ЗМІСТ

ВСТУП

Швидкий розвиток ІТ у світі зумовлює ріст популярності впровадження інформаційних систем у різні сфери діяльності людини. Навчання у ВНЗ не є винятком, а навіть навпаки, це чи не найголовніший і найцікавіший об’єкт для подібних впроваджень. В НТУУ «КПІ» вже ведеться розробка автоматизованої інформаційної системи «Електронний Кампус».

«Електронний Кампус» включає в себе багато різних аспектів навчальної інформаційної системи у вигляді окремих підсистем. Таким чином, тут реалізовані підсистеми дистанційного навчання, соціальної мережі на рівні «КПІ», а також автоматизації навчального процесу.

Важливим для таких систем є залучення розробників для їх розвитку. А для того, щоб це було можливим, уся інфраструктура повинна бути зручною та зрозумілою для них, щоб працювати з нею не викликало труднощів, а навпаки, приносило задоволення.

Завданням даного дипломного проекту є вдосконалення існуючої архітектури, яка використовується на серверному боці системи, та на її основі розробка API підсистеми «Розклад», яке дало б змогу впровадити підсистему на цільових клієнтських платформах. Впровадження та використання покращенної архітектури дозволить:

- більш ефективно використовувати роботу розробників, які займатимуться розробкою та впровадженням підсистем у систему «Електронний Кампус»;
- забезпечити забезпечить однотипність API для різних її компонентів, що полегшить роботу розробникам стороннього програмного забезпечення;
- більш чітко розділити між собою різні ресурси системи, що спростить її розуміння та використаття;

На жаль, навіть с такою кількістю переваг подібні підсистеми мають також певні недоліки, такі як: високий рівень закладенної абстракції, який

необхідний для уніфікації архітектури, що може викликати певні труднощі на перших єтапах її освоєння; узагальнення базового функціоналу та трансформування його у програмний код, який можна використовувати багато разів, може призвести до втрати продуктивності, при некоректному використанні.

Взявши до уваги те, що покращення процесу розробки системи «Електронний Кампус» та залучення до неї нових розробників є дуже важливою, то робота проведенна в рамках цього дипломного проекту є актуальною.

1. АНАЛІЗ ІСНУЮЧОЇ АРХІТЕКТУРИ API СИСТЕМИ «ЕЛЕКТРОННИЙ КАМПУС»

1.1. Визначення поняття API

API [1] *(прикладний програмний інтерфейс, інтерфейс прикладного програмування)* - набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) для використання в зовнішніх програмних продуктах. Використовується програмістами при написанні сторонніх додатків.

В веб-розробці використовується, як правило, певний визначений набір HTTP-запитів, а також визначенні структури HTTP-відповідей, для вираження яких використовують XML або JSON формати. Основними варіантами представленнями веб-орієнтованих API є SOAP та REST.

REST [2] *(скор. від англ. Representational State Transfer - «передача репрезентативного стану»)* - метод взаємодії компонентів розподіленого додатка в мережі Інтернет, при якому виклик віддаленої процедури являє собою звичайний HTTP-запит, а необхідні дані передаються як параметри запиту.

1.2. Основні принципи роботи ASP.NET Web API

Платформа ASP.NET Web API використовує REST, як метод взаємодії між компонентами додатка та базується на взаємодії трьох компонентів: контролера, моделі і представлення.

Контролер приймає запити, обробляє користувацьке введення, взаємодіє з моделлю і представленням і повертає користувачеві результат обробки запиту. Зазвичай один контролер відповідає за одну модель та одне відображення, тобто кількість контролерів не менша, за кількість моделей. Можлива ситуація, коли кількість контролерів перевищу кількість моделей за

рахунок того, що певні контролери відповідають за роботу з однієї і тією ж моделлю, але різними її представленнями.

Модель являє собою шар, що описує логіку організації даних у додатку. Представлення представляє дані, які відображаються користувачеві, як результат обрабки запиту контролером, та може являти собою повне, або часткове відображення моделі.

При обробці запитів фреймворк спирається на систему маршрутизації, яка співставляє всі вхідні запити з визначеними в системі маршрутами, які вказують який контролер і метод повинен обробити даний запит. Вбудований за умовчанням маршрут передбачає триланкову структуру: *контролер / дія / параметр*. Шлях, який проходить запит від надходження на сервер до повернення результату наведено на рис. 1.1.

