Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК_____

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема "Інформаційно-аналітична система автоматизації роботи міністерства молоді та спорту"

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА
МР.ІПЗ – 11.22.33.444

Студент
Лоза Владислав Дмитрович

Науковий керівник
д.т.н. доцент Бичков Олексій Сергійович

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри
Бичков Олексій Сергійович

2021

Зміст

Перелік основних позначень, символів, скорочень

API – Application Programming Interface

ООП – Об’єктно-орієнтоване програмування

HTTP - HyperText Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

MVC – Model–View–Controller

DOM – Document Object Model

REST – Representational State Transfer

SQL – Structured Query Language

UML - Unified Modeling Language

Вступ

У зв’язку з невпинним розвитком технологій з’являться нові сфери та можливості до автоматизації роботи як різних підприємств так і різних державних установ.

З необхідністю оновлення програмних застосунків та розробки нових автоматизованих систем до кафедри програмних систем і технологій звернулись представники міністерства молоді та спорту.

Мета цієї роботи – розробка інформаційно-аналітичної системи для створення і менеджменту різних подій, таких як збори, змагання та інше. Основна ідея полягає у створенні веб-застосунку з доступом до нього різних користувачів з різними ролями, які будуть регулювати можливості та права перегляду.

Актуальність цієї роботи диктується відсутністю прийнятного рішення на даний момент і необхідністю створення застосунку для віддаленої роботи співробітників міністерства в рамках всеукраїнського карантину.

Дана робота безпосередньо пов’язана з державним планом діджиталізації. Надання можливості створення запитів на збори та змагання онлайн для спортивних секцій та тренерів з усієї України та менеджмент міжнародних змагань по типу Олімпійських ігор.

На даний момент у міністерстві молоді та спорту використовується застосунок, доступ до якого мають лише обрані співробітники міністерства та всі дані вводяться вручну. Інтерфейс виглядає як таблиця, додатком важко користуватись і як зрозуміло доступ до нього поза меж міністерства неможливий. Запити на створення різних подій відбуваються вручну, представник секції або тренер телефонують до міністерства, де записують звернення і передають його співробітнику з доступом до застосунку, який в свою чергу додає дані до таблиці.

Результат цієї роботи дозволить автоматизувати роботу міністерства, зменшити ланцюжок дій для створення подій, тим самим оптимізувавши процеси і звільнивши ресурси для задіяння їх в інших сферах та для інших потреб, що в свою чергу покращить отриманий результат та вивільнить кошти бюджету для розподілення їх до інших підрозділів або створення нових.

Розділ 1 Аналіз та збір вимог

1.1 Загальні положення

Міністерство молоді та спорту звернулося до кафедри програмних систем і технологій з проханням розробити автоматичну систему менеджменту спортивних подій.

Основна мета системи – автоматизація роботи міністерства для прискорення процесу створення спортивних подій та покращення роботи міністерства в цілому.

Перш ніж приступити до розробки необхідно налагодити комунікацію зі співробітниками міністерства, зібрати та проаналізувати вимоги для побудови якісного додатку, а саме: функціональні та нефункціональні вимоги, вимоги до дизайну; усі необхідні діаграми: варіантів використання системи, розгортання системи, діаграма послідовності.

1.2 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення - це опис функціональних можливостей та особливостей та системи що розробляється. Вони передають очікування користувачів від програмного продукту. Вимоги можуть бути прихованими або очевидними, невідомими чи відомими, несподіваними чи очікуваними з точки зору замовника.

Розробка вимог це процес збору вимог до програмного забезпечення від замовника, їх аналізу та документування.

Метою розробки вимог є розробка та підтримка «Специфікації системних вимог».

1.2.1 Функціональні вимоги

Було поставлено такі функціональні вимоги:

- застосунок має надавати можливість користувачам (співробітникам міністерства молоді та спорту, а також представниками різних спортивних секцій та тренерам) заходити до системи;
- залежно від ролі користувача застосунок має надавати різний функціонал та інтерфейс;
- для певних ролей має бути доступ до сторінки створення події;
- для певних ролей має бути доступ до календаря подій з можливістю навігації по датам та можливістю зміни типу відображення календаря (тиждень, місяць, рік);
- для певних ролей має бути можливість створення звітів з обраних видів спорту за обраний період;
- застосунок має надавати можливість використання одночасно декількох «календарів», тобто мати можливість фільтрувати дані за певним полем.

1.2.2 Нефункціональні вимоги

Було поставлено такі нефункціональні вимоги:

- продуктивність: забезпечення високошвидкісної роботи при шифруванні даних, обробці запитів багатьох користувачів водночас;
- кросплатформеність: наявність хостингу на більшості операційних систем (Windows, Linux, MacOS і т.д.);
- зручність супроводу: забезпечення наявності інструкції з розгортання системи, що дозволила б її правильно використовувати;
- надійність: відмовостійкість компонентів програмного забезпечення повинна бути на високому рівні, система повинна продовжувати своє функціонування при виході з ладу одного з компонентів.

1.2.3 Вимоги до дизайну

Було поставлено такі вимогу до дизайну додатку:

- зручний у використанні;
- навігаційний бар на верхній частині додатку для переходу по різним розділам сайту;
- приємний, інтуїтивно зрозумілий користувальницький інтерфейс у єдиному стилі;
- відсутність зайвої інформації на сторінках;
- дизайн повинен бути чуйним (змінюватись в залежності від різних форматів екрану) для забезпечення якомога більшого охоплення аудиторії та підвищення зручності виборців.

1.2.4 Діаграма варіантів використання програмного забезпечення

Діаграма варіантів використання програмного забезпечення в UML - діаграма, що відображає відносини між користувачами і можливими діями та є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

Прецеденти - можливості модельованої системи, завдяки якій користувач може отримати очікуваний результат. Прецедент відповідає окремому блоку функціоналу системи (сервісу), вказує один з варіантів її використання і описує звичайний спосіб взаємодії користувача з системою.

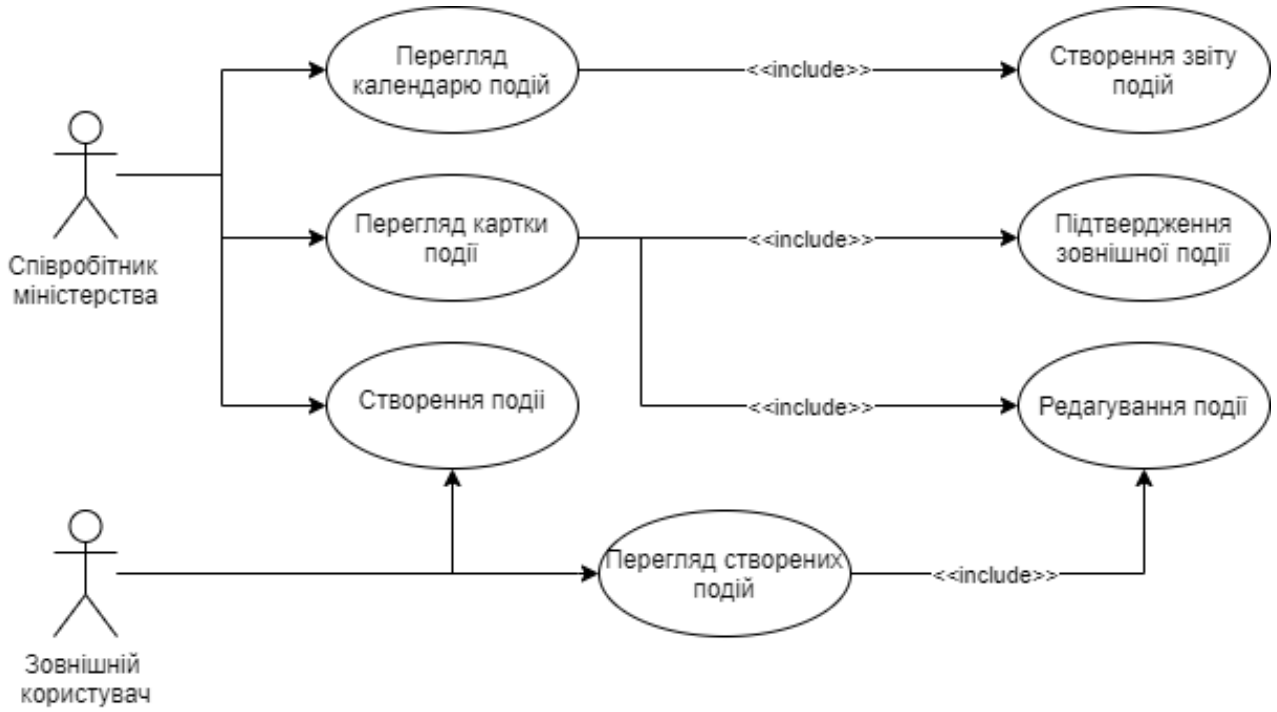

Рис. 1 Діаграма варіантів використання (прецедентів)

- Коротка специфікація прецеденту «Перегляд календарю подій»
  - Короткий опис
    - Співробітник міністерства має змогу переглядати календар подій.
  - Суб’єкт
    - Користувач у ролі «Співробітник міністерства».
  - Передумова
    - Користувач авторизується у ролі «Співробітник міністерства» та заходить на сторінку «Календар».
  - Основний потік
    - Система відображає календар з подіями з можливістю фільтрації, сортування, вибору типу відображення календаря, а також можливості створення звіту по подіям.

        - Після натискання на кнопку «Створити звіт» відкривається сторінка створення звіту на якій потрібно заповнити необхідні поля та натиснути кнопку «Створити».
    - Альтернативний потік
        - А1. Помилка серверу. Якщо сервер не може відобразити дані, то система видає повідомлення «Помилка».
        - А2. Неповна інформація. Якщо користувач не заповнив усі необхідні поля, система дає можливість ввести інформацію. Прецедент продовжується.
    - Постумови
        - У разі виникнення помилок стан системи залишається незмінним. Помилки заносяться до журналу помилок.

- Коротка специфікація прецеденту «Перегляд картки подій»
    - Короткий опис
        - Співробітник міністерства має змогу переглядати картку подій.
    - Суб’єкт
        - Користувач у ролі «Співробітник міністерства».
    - Передумова
        - Користувач авторизується у ролі «Співробітник міністерства» та заходить на сторінку «Календар», де обирає подію і натискає «Детальніше».
    - Основний потік
        - Система відображає картку події.
        - На картці є можливість редагувати поля та натиснути кнопку «Зберегти».

