реалізацію бізнес-логіки без зміни коду контролерів.

Наприклад, інтерфейс ITaskService визначає методи для створення, читання, оновлення та видалення завдань.

**Сервіси**

Сервіси реалізують бізнес-логіку системи. Вони використовують інтерфейси для взаємодії з репозиторіями.

Наприклад, сервіс TaskService реалізує інтерфейс ITaskService. Сервіс використовує репозиторій TaskRepository для доступу до бази даних.

**Репозиторії**

Репозиторії відповідають за доступ до бази даних. Репозиторії роблять систему більш незалежною від бази даних. Вони дозволяють змінювати базу даних без зміни коду сервісів.

**База даних**

База даних зберігає дані системи. Вона складається з таблиць, які містять інформацію про завдання, списки завдань та користувачів. База даних може бути реалізована в будь-якому системі управління базами даних (СУБД). У цьому випадку використовується СУБД PostgreSQL.

**Переваги архітектури MVC**

Архітектура MVC має ряд переваг:

– Чітке розділення відповідальності між компонентами системи. Це полегшує розробку, тестування та обслуговування системи.

– Зручність тестування. Контролери і шаблони можуть бути протестовані незалежно від бізнес-логіки.

– Модульність. Система може бути легко розширена або модифікована.

**Недоліки архітектури MVC**

Архітектура MVC також має деякі недоліки:

– Додатковий код. Для реалізації архітектури MVC потрібно написати додатковий код, який забезпечує взаємодію між компонентами системи.

– Комплексність. Архітектура MVC може бути складною для розуміння і реалізації.