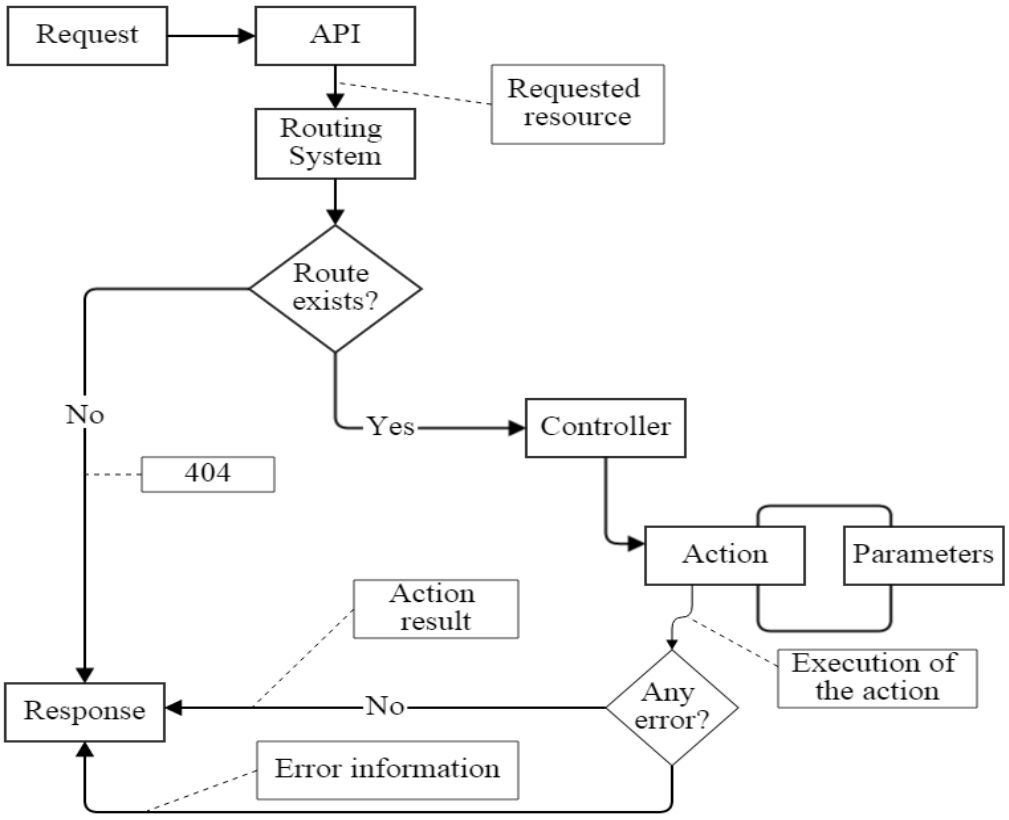

Рис. 1.1. Стандартний маршрут запиту в ASP.NET Web API

1.3. Огляд та аналіз архітектури системи «Електронний Кампус»

1.3.1. Огляд архітектури

Архітектура системи «Електронний Кампус» відповідає описаній стандартній архітектурі, та являє собою структуру, показану на рис. 1.2.