        - Після натискання на кнопку «Підтвердити» для події створеної «Зовнішнім користувачем» подія переходить до статусу актуальної та відображається на календарях співробітників міністерства.
    - Альтернативний потік
        - А1. Помилка серверу. Якщо сервер не може відобразити дані, то система видає повідомлення «Помилка».
        - А2. Неповна інформація. Якщо користувач не заповнив усі необхідні поля, система дає можливість ввести інформацію. Прецедент продовжується.
    - Постумови
        - У разі виникнення помилок стан системи залишається незмінним. Помилки заносяться до журналу помилок.

- Коротка специфікація прецеденту «Створення події»
    - Короткий опис
        - Співробітник міністерства та зовнішній користувач мають змогу створити нову подію.
    - Суб’єкт
        - Користувач у ролі «Співробітник міністерства» або «Зовнішній».
    - Передумова
        - Користувач авторизується у ролі «Співробітник міністерства» або «Зовнішній» та обирає пункт «Створити подію».
    - Основний потік

        - Система відображає картку події. На картці є можливість заповнити поля та натиснути кнопку «Зберегти».
    - Альтернативний потік
        - А1. Помилка серверу. Якщо сервер не може відобразити дані, то система видає повідомлення «Помилка».
        - А2. Неповна інформація. Якщо користувач не заповнив усі необхідні поля, система дає можливість ввести інформацію. Прецедент продовжується.
    - Постумови
        - У разі виникнення помилок стан системи залишається незмінним. Помилки заносяться до журналу помилок.

- Коротка специфікація прецеденту «Перегляд створених подій»
    - Короткий опис
        - Зовнішній користувач має змогу переглядати картку створеної ним події.
    - Суб’єкт
        - Користувач у ролі «Зовнішній».
    - Передумова
        - Користувач авторизується у ролі «Зовнішній» та заходить на сторінку «Календар», де обирає подію і натискає «Детальніше».
    - Основний потік
        - Система відображає картку події на якій є можливість переглянути статус.

        - На картці є можливість редагувати поля та натиснути кнопку «Зберегти», якщо подія ще не була підтверджена співробітником міністерства.
    - Альтернативний потік
        - А1. Помилка серверу. Якщо сервер не може відобразити дані, то система видає повідомлення «Помилка».
        - А2. Неповна інформація. Якщо користувач не заповнив усі необхідні поля, система дає можливість ввести інформацію. Прецедент продовжується.
    - Постумови
        - У разі виникнення помилок стан системи залишається незмінним. Помилки заносяться до журналу помилок.

1.2.5 Діаграма розгортання системи

Діаграма розгортання системи - це тип діаграми UML, який показує архітектуру виконання системи, включаючи такі вузли, як середовища виконання апаратного чи програмного забезпечення, та проміжне програмне забезпечення, що їх з'єднує.

Діаграми розгортання зазвичай використовуються для візуалізації фізичного обладнання та програмного забезпечення системи. За його допомогою ви можете зрозуміти, як система буде фізично розгорнута на апаратному забезпеченні.

Діаграми розгортання допомагають моделювати апаратну топологію системи порівняно з іншими типами діаграм UML, які в основному окреслюють логічні компоненти системи.

Коли використовувати діаграму розгортання?

- З якими існуючими системами потрібно буде щойно доданій системі взаємодіяти чи інтегруватися?

- Наскільки надійною повинна бути система (наприклад, надлишкове обладнання в разі збою системи)?
- Що і хто буде підключатись до системи або взаємодіяти з нею, і як вони це робитимуть
- Яке проміжне програмне забезпечення, включаючи операційну систему та підходи та протоколи зв'язку, буде використовувати система?
- З яким апаратним та програмним забезпеченням користувачі безпосередньо взаємодіятимуть (ПК, мережеві комп’ютери, браузери тощо)?
- Як ви будете контролювати систему після розгортання?
- Наскільки безпечною повинна бути система (потрібен брандмауер, фізично захищене обладнання тощо)?

Нижче наводиться приклад діаграми розгортання, яка повністю покриває потреби цієї роботи.

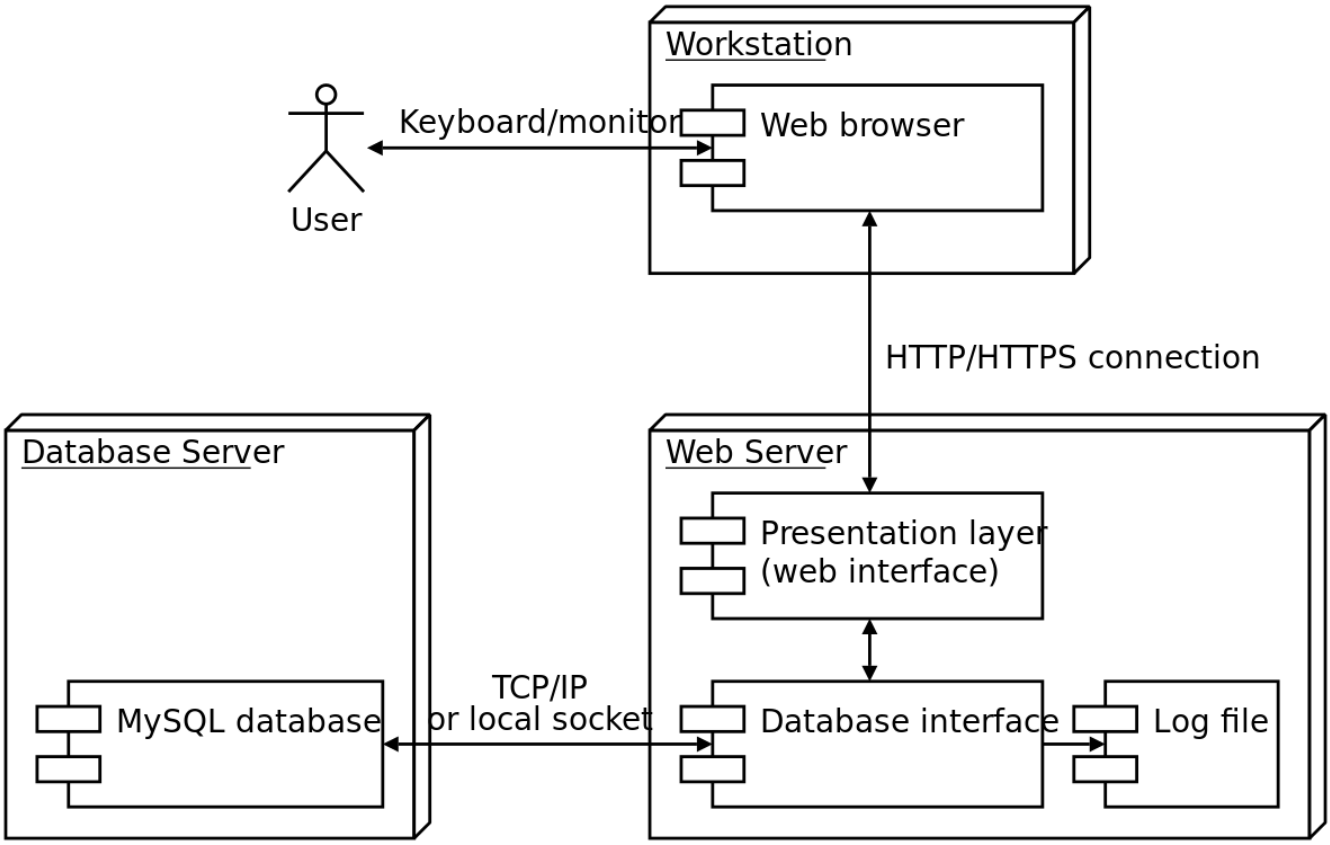

Рис. 2 Діаграма розгортання

Розділ 2 Проектування програмного забезпечення2.1 Обрані технології

Мова програмування C#. Для розробки даного застосунку було обрано стек ASP.NET Core 5.0 Server + VueJS SPA, а також середовище програмування Microsoft Visual Studio 2019. Для клієнтської частини додатково було обрано бібліотеку компонентів Vuetify.

2.1.1 C#

C # - це мова програмування загального призначення з багатьма парадигмами, що охоплює статичну типізацію, сильну типізацію, лексичну область дії, імперативну, декларативну, функціональну, загальну, об'єктно-орієнтовану (на основі класів) та дисципліни програмування, орієнтовану на компоненти.

C # був розроблений Microsoft в 2000 році в рамках його ініціативи .NET, а згодом затверджений як міжнародний стандарт Ecma (ECMA-334) у 2002 році та ISO (ISO / IEC 23270) у 2003 році. В даний час команду розробників очолює Мадс Торгерсен, яка є однією з мов програмування, розроблених для Спільної мовної інфраструктури (CLI). Остання версія - 9.0, випущена в 2020 році в .NET 5.0 і включена у Visual Studio 2019 версії 16.8.

Mono - це безкоштовний проект із відкритим кодом для розробки крос-платформного компілятора та середовища виконання (тобто віртуальної машини) для мови.

Стандарт Ecma містить цілі проектування для C #:

- Мова задумана як проста, сучасна, об'єктно-орієнтована мова програмування загального призначення.
- Мова та її реалізації повинні забезпечувати підтримку таких принципів програмної інженерії, як сильна перевірка типу, перевірка меж масиву,

виявлення спроб використання неініціалізованих змінних та автоматичний збір сміття. Важливі надійність програмного забезпечення, довговічність та продуктивність програміста.

- Мова призначена для використання при розробці програмних компонентів, придатних для розгортання в розподілених середовищах.
- Переносимість дуже важлива для вихідного коду та програмістів, особливо тих, хто вже знайомий з C та C ++.
- Підтримка інтернаціоналізації дуже важлива.
- C # призначений для придатності для написання додатків як для розміщених, так і для вбудованих систем, починаючи від дуже великих, які використовують складні операційні системи, і закінчуючи дуже малими, що мають спеціальні функції.
- Незважаючи на те, що програми C # призначені для економічного використання пам’яті та вимог до потужності обробки, мова не була призначена конкурувати безпосередньо за продуктивністю та розміром з мовою C або мовою збірки.

2.1.2 Microsoft Visual Studio

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Застосовується для розробки комп’ютерних програм, а також веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач вихідного рівня, так і налагоджувач машинного рівня. Інші

вбудовані інструменти включають в себе профайлер коду, дизайнер для побудови графічних інтерфейсів, веб-дизайнер, дизайнер класів та конструктор схем баз даних. Він приймає плагіни, які розширюють функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування джерелами (таких як Subversion і Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що належать до домену, або наборів інструментів для інших аспектів розробки програмного забезпечення. життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (різною мірою) майже будь-яку мову програмування, за умови, що існує спеціальна послуга. До вбудованих мов належать C, C ++, C ++ / CLI, Visual Basic .NET, C #, F #, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M, серед інших, доступна через плагіни. Раніше підтримувались Java (і J #).

