# 3 АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ ТА ОБРАНІ ТЕХНОЛОГІЇ

Під час вибору технологій для реалізації цього проекту було проаналізовано багато сучасних мов програмування та фреймворків і були обрані найактуальніші з них. Тож  було вирішено використати об’єктно-орієнтовану мову програмування C# та Microsoft .NET Framework 4.5 [12].

C# – об'єктно-орієнтована мова[13] програмування з безпечною системою типізації для платформи .NET. Перевагою даної мови є те, що розмір результуючих програмних продуктів є досить не великим і не потребує додаткових бібліотек. Синтаксис C# близький до С++ і Java. З огляду на дуже зручного об'єктно-орієнтованого дизайну, C# є гарним вибором для швидкого конструювання різних компонентів - від високорівневою бізнес логіки до системних додатків, що використовують низькорівневий код.

Також слід зазначити, що C# є і Web орієнтованим - використовуючи прості вбудовані конструкції мови ваші компоненти можуть бути легко перетворені в Web сервіси, до яких можна буде звертатися з Internet за допомогою будь-якої мови на будь-якій операційній системі. Додаткові можливості та переваги перед іншими мовами приносить в C# використання передових Web технологій.

C# підтримує універсальні методи і типи , забезпечуючи більш високий рівень безпеки і продуктивності. Вирази LINQ роблять строго типізований запит дуже зручною мовною конструкцією .

Для реалізації проекту було вирішено використовувати принципи проектування SOLID [15].

S (The Single Responsibility Principle) – принцип єдиної відповідальності. Головний сенс принципу: боротьба зі складностями, важливість яких різко зростає при розвитку логіки додатку. Короткий опис: будь-який складний клас повинен бути розбитий на кілька простих складових, що відповідають за певний аспект поведінки, що спрощує як розуміння, так і майбутній розвиток. Типові приклади порушення принципу:

– змішування логіки та інфраструктури: бізнес-логіка змішана з поданням, знаходиться всередині WCF або windows-сервісів і т.п.;

– клас/модуль вирішує завдання різних рівнів абстракції: обчислює CRC і відправляє повідомлення по електронній пошті; розбирає json-об'єкт і аналізує його вміст і т.п.

O (The Open-Closed Principle) – принцип відкритості та закритості. Визначення: програмні сутності (класи, модулі, функції і т.п.) повинні бути відкритими для розширення, але закритими для модифікації. Головний сенс принципу: обмежити поширення змін мінімальним числом класів/модулів; дозволити вести паралельну розробку шляхом фіксації інтерфейсів класів та відкритості реалізацій. Короткий опис: закритість модулів означає стабільність інтерфейсу і можливість використання класів/модулів клієнтами. Відкритість модулів означає можливість внесення змін у поведінці, шляхом зміни реалізації або ж шляхом перевизначення поведінки в спадкоємцях. Боротьба зі змінами полягає в обмеженні кількості змін мінімальним числом класів/модулів і не має на увазі можливість зміни поведінки без перекомпіляції. На практиці необхідна «гнучкість» забезпечується за рахунок успадкування та зіставлення із зразком, залежно від того, яку операцію ми хочемо спростити - додавання нового підтипу в ієрархію спадкування або додавання нової операції в сімейство типів. Типові приклади порушення принципу: розмазування інформації про ієрархію типів по всьому додатку.

L (The Liskov Substitution Principle) – принцип заміщення Барбари Лісков. Визначення: повинна бути можливість замість базового типу підставити будь-який його підтип. Головний сенс принципу: реалізуйте спадкування підтипів правильно. Короткий опис: для коректної реалізації відносини «являється», спадкоємець може послаблювати передумову і посилювати постумову (вимагати менше і гарантувати більше), при цьому інваріанти базового класу повинні виконуватися спадкоємцем. При порушенні цих правил підстановка екземплярів спадкоємця в метод, який приймає базовий клас буде призводити до непередбачуваних наслідків. Типові приклади порушення принципу: неузгоджена поведінка спадкоємців призводить до необхідності приводити екземпляри базового класу до конкретних типів спадкоємців.

I (Interface Segregation Principle) – принцип поділу інтерфейсів. Визначення: клієнти не повинні вимушено залежати від методів, якими не користуються. Головний сенс додатку: клас повинен надавати зручний інтерфейс з точки зору його різноманітних клієнтів. Короткий опис: інтерфейс класу повинен бути цілісним і узгодженим, не залежно від кількості клієнтів. Кілька різних клієнтів цілком можуть використовувати лише підмножину методів класу, до тих пір, поки інтерфейс класу залишатиметься узгодженим. Проблеми виникають тоді, коли інтерфейс класу починає розпухати або з'являються різні методи зі схожою семантикою лише для того, щоб ними було зручно користуватися певним клієнтам. Типові приклади порушення принципу:

– клас або інтерфейс містить кілька методів зі схожою семантикою, які використовуються різними клієнтами;

– інтерфейс класу занадто різнорідний і містить методи, що відповідають за слабкозв'язані операції.

