МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра програмних систем і технологій

Дисципліна

«Архітектура та проектування програмного забезпечення»

Лабораторна робота № 3

«Моделювання та візуалізація архітектури програмного забезпечення»

на тему:

Онлайн клієнт для погодних даних

Виконав:	Гоша Д.О	**Перевірив**:	Берестов Д.С
Група	ІП3-23	Дата перевірки	
Форма навчання	денна	Оцінка	
Спеціальність	121		

2022

Мета: отримати практичні навички у архітектурному моделюванні та візуалізації ПЗ.

Завдання

1. Визначити основні (принаймні три) архітектурні проблеми, що мають стосунок до проекту. Чому та яке їх значення?
2. Визначити набір точок зору на архітектуру системи, керуючись вищезазначеними проблемами (проблема –> точка зору).
3. Обрати нотацію моделювання та візуалізацію для кожної точки зору.
4. Змоделювати архітектуру системи, фіксуючи у моделі рішення, які пов’язані з визначеним набором точок зору. Усі основні компоненти та з’єднувачі мають бути репрезентовані у моделях.

До звіту необхідно включити наступне:

- Перелік архітектурно важливих проблем для системи, обґрунтування важливості.
- Обґрунтування вибору точок зору, нотацій моделювання та візуалізацій. Важливі обмеження (стиль або обмеження в реалізації) архітектури системи.
- Опис типів з’єднувачів та їх застосованих аспектів.
- Усі побудовані візуалізації архітектурної моделі та обґрунтування архітектурних рішень (це повинно бути основним розділом звіту).
- Опис усіх можливих основних технічних проблем, ризиків та відкритих питань.
- Будь-які додаткові доречні спостереження та цікаві відкриття.

Хід роботи

1. Архітектура програми:

Мною була выбрана 3-рівнева архітектура вона є стилями розгортання, що описують поділ функціональності на сегменти, багато в чому аналогічно багатошаровій архітектурі, але в даному випадку ці сегменти можуть фізично розміщуватися на різних комп'ютерах, їх називають рівнями. Дані архітектурні стилі було створено з урахуванням компонентно-орієнтовного підходу і, зазвичай, зв'язку використовують методи платформи, а чи не повідомлення.

1. Модель
2. Контроллер
3. Представленя

2. Недоліки концепції MVC

1. Необхідність використання більшої кількості ресурсів. Складність обумовлена тим, що всі три фундаментальні блоки є абсолютно незалежними і взаємодіють між собою виключно шляхом передачі даних. Controller повинен завжди завантажити (і за необхідності створити) всі можливі комбінації змінних і передати їх у Model. Model, у свою чергу, повинен завантажити всі дані для візуалізації та передати їх у View. Наприклад, модульний підхід, модуль може безпосередньо обробляти змінні оточення і візуалізувати дані без завантаження їх в окремі секції пам'яті.2
2. Ускладнено механізм поділу програми на модулі. У концепції MVC наявність