2.1.3 ASP.NET Core

ASP.NET Core - це безкоштовний веб-фреймворк з відкритим кодом і наступник ASP.NET, [5] розроблений Microsoft. [6] Це модульний фреймворк, який працює як на повному .NET Framework, так і на Windows, а також на міжплатформенному .NET Core. Однак версія ASP.NET Core 3 працює лише на підтримці .NET Framework .NET Framework. [7]

Фреймворк - це повний рерайт, який об’єднує раніше окремі ASP.NET MVC та ASP.NET Web API в єдину модель програмування.

Незважаючи на те, що це новий фреймворк, побудований на новому веб-стеку, він має високий ступінь сумісності концепцій з ASP.NET. Програми ASP.NET Core підтримують паралельне керування версіями, в якому різні

програми, що працюють на одній машині, можуть націлювати різні версії ASP.NET Core. Це неможливо з попередніми версіями ASP.NET.

ASP.NET Core - це нова платформа з відкритим кодом та міжплатформна платформа для створення сучасних підключених до Інтернету хмарних додатків, таких як веб-програми, програми IoT та мобільні серверні мережі. Програми ASP.NET Core можуть працювати на .NET Core або на повній .NET Framework. Він був розроблений для забезпечення оптимізованої основи розробки програм, які розгортаються в хмарі або виконуються локально. Він складається з модульних компонентів з мінімальними накладними витратами, завдяки чому ви зберігаєте гнучкість при розробці своїх рішень. Ви можете розробляти та запускати крос-платформні програми ASP.NET Core на Windows, Mac та Linux. ASP.NET Core є відкритим кодом на GitHub.

Перший попередній випуск ASP.NET вийшов майже 15 років тому як частина .NET Framework. З тих пір мільйони розробників використовували його для створення та запуску чудових веб-програм, і протягом багатьох років ми додавали та розвивали багато можливостей до нього.

ASP.NET Core має ряд архітектурних змін, які призводять до набагато вишуканішої та модульної структури. ASP.NET Core більше не базується на System.Web.dll. Він базується на наборі детальних та добре продуманих пакетів NuGet. Це дозволяє оптимізувати додаток для включення лише необхідних пакунків NuGet. Переваги меншої площі додатків включають жорсткіший захист, зменшення обслуговування, покращену продуктивність та зменшення витрат у моделі оплати за те, що ви використовуєте.

За допомогою ASP.NET Core ви отримуєте такі основні покращення:

- Уніфікована історія для створення веб-інтерфейсу та веб-API

- Інтеграція сучасних фреймворків на стороні клієнта та робочих процесів розробки
- Готова до хмари система конфігурації, заснована на середовищі
- Вбудована ін’єкція залежностей
- Новий легкий та модульний конвеєр запитів HTTP
- Можливість розміщення на IIS або самостійного розміщення у власному процесі
- Побудований на .NET Core, який підтримує справжнє паралельне керування версіями програм
- Відправляється цілком як пакети NuGet
- Новий інструмент, який спрощує сучасну веб-розробку
- Створюйте та запускайте крос-платформні програми ASP.NET на Windows, Mac та Linux
- Відкритий код та орієнтованість на громаду

2.1.4 VueJS

Vue - це новий реактивний фреймворк для створення користувацьких інтерфейсів. На відміну від інших монолітів, цей створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Назва фреймворку - Vue - в англійській мові схожа фонетично із view, і вона посилає до традиційної архітектури Model-View-Controller (MVC). Іншими словами, view - це інтерфейс користувача системи, а Vue.js в основному формує рівень представлення. Але MVC не означає, що Vue.js не можна використовувати з

іншим архітектурним підходом, таким як архітектура на основі компонентів (CBA), що використовується в React.

У нинішньому вигляді Vue.js, розроблений Еваном Ю на постійній основі, отримував вигоду від внесків членів громади та фінансувався переважно Патреоном. Без фінансової підтримки таких підприємств, як Facebook (React) та Google (Angular), Vue все ж домігся широкого поширення на Github.

2.1.5 Vuetify

Vuetify - це повний фреймворк, побудований поверх Vue.js. Мета проекту - надати розробникам інструменти, необхідні для створення багатого та привабливого інтерфейсу для користувачів. На відміну від інших фреймворків, Vuetify розроблений з нуля, щоб бути простим у засвоєнні та корисним, сотні ретельно розроблених компонентів із специфікації Material Design.

Vuetify використовує підхід до дизайну «mobile first», що означає, що додаток працюватиме «з коробки» - будь то на телефоні, планшеті чи настільному комп’ютері.

Vuetify має дуже активний цикл розробки і оновлюється щотижня, реагуючи на проблеми спільноти та звіти з шаленою швидкістю, що дозволяє частіше застосовувати виправлення помилок та вдосконалення. Крім того, кожен великий випуск супроводжується 18-місячною довгостроковою підтримкою попередньої другорядної версії.

2.1.6 MVC

Модель-View-Controller (MVC) - це архітектурний шаблон, який розділяє додаток на три основні логічні компоненти: модель, вигляд та контролер. Кожен із цих компонентів створений для обробки конкретних аспектів розробки програми.

MVC - одна з найбільш часто використовуваних галузевих стандартів веб-розробки для створення масштабованих та розширюваних проектів.

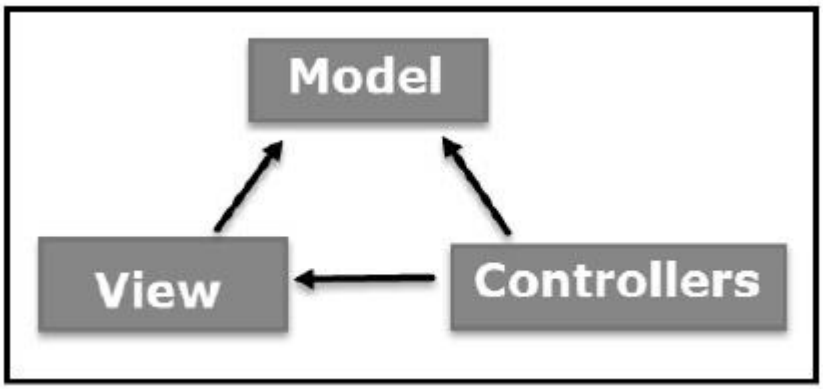

Рис. 3 Компоненти MVC

Модель. Компонент Model відповідає всій логіці, пов’язаній з даними, з якою працює користувач. Це може представляти як дані, що передаються між компонентами View та Controller, так і будь-які інші дані, пов’язані з бізнес-логікою. Наприклад, об'єкт "Клієнт" буде отримувати інформацію про клієнта з бази даних, маніпулювати нею та оновлювати дані назад у базу даних або використовувати її для візуалізації даних.

Вью. Компонент View використовується для всієї логіки інтерфейсу програми. Наприклад, подання Клієнт включатиме всі компоненти інтерфейсу, такі як текстові поля, випадаючі меню тощо, з якими взаємодіє кінцевий користувач.

Контролер. Контролери виступають інтерфейсом між компонентами Model і View для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Model та взаємодії з поданнями для надання кінцевого результату. Наприклад, контролер замовника буде обробляти всі взаємодії та входи з перегляду замовника та оновлювати базу даних за допомогою моделі замовника. Той самий контролер буде використовуватися для перегляду даних Клієнта.

ASP.NET підтримує три основні моделі розробки: веб-сторінки, веб-форми та MVC (контролер перегляду моделі). Структура ASP.NET MVC - це легкий, дуже перевірений фреймворк презентацій, інтегрований із існуючими функціями ASP.NET, такими як основні сторінки, автентифікація тощо. У рамках .NET ця структура визначена у збірці System.Web.Mvc. Остання версія MVC Framework - 5.0. Ми використовуємо Visual Studio для створення програм ASP.NET MVC, які можна додати як шаблон у Visual Studio.

Особливості ASP.NET MVC. ASP.NET MVC надає такі функції:

- Ідеально підходить для розробки складних, але легких додатків.
- Забезпечує розширювану та підключається структуру, яку можна легко замінити та налаштувати. Наприклад, якщо ви не хочете використовувати вбудований механізм перегляду Razor або ASPX, тоді ви можете використовувати будь-які інші сторонні механізми перегляду або навіть налаштувати існуючі.
- Використовує дизайн програми на основі компонентів, логічно поділяючи її на компоненти Model, View та Controller. Це дозволяє розробникам управляти складністю масштабних проектів і працювати над окремими компонентами.
- Структура MVC покращує тестовий розвиток та перевірку програми, оскільки всі компоненти можуть бути розроблені на основі інтерфейсу та перевірені за допомогою макетних об'єктів. Отже, ASP.NET MVC Framework ідеально підходить для проектів із великою командою веб-розробників.
- Підтримує всі існуючі широкі функціональні можливості ASP.NET, такі як авторизація та автентифікація, основні сторінки, прив'язка даних, елементи керування користувачами, членство, маршрутизація ASP.NET тощо.
- Не використовує концепцію стану перегляду (яка присутня в ASP.NET). Це допомагає створювати легкі програми та дає повний контроль розробникам.

Таким чином, можно розглядати MVC Framework як основний фреймворк, побудований поверх ASP.NET, що надає великий набір додаткових функціональних можливостей, орієнтованих на розробку та тестування на основі компонентів.

2.2 Архітектура системи

Архітектура системи є абстрактною, орієнтованою на концептуалізацію, глобальною та спрямованою на досягнення концепції місії та життєвого циклу системи. Вона також фокусується на структурі високого рівня в системах та елементах системи. У ній розглядаються архітектурні принципи, концепції, властивості та характеристики системи інтересів. Архітектура системи також може застосовуватися до більш ніж однієї системи, в деяких випадках формуючи загальну структуру, патерн та набір вимог до класів або сімейств подібних або споріднених систем.

Мета діяльності архітектури системи полягає у визначенні комплексного рішення, заснованого на принципах, концепціях та властивостях, що логічно пов'язані та узгоджуються між собою. Архітектура рішення має особливості, властивості та характеристики, які, наскільки це можливо, задовольняють проблему чи можливість, висловлену набором системних вимог (простежуваних до місії / бізнесу та вимог зацікавлених сторін) та концепцій життєвого циклу (наприклад, експлуатаційні, підтримка) і які можна реалізувати за допомогою технологій (наприклад, механіка, електроніка, гідравліка, програмне забезпечення, послуги, процедури, діяльність людини).