D (The Dependency Inversion Principle) – принцип інверсії залежностей. Визначення: модулі верхнього рівня не повинні залежати від модулів нижнього рівня. І ті й інші повинні залежати від абстракцій. Головний сенс принципу: зробити ключові та/або мінливі залежності класу явними. Короткий опис: занадто велике число залежностей класу говорить про проблеми в дизайні. Можливо клас робить занадто багато, або ж поточний клас не вдалий, що призводить до необхідності смикання одного методу у занадто великого числа залежностей. Будь-який об'єктний дизайн являє собою деякий граф взаємодіючих об'єктів, при цьому деякі залежності є частиною реалізації і повинні створюватися безпосередньо (композиція), а деякі – передаватися йому ззовні (агрегація). Виділяти залежності особливо корисно, коли вони є мінливими (зав'язані на оточення), або ж являють собою деяку форму «стратегій». Типові приклади порушення принципу: використання Singlton [18], сервіс-локаторів або ж створення ключових залежностей класу в закритих методах. Всі ці принципи дуже важливі при розробці програмного забезпечення, а правильне використання приходить з досвідом

Паттерн архітектури Model-View-Controller (MVC) поділяє програму на три основні частини: модель, уявлення і контролер. Розділення це здійснюється як на фізичному так і на логічному рівнях.

Основна мета застосування цієї концепції полягає в відділенні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу підвищується можливість повторного використання. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і/або з різних точок зору. Наглядна схема шаблону MVC [16] приведена нижче на рисунку 3.1.

У моделі описуються основні сутності, використані в проекті. Контролер приймає і переадресовує запити користувачів завдяки своїм спеціальним методам, які обробляють запити і викликають певні відображення відповідних до запиту даних, отриманих з моделі. Представлення служить для відображення інтерфейсу сервісу.

Метою шаблону MVC є розробка гнучкого програмного дизайну, котрий під час подальшої підтримки програмної системи полегшує роботу та робить її швидшою і зрозумілішою. З’являється можливість повторного використання деяких компонентів програмної системи. MVC також полегшує тестування проектів в порівнянні з веб-проектами ASP.NET на основі веб-форм. Платформа MVC поділяє компоненти й активно використовує інтерфейси, що дозволяє тестувати окремі елементи поза решти структури.

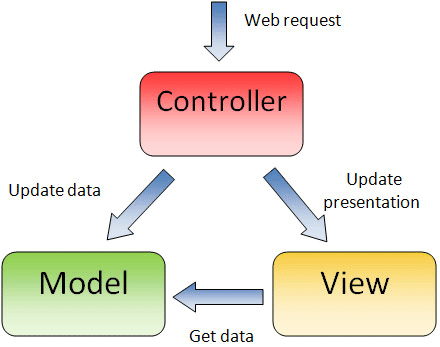


Рисунок 3.1 – Схема шаблону MVC

ASP.NET MVC визнає важливість отримання чистої, відповідної стандартам розмітки. Вбудовані методи HTML помічника надають відповідні стандартам вихідні дані. Разом з платформою ASP.NET MVC можна використовувати існуючі функції ASP.NET, наприклад вкладені головні сторінки, вбудовані вирази, декларативні серверні елементи управління, шаблони, прив'язку даних, локалізацію та багато інших аспектів.

12. Шилдт, Г. С# 4.0: полное руководство [Текст] / Г. Шилдт: пер. с англ. − М.: ООО «И.Д. Вильямc», 2011. − 1056 с.

14. Фримен A. LINQ. Язык интегрированных запросов в C# [Текст] / A. Фримен: пер. с англ. − ООО «И.Д. Вильямc», 2011. − 656 с.

15. Принципи проектування класів S.O.L.I.D [Електронний ресурс] : − портал http://blog.byndyu.ru − Режим доступу: www/URL:http://blog.byndyu.ru/2009/10/solid.html − 02.06.2016 г. − Загол. з экрану.

16. Сандерсон, С. ASP.NET MVC Framework с примерами на C# для профессионалов. [Текст] / C. Сандерсон: пер. с англ. - М.: ООО "И.Д. Вильямс", 2010. – 560 с.

18. Тепляков С. Патерны проектирования на платформе .NET [Текст] / С. Тепляков – СПб: Питер, 2015. − 320с.