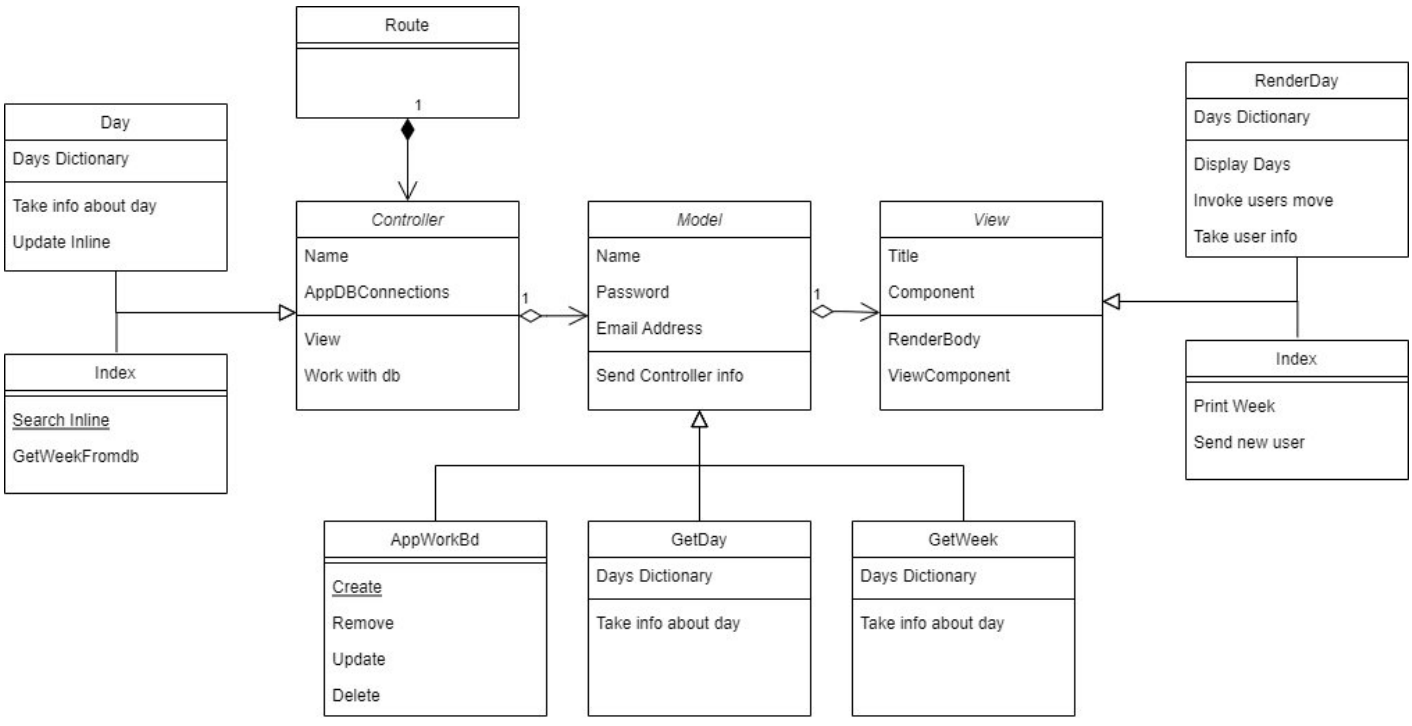
трьох блоків (Model, View, Controller) жорстко прописано. Відповідно, кожен функціональний модуль повинен складатися з трьох блоків, що, у свою чергу, дещо ускладнює архітектуру функціональних модулів програми.

3. Ускладнено процес розширення функціоналу. Проблема дуже схожа на вищеописану. Недостатньо просто написати функціональний модуль та підключити його в одному місці програми. Кожен функціональний модуль повинен складатися з трьох частин, і кожна з цих частин повинна бути підключена до відповідного блоку.

3. Переваги концепції MVC

1. Єдина концепція системи. Безперечним плюсом MVC є єдина глобальна архітектура програми. Навіть у складних системах, розробники (як ті, які розробляли систему, так і ті, що знову приєдналися) можуть легко орієнтуватися в програмних блоках. Наприклад, якщо виникла помилка в логіці обробки даних, розробник відразу відкидає 2-блоки програми (controller та view) і займається дослідженням 3-го (model). Я неодноразово дивувався, наскільки сильно спростилася локалізація проблем.
2. Спрощено механізм налагодження програми. весь механізм візуалізації тепер сконцентрований в одному програмному блоці, спростилися механізми опціонального виведення графічних елементів. Я не можу оцінити наскільки це твердження застосовно у програмуванні класичних додатків, але в Web програмуванні ця архітектурна особливість стала безперечним плюсом.

4. Архітектурні рішення стосовно візуалізації та моделювання:



Діаграма класів

Діаграма класів, показує як вибрані класи з’єдані. Тобто показує взаємодію класів у

програмі. У даному випадку , ми бачимо , що кожну конкретну проблему , тобто розширення функціоналу, потрібно розкладати на 3 частини і вносити у кожний з базових класів. А саме , наприклад якщо знадобиться розширити функціонал та додати рівень тиску до кожного дня. Потрібно змінити частину бд де зберігається інформація про день та додати колонку , яка зберігатиме рівень тиску. Так як модель агрегує контроллер то у методі , що передає модель треба додати поле , що свідчить про рівень тиску. З цього випливає , що неминуча зміна інтерфейсу, де буде друкуватися рівень тиску для користувача. Ось така невелика зміна веде до купи роботи.

В той же час, для мене залишилося відкритим питання, наскільки можна застосувати концепцію MVC для розробки звичайних програм (наприклад, для Windows). На знижку можна, але не факт, що це буде оптимально. Ну і найголовніше питання: чи буду використовувати концепцію MVC, в наступних своїх проектах? Відповідь: думаю так.

Висновок

У данному проекті було отримано практичні навички у побудовані візуалізації архітектурної моделі та обґрунтування архітектурних рішень , вибору точок зору, нотацій моделювання та візуалізацій. Важливі обмеження (стиль або обмеження в реалізації) архітектури системи. Описі типів з’єднувачів та їх застосованих аспектів