Архітектура системи описує її основні компоненти, їх взаємозв'язки (структури) та те, як вони взаємодіють між собою. Архітектура та дизайн програмного забезпечення включає кілька таких факторів, як бізнес-стратегія, показники якості, людська динаміка, дизайн та IT-середовище.

Ми можемо розділити архітектуру та дизайн програмного забезпечення на дві різні фази: архітектуру програмного забезпечення та дизайн програмного забезпечення. В архітектурі нефункціональні рішення визначаються та розділяються функціональними вимогами. У дизайні виконуються функціональні вимоги.

Архітектура програмного забезпечення.

Архітектура служить проектом системи. Він забезпечує абстракцію для управління складністю системи та встановлення механізму зв'язку та координації між компонентами.

- Він визначає структуроване рішення, яке відповідає всім технічним та експлуатаційним вимогам, одночасно оптимізуючи загальні атрибути якості, такі як продуктивність та безпека.
- Далі, це передбачає набір важливих рішень щодо організації, пов’язаних із розробкою програмного забезпечення, і кожне з цих рішень може мати значний вплив на якість, ремонтопридатність, продуктивність та загальний успіх кінцевого продукту. Ці рішення складаються з:
    - Вибір структурних елементів та їх інтерфейсів, за якими складається система.
    - Поведінка, як зазначено у співпраці серед цих елементів.
    - Склад цих структурних та поведінкових елементів у велику підсистему.
    - Архітектурні рішення відповідають цілям бізнесу.
    - Архітектурні стилі керують організацією.

Розробка програмного забезпечення.

Розробка програмного забезпечення передбачає проектний план, який описує елементи системи, їх відповідність та спільну роботу для задоволення вимог системи. Цілі складання плану проекту наступні:

- Для узгодження системних вимог та встановлення очікувань із замовниками, маркетингом та управлінським персоналом.
- Діяти як проект в процесі розробки.
- Керувати завданнями впровадження, включаючи детальний дизайн, кодування, інтеграцію та тестування.

Він постає перед детальним проектуванням, кодуванням, інтеграцією та тестуванням, а також після аналізу доменів, аналізу вимог та аналізу ризиків.

Верхньорівнево створена у цій роботі система складається з наступних частин:

- OnlineCalendar WebApi – серверна частина, яка відповідає за читання даних з бази, обробку цих даних, бізнес логіку, обробку помилок, підготування даних для клієнтської частини.

- OnlineCalendar SPA – клієнтська частина, яка відповідає за інтерфейс користувача, несе у собі клієнтську логіку show/hide та required полів.

- OnlineCalendar Database – основна база даних з якою працює додаток.

- External Database – надана міністерством база даних з архівними даними.

- OnlineCalendar Database Migartion Tool – консольний додаток для міграції даних з архівної до основної бази даних.

2.2.1 UML-діаграма

UML - це абревіатура, що розшифровується як Unified Modeling Language. Простіше кажучи, UML - це сучасний підхід до моделювання та документування програмного забезпечення. Насправді це один з найпопулярніших методів моделювання бізнес-процесів.

Він базується на схематичному зображенні програмних компонентів. Як говорить старе прислів’я: «картинка коштує тисячі слів». Використовуючи візуальні подання, ми можемо краще зрозуміти можливі недоліки або помилки програмного забезпечення чи бізнес-процесів.

Існує 14 типів діаграм UML, які допоможуть вам змоделювати поведінку додатку:

- Structure Diagrams
    - Class Diagram (Діаграма класів)
    - Component Diagram (Діаграма компонентів)
    - Deployment Diagram (Діаграма розгортання)
    - Object Diagram (Діаграма об’єктів)
    - Package Diagram (Діаграма пакетів)
    - Profile Diagram (Профільна діаграма)
    - Composite Structure Diagram (Діаграма композитної структури)
- Behavioral Diagrams
    - Use Case Diagram (Діаграма варіантів використання)
    - Activity Diagram (Діаграма діяльності)
    - State Machine Diagram (Діаграма станів автомата)
    - Sequence Diagram (Діаграма послідовності)
    - Communication Diagram (Діаграма комунікації)
    - Interaction Overview Diagram (Діаграма взаємодії)
    - Timing Diagram (Діаграма часу)

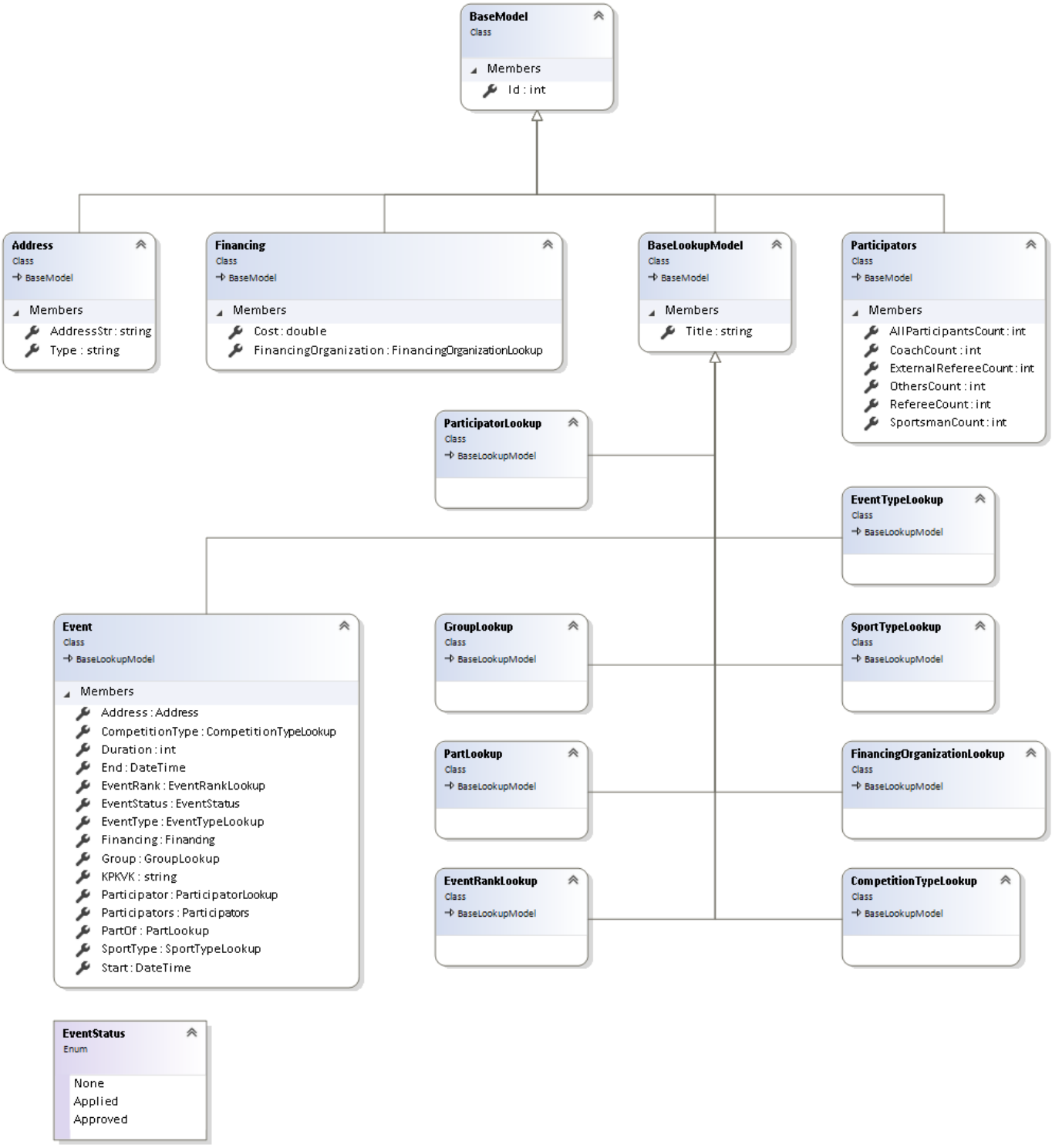

Рис. 4 UML-діаграма класів

2.2.2 Діаграма послідовності

Діаграми послідовності UML - це діаграми взаємодії, які детально описують спосіб виконання операцій. Вони фіксують взаємодію між об’єктами в контексті співпраці. Діаграми послідовності - це фокус на часі, і вони наочно показують порядок взаємодії, використовуючи вертикальну вісь діаграми, щоб представити час, які повідомлення надсилаються та коли.

Діаграми послідовності фіксують:

- Взаємодії, які відбуваються у співпраці, яка реалізує варіант використання або операцію (діаграми екземплярів або загальні діаграми)
- взаємодії на високому рівні між користувачем системи та системою, між системою та іншими системами або між підсистемами (іноді їх називають діаграмами системних послідовностей)

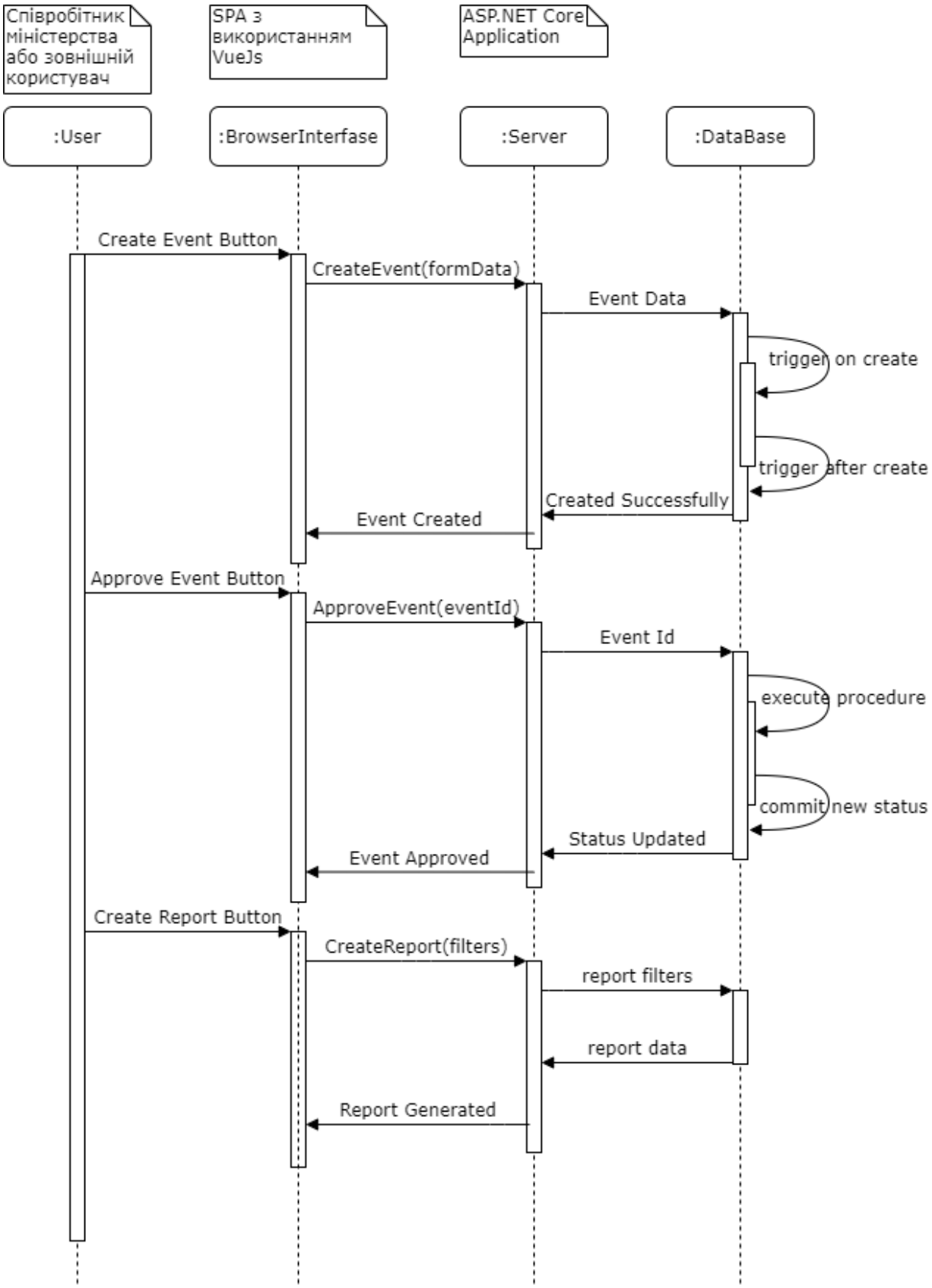

Рис. 5 UML-діаграма послідовності

2.2.3 База даних

База даних була надана міністерством молоді та спорту. Однак в ході виконання роботи прийнято рішення створити нову базу даних на основі практичного досвіду з використанням нормалізації баз даних.

Нормалізація бази даних - це техніка організації даних у базі даних. Нормалізація - це систематичний підхід до розкладання таблиць для усунення надмірності даних (повторення) та небажаних характеристик, таких як аномалії вставки, оновлення та видалення. Це багатоступеневий процес, який переводить дані в табличну форму, видаляючи дубльовані дані з таблиць відношень.

Нормалізація використовується в основному для двох цілей:

- Видалення зайвих (марних) даних.
- Забезпечення сенсу залежності даних, тобто дані логічно зберігаються.

Правила нормалізації поділяються на такі нормальні форми:

- Перша нормальна форма
- Друга нормальна форма
- Третя нормальна форма
- НФБК (Нормальна форма Бойса-Кодда)
- Четверта нормальна форма

Перша нормальна форма. Якщо відношення містить складений або багатозначний атрибут, воно порушує першу нормальну форму або відношення знаходиться в першій нормальній формі, якщо воно не містить жодного складеного або багатозначного атрибута. Відношення знаходиться в першій нормальній формі, якщо кожен атрибут у цьому відношенні виділяється значенням атрибута.

Друга нормальна форма. Щоб мати відношення в другій нормальній формі, відношення має бути у першій нормальній формі, і відношення не повинно містити

часткової залежності. Відношення є в 2НФ, якщо воно не має часткової залежності, тобто жоден непростий атрибут (атрибути, які не є частиною будь-якого ключа-кандидата) не залежить від будь-якої належної підмножини будь-якого ключа-кандидата таблиці.

Третя нормальна форма. Відношення знаходиться в третій нормальній формі, якщо немає перехідної залежності для непростих атрибутів, а також знаходиться у другій нормальній формі.

Співвідношення є у 3НФ, якщо принаймні одна з наступних умов виконується в кожній нетривіальній залежності функції X -> Y

- X - це супер ключ.
- Y - основний атрибут (кожен елемент Y є частиною якогось ключа-кандидата).

Далі немає сенсу розглядати нормальні форми, оскільки перших трьох зазвичай достатньо для створення якісної БД. Існує термін денормалізації баз даних, це навмисне приведення структури бази даних в стан, що не відповідає критеріям нормалізації, який зазвичай проводиться з метою прискорення операцій читання з бази за рахунок додавання надлишкових даних. Через велику кількість запитів до великих таблиць надмірна нормалізація призводить до проблем читання пов’язаних із часом виконання запиту.

2.3 Патерни дизайну

Патерни проектування є типовими рішеннями найпоширеніших проблем при розробці програмного забезпечення. Вони схожі на заздалегідь зроблені креслення, які ви можете налаштувати для вирішення постійної проблеми дизайну у своєму коді.

Ви не можете просто знайти патерн і скопіювати його у свою програму так, як це можна зробити за допомогою готових функцій або бібліотек. Патерн - це не конкретна частина коду, а загальна концепція вирішення певної проблеми. Ви можете стежити за деталями патерну та впроваджувати рішення, яке відповідає реаліям вашої власної програми.

Патерни часто плутають з алгоритмами, оскільки обидві концепції описують типові рішення деяких відомих проблем. Хоча алгоритм завжди визначає чіткий набір дій, які можуть досягти певної мети, патерн є більш високим рівнем опису рішення. Код одного і того ж , застосований до двох різних програм, може бути різним.

Аналогією з алгоритмом є рецепт приготування: обидва мають чіткі кроки для досягнення мети. З іншого боку, патерн більше нагадує план: ви можете бачити, який результат і його особливості, але точний порядок реалізації залежить від вас.

Більшість зразків описано дуже формально, тому люди можуть відтворювати їх у багатьох контекстах. Ось розділи, які зазвичай присутні в описі патерну:

- Намір патерну коротко описує як проблему, так і спосіб її вирішення.
- Мотивація додатково пояснює проблему та рішення, яке модель дає можливість.
- Структура класів показує кожну частину патерну та їх взаємозв'язок.
- Приклад коду в одній з популярних мов програмування полегшує розуміння ідеї патерну.

Деякі каталоги патернів містять інші корисні деталі, такі як застосовність патерну, кроки реалізації та взаємозв'язок з іншими патернами.

Патерни дизайну відрізняються своєю складністю, рівнем деталізації та масштабом застосовності до всієї проектуваної системи. Мені подобається аналогія

з будівництвом доріг: ви можете зробити перехрестя безпечнішим, встановивши кілька світлофорів або побудувавши цілу багаторівневу розв'язку з підземними переходами для пішоходів.

Найпростіші та низькорівневі схеми часто називають ідіомами. Зазвичай вони застосовуються лише до однієї мови програмування.

Найбільш універсальними та високорівневими патернами є архітектурні. Розробники можуть застосувати ці патерни практично на будь-якій мові. На відміну від інших зразків, їх можна використовувати для проектування архітектури цілого додатка.

Крім того, усі схеми можуть бути класифіковані за їхнім наміром або призначенням. Ця книга охоплює три основні групи зразків:

- Породжувальні патерни забезпечують механізми створення об’єктів, що підвищують гнучкість та повторне використання існуючого коду.
- Структурні патерни пояснюють, як збирати об’єкти та класи у великі структури, зберігаючи при цьому гнучкість та ефективність структур.
- Патерни поведінки дбають про ефективне спілкування та розподіл обов’язків між об’єктами.

2.3.1 Породжувальні патерни

У програмній інженерії креативні патерни дизайну - це патерни дизайну, які мають справу з механізмами створення об’єктів, намагаючись створити об’єкти у спосіб, відповідний ситуації. Основна форма створення об’єкта може спричинити проблеми з дизайном або ускладнити дизайн. Породжувальні патерни дизайну вирішують цю проблему, так чи інакше керуючи створенням цього об’єкта.

- Абстрактна фабрика (Abstract Factory) – Створює екземпляр декількох сімей класів

- Будівельник (Builder) – Відокремлює конструкцію об’єкта від його подання
- Фабричний метод (Factory Method) – Створює екземпляр декількох похідних класів
- Пул об’єктів (Object Pool) – Уникайте дорогого придбання та вивільнення ресурсів шляхом переробки об’єктів, які більше не використовуються
- Прототип (Prototype) – Повністю ініціалізований екземпляр, який потрібно скопіювати або клонувати
- Одинак (Singleton) – Клас, в якому може існувати лише один екземпляр

2.3.2 Структурні патерни

У програмній інженерії структурні патерни проектування - це патерни дизайну, які полегшують дизайн, визначаючи простий спосіб реалізації взаємозв’язків між сутностями.

- Адаптер (Adapter) – Інтерфейси відповідності різних класів
- Міст (Bridge) – Відокремлює інтерфейс об'єкта від його реалізації
- Компонувальник (Composite) – Деревоподібна структура простих та складених об’єктів
- Декоратор (Decorator) – Динамічно додавати відповідальність за об’єкти
- Фасад (Facade) – Один клас, що представляє цілу підсистему
- Легковаговик (Flyweight) – Дрібнозернистий екземпляр, який використовується для ефективного обміну
- Дані приватного класу (Private Class Data) – Обмежує доступ до ацесора/мутатора
- Замісник (Proxy) – Об'єкт, що представляє інший об'єкт

2.3.3 Поведінкові патерни

У програмній інженерії поведінкові шаблони проектування - це шаблони проектування, які визначають загальні моделі спілкування між об’єктами та реалізують ці закономірності. Роблячи це, ці моделі збільшують гнучкість у здійсненні цього спілкування.

- Ланцюжок обов’язків (Chain of responsibility) – Спосіб передачі запиту між ланцюжком об’єктів
- Команда (Command) – Інкапсулювати запит на команду як об’єкт
- Інтерпретатор (Interpreter) – Спосіб включення мовних елементів у програму
- Ітератор (Iterator) – Послідовний доступ до елементів колекції
- Посередник (Mediator) – Визначає спрощене спілкування між класами
- Знімок (Memento) – Захоплення та відновлення внутрішнього стану об'єкта
- Нульовий об'єкт (Null Object) – Призначений для дії як значення об’єкта за замовчуванням
- Спостерігач (Observer) – Спосіб повідомлення про зміни у ряді класів
- Стан (State) – Змінити поведінку об’єкта, коли його стан змінюється
- Стратегія (Strategy) – Інкапсулює алгоритм всередині класу
- Шаблонний метод (Template method) – Відкладіть точні кроки алгоритму до підкласу
- Відвідувач (Visitor) – Визначає нову операцію для класу без змін

2.4 ООП

Об'єктно-орієнтоване програмування (ООП) - це модель комп'ютерного програмування, яка організовує розробку програмного забезпечення навколо даних

або об'єктів, а не функцій та логіки. Об’єкт можна визначити як поле даних, яке має унікальні атрибути та поведінку.

ООП фокусується на об’єктах, якими розробники хочуть маніпулювати, а не на логіці, необхідній для маніпулювання ними. Цей підхід до програмування добре підходить для великих, складних програм, що активно оновлюються чи підтримуються.

Організація об'єктно-орієнтованої програми також робить метод корисним для спільного розвитку, коли проекти поділяються на групи.

До додаткових переваг ООП належать багаторазове використання коду, масштабованість та ефективність. Навіть використовуючи мікросервіси, розробники повинні продовжувати застосовувати принципи ООП.

Першим кроком в ООП є збір усіх об’єктів, якими програміст хоче маніпулювати, та визначення їх співвідношення між собою - вправа, часто відома як моделювання даних.

Приклади об’єкта можуть варіюватися від фізичних сутностей, таких як людина, яка описується такими властивостями, як ім’я та адреса, аж до невеликих комп’ютерних програм, таких як віджети.

Після того, як об'єкт відомий, він позначається класом об'єктів, який визначає тип даних, які він містить, і будь-які логічні послідовності, які можуть ним керувати. Кожна окрема логічна послідовність відома як метод. Об'єкти можуть взаємодіяти з чітко визначеними інтерфейсами, які називаються повідомленнями.

2.4.1 Принципи ООП

Об'єктно-орієнтоване програмування базується на таких принципах:

- Інкапсуляція. Реалізація та стан кожного об'єкта приватно зберігаються у визначеній межі або класі. Інші об'єкти не мають доступу до цього класу

або повноважень вносити зміни, а лише можуть викликати список загальнодоступних функцій або методів. Ця характеристика приховування даних забезпечує більшу безпеку програми та дозволяє уникнути ненавмисного пошкодження даних.

- Абстракція. Об'єкти розкривають лише внутрішні механізми, які мають значення для використання інших об'єктів, приховуючи будь-який непотрібний код реалізації. Ця концепція допомагає розробникам легше вносити зміни та доповнення з часом.
- Наслідування. Можуть бути призначені взаємозв'язки та підкласи між об'єктами, що дозволяє розробникам повторно використовувати загальну логіку, зберігаючи при цьому унікальну ієрархію. Ця властивість ООП забезпечує більш ретельний аналіз даних, скорочує час розробки та забезпечує більш високий рівень точності.
- Поліморфізм. Об'єкти можуть приймати більше однієї форми залежно від контексту. Програма визначить, яке значення або використання необхідно для кожного виконання цього об'єкта, скорочуючи необхідність дублювання коду.

2.4.2 Критика ООП

Модель об’єктно-орієнтованого програмування критикується розробниками з багатьох причин. Найбільше занепокоєння викликає те, що ООП надмірно підкреслює компонент даних розробки програмного забезпечення і недостатньо зосереджується на обчисленнях або алгоритмах. Крім того, код ООП може бути складнішим для написання та складанням часу довшим.

Альтернативні методи ООП включають:

- функціональне програмування
- структуроване програмування

- імперативне програмування

Найбільш просунуті мови програмування дають розробникам можливість поєднувати ці моделі.

Популярні мови, що підтримують ООП:

- Java.

  Без сумніву, Java є однією з найбільш широко використовуваних мов ООП на ринку. Частина причин широкого використання Java полягає в тому, що це одна з офіційних мов для розробки Android. Оскільки Android - найпопулярніша мобільна операційна система на планеті, Java ще деякий час не буде збита з цього першого місця.

  Java була створена як мова "напиши один раз, запусти будь-де", що змушує її працювати для багатьох програм. І якщо Java на 100 відсотків на базі концепцій та принципів, що лежать в основі ООП, зрозуміло, чому вона живе вгорі цього списку.

- Python.

  Однією з найкращих речей Python є те, що це мова загального призначення, яка може застосовуватися у багатьох випадках використання. Однак одна з причин, чому саме ця об’єктно-орієнтована мова включена вгорі цього списку, полягає в тому, що це одна з найважливіших мов для машинного навчання та науки про дані. Швидше за все, ви не знайдете мови, яка найкраще підходить для цих 2 цілей. І як ML, так і DS продовжують розширювати своє значення в технологіях, Python буде лише продовжувати рости більш популярним.

- C++.

  C ++ - одна з небагатьох мов, що використовується для побудови компіляторів та інтерпретаторів, які компілюють інші мови програмування. C ++ включає швидкість C, з додаванням основних

концепцій ООП, що робить його швидким та гнучким. Однією з основних причин, чому C ++ настільки популярний, є те, що всі операційні системи написані з комбінацією C і C ++. Тож без C ++ не було б жодної операційної системи, з якою можна було б користуватися C ++. Іншими словами, C ++ є ключовою мовою для технологій. C ++ також використовується для розробки веб-браузерів (таких як Chrome, Firefox та Safari).

- C#.

  C # - ще одна мова ООП загального призначення. Ця мова була розроблена корпорацією Майкрософт ще в 2000 році в рамках ініціативи .NET і має певну схожість з Java, C та C ++. C # також був розроблений таким чином, що ним можуть користуватися інші мови. Одне з основних застосувань C # - для настільних комп'ютерів та веб-додатків, що робить його досить популярним варіантом для розробників, які хочуть створювати програмне забезпечення з графічним інтерфейсом.

  Однак одне з найпопулярніших застосувань C # - в ігровій індустрії. Причиною цього є те, що C # легко інтегрується з Windows. Оскільки C # включає автоматичне збір сміття, має видатні інструменти для розробників і є мовою ООП, він і надалі буде домінувати в секторі ігрового дизайну.

- Ruby.

  Ruby схожий на Python тим, що це мова загального призначення. Створений спеціально для простоти, повноти, розширюваності та портативності, Ruby є інтерпретованою мовою ООП з відкритим кодом, яку можна використовувати на численних платформах.

Ruby використовує дуже простий синтаксис, і його легко вивчити кожен, хто працював із сучасною мовою програмування. Він має простоту у використанні Perl, але додає повну сукупність концепцій ООП у суміш. Ruby часто використовується як для фронту, так і для розробки, і його можна застосовувати до областей аналізу даних, створення прототипів та перевірки концепцій.

Розділ 3 Використання програмного забезпечення

3.1 Інструкція розгортання системи

Розгортання системи складається з двох блоків:

- Інсталяція
    - Microsoft Visual Studio (для розробників) - IDE, що використовується для розробки комп'ютерних програм, веб-застосунків, веб-сервісів та ін.;
    - Microsoft SQL Server - система управління реляційними базами даних;
    - SQL Server Management Studio - програма для налаштування, управління і адміністрування усіх компонентів Microsoft SQL Server;
    - Internet Information Services (IIS) - веб-сервер, створений Microsoft для використання на Windows. Для інсталяції треба включити його до компонентів віндовс, перейшовши на вікно "Turn Windows features on or off" і обравши там Internet Information Services;
    - Microsoft .NET Core 2.1.0 - Windows Server Runtime & Hosting Bundle - включає в себе ресурси, необхідні для виконання .NET Core застосунків;
    - Microsoft .NET Core Software Development Kit - включає в себе все необхідне для створення та виконання .NET Core застосунків.
- Налаштування і підготовка до роботи. Цей пункт слід робити, беручи до уваги опис системи
    - Розвернути бази даних (через Visual Studio зробити публікацію проектів .Database або розвернути з наданих файлів-бекапів баз даних);
    - Зробити Publish проекту (лічильника чи реєстратора) до бажаних папок;
    - За допомогою IIS, сайту localhost (присутній по замовчуванню) додати сайт, і вказати папку, використану у попередньому пункті, а також

створити власний Application Pool для того, щоб проект виконувався у своєму процесі;

- Дати право створеному Application Pool на папку з проектом. Для цього треба перейти на вкладку Security в налаштування папки і додати користувача IIS AppPool*назва пулу* з відповідними правами;
- Дати право створеному Application Pool на базу даних. Для цього в SQL Server Management Studio треба відкрити сервер, перейти на вкладку Security, далі обрати New Login в контекстному меню, де вказати логін користувача IIS_AppPool*назва пулу*;
- Перевірити файли-налаштування (файли з розширенням .config) проекту;
- При необхідності додати SQL Server Agent Job для оновлення статистики. Це можна зробити через SQL Server Management Studio на вкладці SQL Server Agent, обравши в контекстному меню пункт New Job;
- Перевірити, чи працює проект. При необхідності повторити всі кроки.
- Інші умови, бажані до виконання:
  - Налаштування HTTPS доступу до обох серверів;

3.2 Інструкція користувача

У системі є дві основні ролі з різними функціями:

1. Співробітник міністерства – Головний користувач цієї системи. Має права на перегляд усіх подій зі свого департаменту. Має права на редагування цих подій та створення звітів. Має права на підтвердження зовнішній подій.
2. Зовнішній користувач – Другорядна роль. Має право на створення подій-заявок. Має право на перегляд та редагування створених подій-заявок, якщо вони ще не були підтверджені Співробітником міністерства.

Загалом робота програми виглядає так:

1. Співробітнику міністерства доступна сторінка «Календар подій». На сторінці доступний перегляд усіх подій з його департаменту, можливість відкрити сторінку редагування події, а також створення нової, можливість створити звіт та переглянути події-заявки

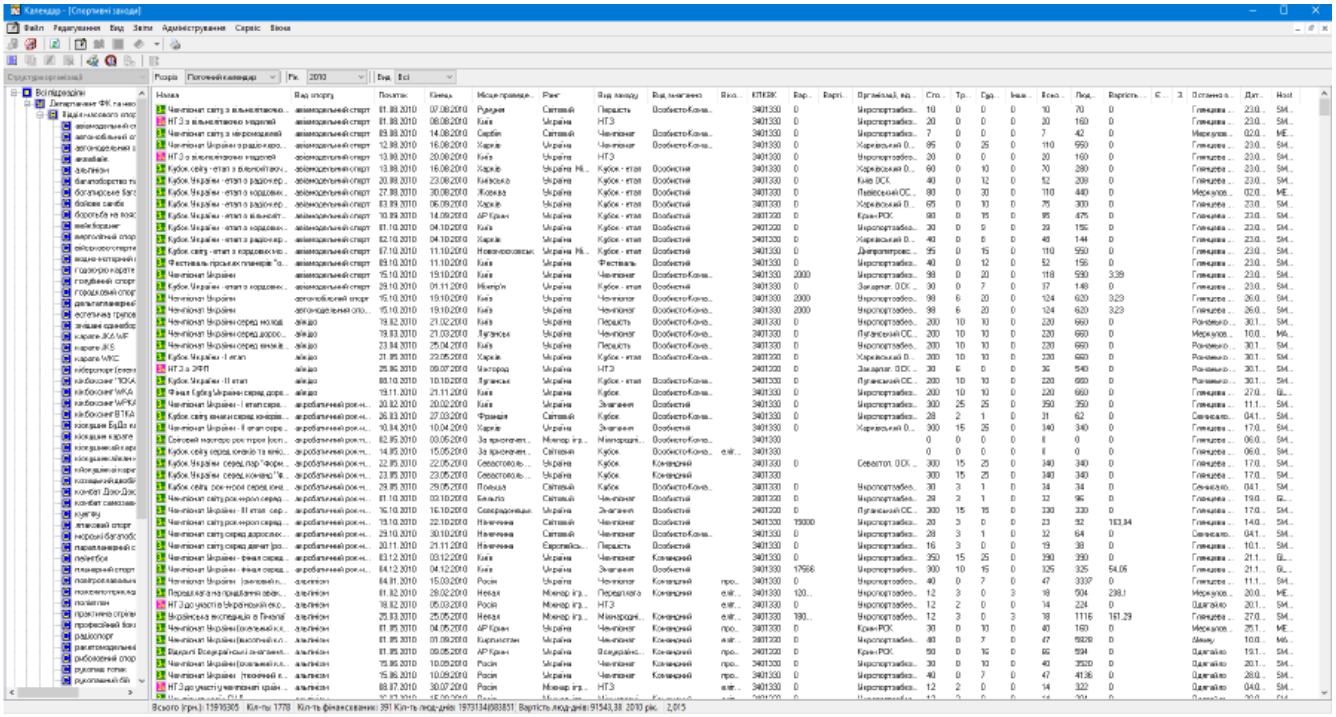

Рис. 6 Головна сторінка додатку

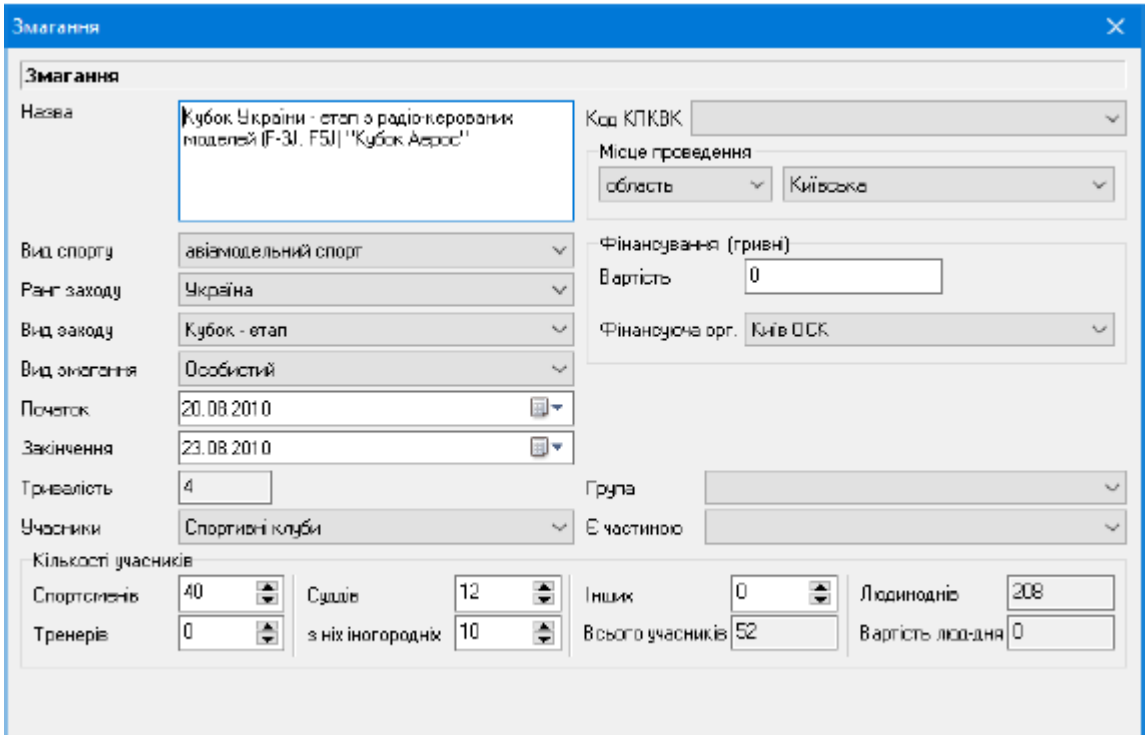

Рис. 7 Картка створення та редагування події

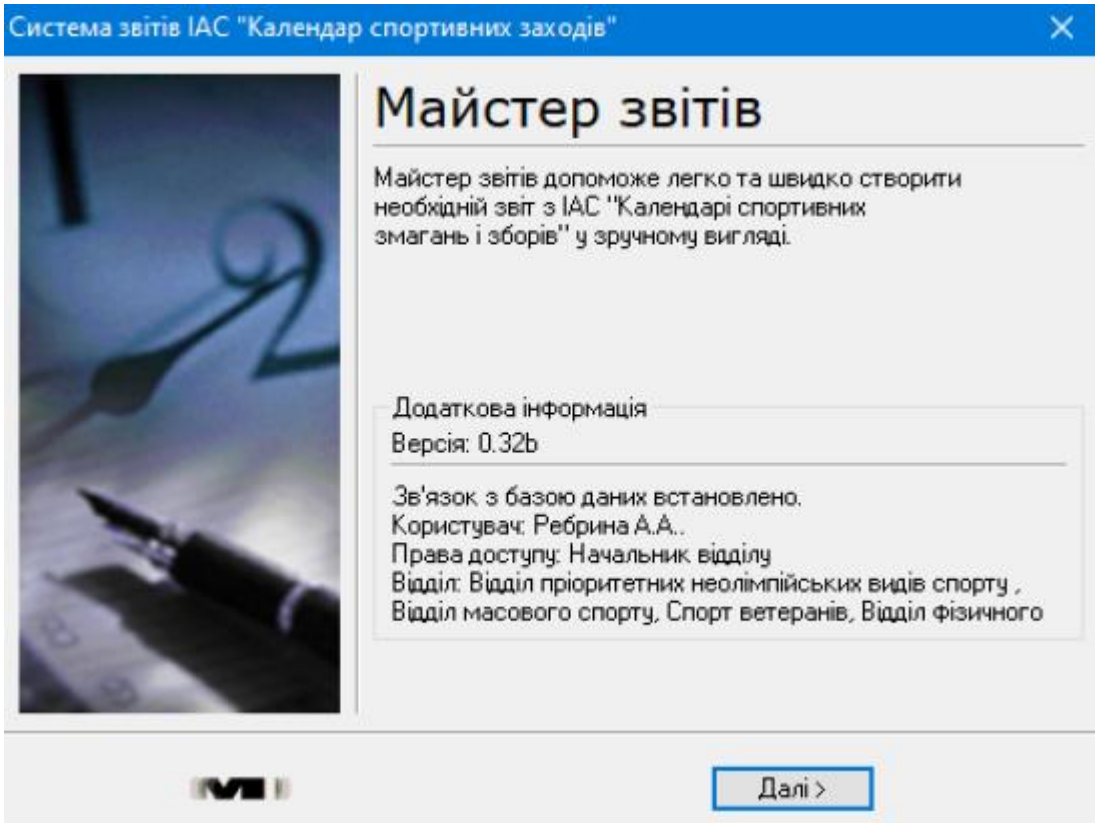

Рис. 8 Сторінка створення звіту (1)

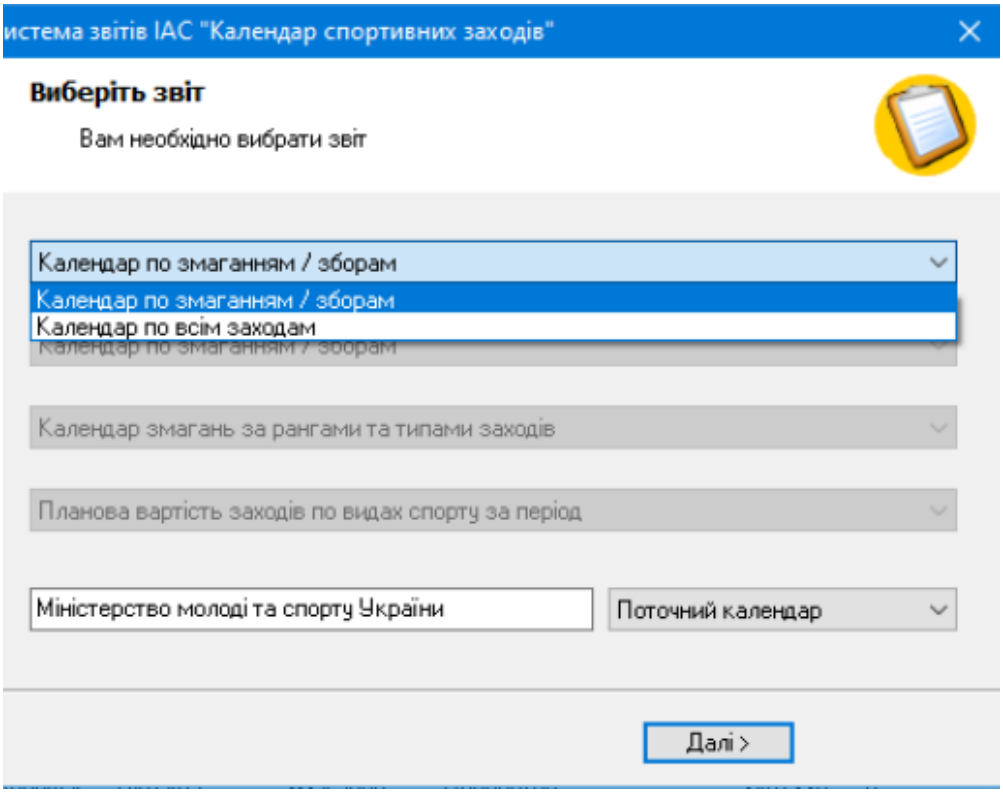

Рис. 9 Сторінка створення звіту (2)

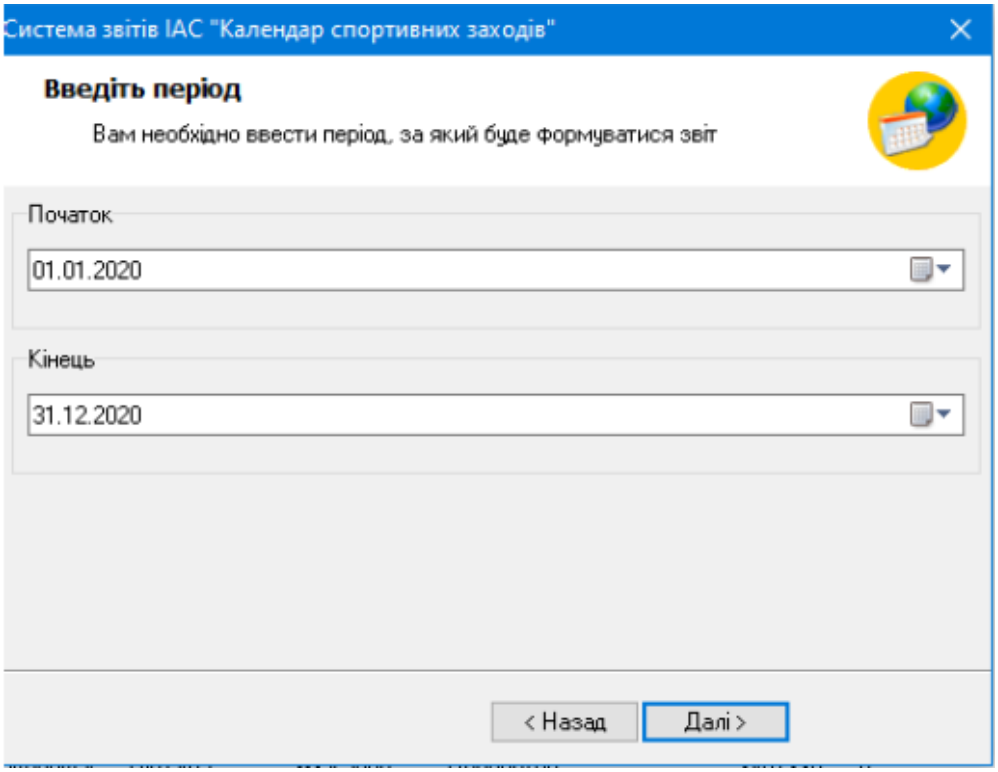

Рис. 10 Сторінка створення звіту (3)

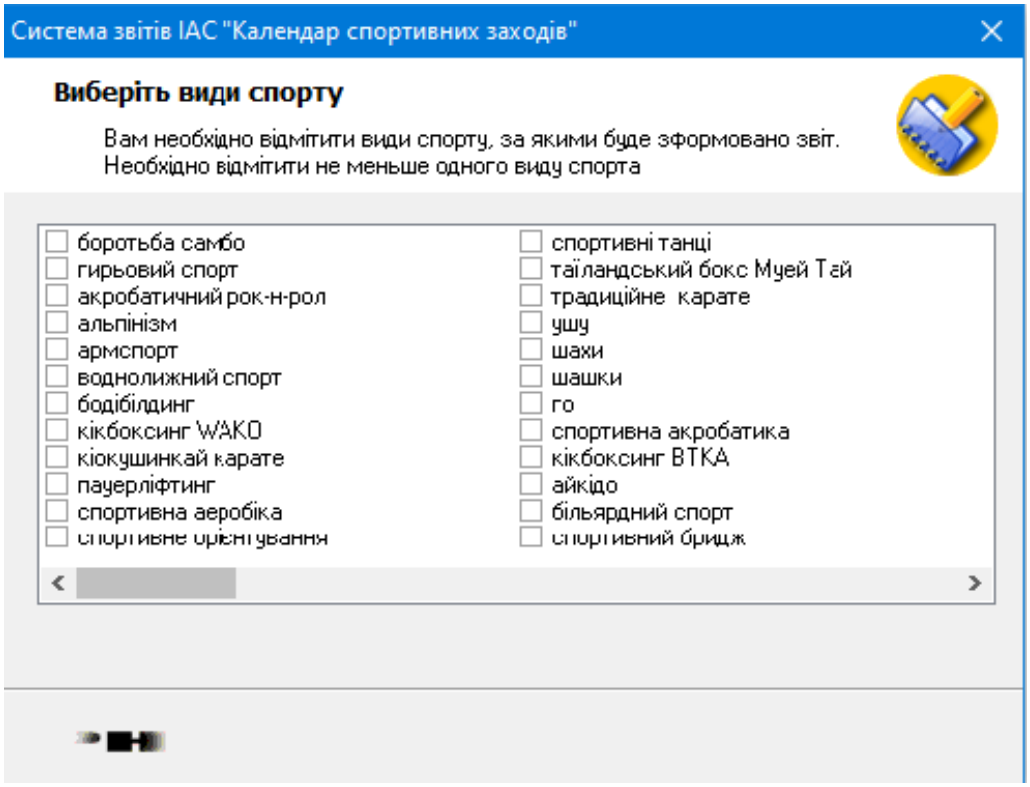

Рис. 11 Сторінка створення звіту (4)

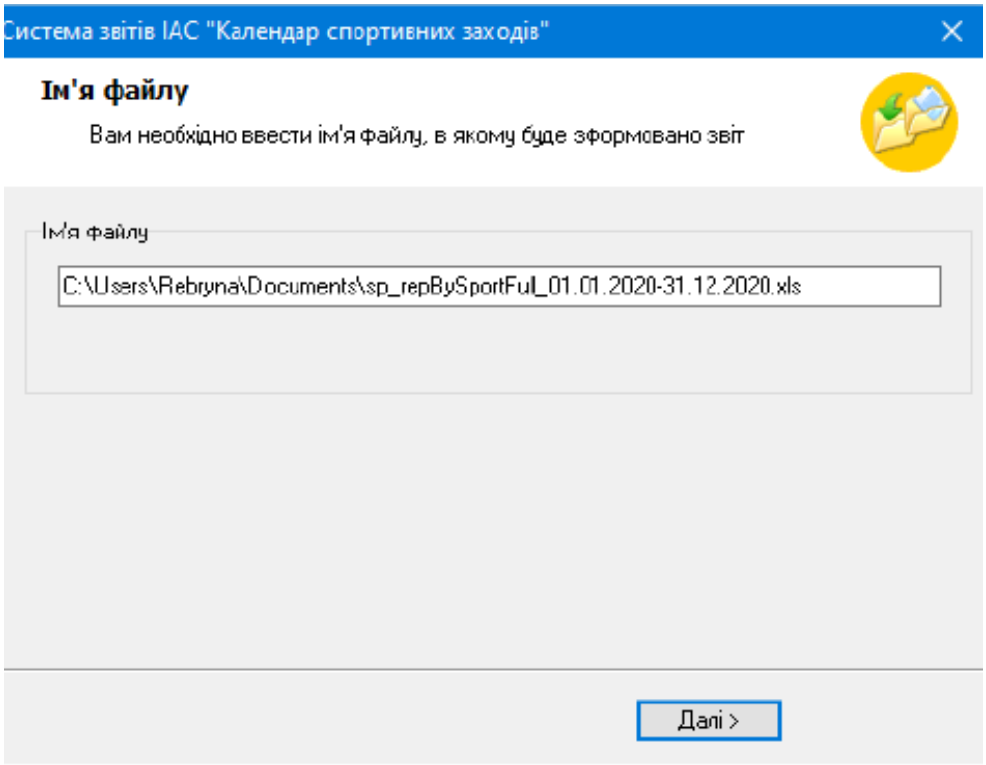

Рис. 12 Сторінка створення звіту (5)

2. Зовнішньому користувачу також, як і співробітнику міністерства доступні календар з подіями і картка подій, проте він у змозі бачити лише власно-створені події-заявки та редагувати лише їх, якщо співробітник міністерства ще не підтвердив цю подію-заявку. Також зовнішньому користувачу не доступні сторінки створення звіту.

Висновки

У процесі виконання цієї роботи було:

- Проведено аналіз і актуалізація вимог міністерства. Налагоджені зв’язки з кінцевим користувачем для розробки якісного додатку.
- Спроектовано нову базу даних з використанням отриманого досвіду та теоретичних знань здобутих у вищому навчальному закладі.
- Спроектовано та реалізовано додаток, який покриває усі необхідні юз-кейси працівника міністерства молоді і спорту, а також дає можливість зовнішнім користувачам створювати заявки на події для подальшого їх розгляду.
- Спроектовано та реалізовано додаток, який дозволяє мігрувати дані з архівної бази міністерства до нової бази додатку.

Результати даної роботи разом з інструкціями розгортання та користувача були передані до міністерства для подальшої інсталяції на використання.

Підводячи підсумок варто зазначити, що усі розроблені додатки, бази даних та документація виконані на високому рівні, завдяки знанням отриманим при теоретичному та практичному навчанні на кафедрі програмних систем і технологій, проходженні практик на підприємствах від університету та корпоративному досвіду від зовнішніх компаній.

Список використаних джерел

1. Javascript | MDN Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

2. Daniel Roth, Rick Anderson, and Shaun Luttin – Introduction To ASP.NET Core, 07.04.2019.

3. Станислав Протасевич - HTTP: протокол, который каждый разработчик должен знать (часть 1) – 13.05.2013 [Електронний ресурс]. – Режим доступу до ресурсу: https://ruseller.com/lessons.php? rub=28&id=1726.

4. Martin Gibbs – What is Microsoft SQL Server? [Електронний ресурс]. – Режим доступу до ресурсу: https://study.com/academy/lesson/what-is-microsoft-sql-server.html.

5. Steve Smith, Scott Addie, and Luke Latham – Dependency injection in ASP.NET Core – 07.04.2019 [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/depen dency-injection?view=aspnetcore-2.2.

6. Metanit – ASP.NET Core | Контроллеры – 10.06.2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/5.1.php.

7. Metanit – ASP.NET Core | Представления – 10.06.2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/7.1.php.

8. Database projects and data-tier applications – Visual Studio | Microsoft Docs – 21.11.2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/data-tools/creating-and-managing-databases-and-data-tier-applications-in-visual-studio?view=vs-2019.

9. Metanit – ASP.NET Core | Конвейер обработки запроса и middleware – 09.06.2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/2.2.php.

10. Создание файлов ресурсов для приложений .NET | Microsoft Docs – 30.03.2017 [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/framework/resources/creating-resource-files-for-desktop-apps.