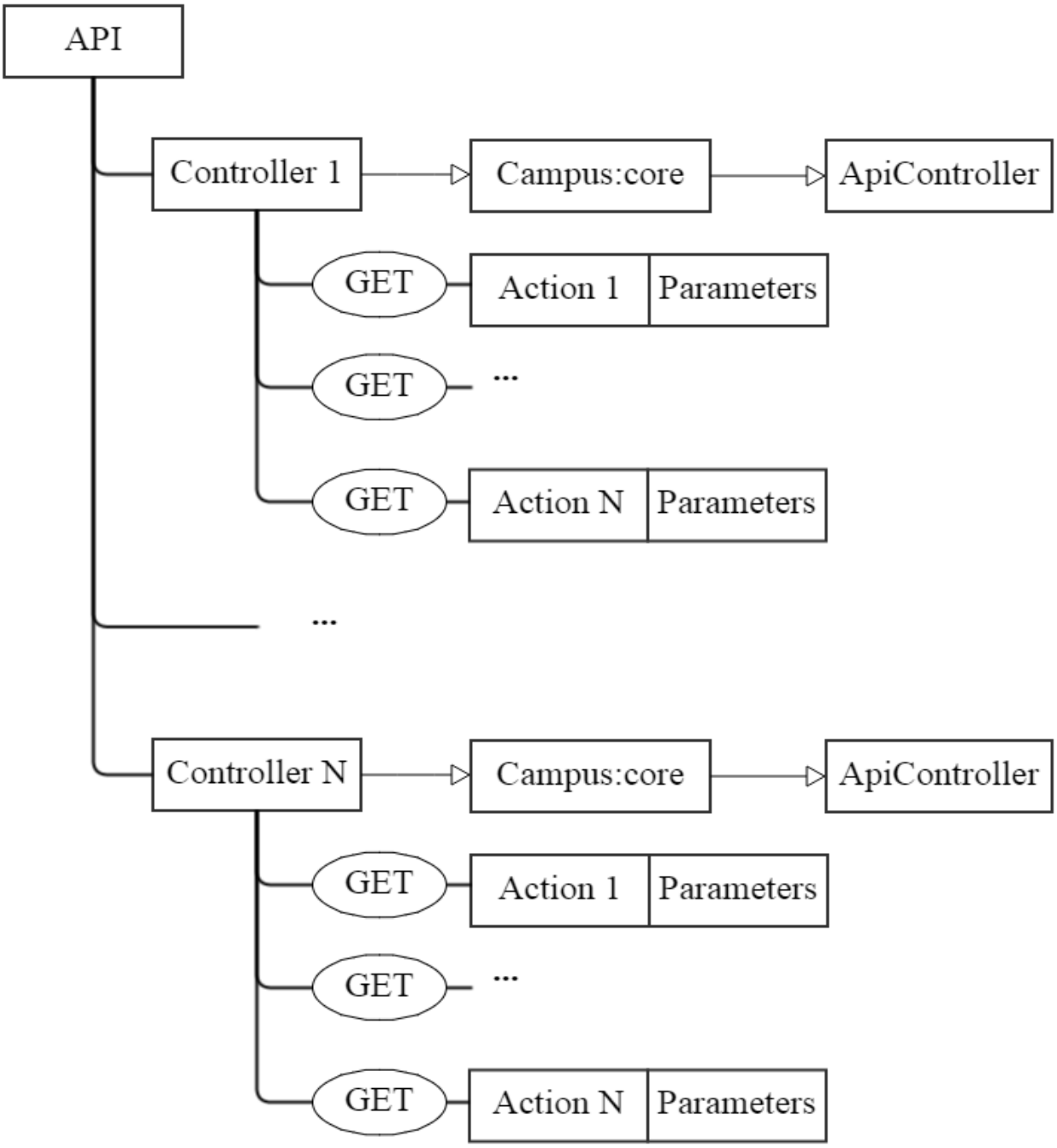

Рис. 1.2. Структура API «Електронний Кампус»

Як видно з наведенної структурної моделі, API складається з набору контролерів, кожен з яких містить у собі методи для маніпуляції та отримання даних. Усі методи використовують HTTP-метод GET, що означає, що усі маршрути у системі матимуть наступний вигляд:

{api}/{controller}/{action}?{parameter1}={parameter1 value}&… (1.1)

Кожен з реалізованих методів може являти собою як метод для отримання даних, так і метод для їх додавання, оновлення, або видалення, оскільки симантика контролеру не накладає жодних обмежень і призначення методу ідентифікується лише його назвою.

Кожен метод повертає стандартизований результат, який визначено у базовому, для кожного контролеру системи, класі *Campus.Core.ApiController* [3], та має структуру, наведену на рис. 1.3.

```
{
        "StatusCode": ,
        "TimeStamp": ,
        "ExecutionTime": ,
        "Guid":
        "Paging":,
        "Compression":,
        "Data":
}
```

Рис. 1.3. Структура відповіді методі API

1.3.2. Переваги архітектури

Перевагами архітектури можна зазначити відносну простоту її організації та відсутність жорсткої шаблонності. За рахунок практичної

відсутності спільного програмного коду загального користування, значно підвищується швидкодія, так як під кожен необхідний функціонал, пишиться окрема частина програмного коду, яка відповідає лише за нього, а не втрачає швидкість за рахунок обробки загальних ситуацій.

Контролер не обмежений певною кількістю методів, а кожен метод є незалежним один від одного і не є обмеженим у функціоналі. Таким чином, логіка, необхідна для маніпуляцій з моделлю, може бути реалізована зручним для розробника шляхом, що дає йому свободу дій та вибору.

З іншого боку, структура відповіді контролером на запити є стандартизованою та машино-читаємою, що дає змогу розробки простого інструментарію для робити з API.

1.3.3. Недоліки архітектури

Головна перевага, а точніше відсутність жорсткої шаблонності, з часом перетворюється на її найбільший недолік. Зазвичай, над різними контролерами системи працюють окремі розробники, а інколи і декілька одночасно. В кожного з них є своє бачення реалізації того чи іншого функціоналу, вподобань щодо найменування методів, тощо.

Це може призвести до дублюванню функціоналу у різних частинах системи, а інколи і в одній і тій самій, що негативно впливає на базу програмного коду, його кількість, та зрозумілість; погіршується зрозумілість кінцевого інтерфейсу, який надається користувачам API, оскільки кількість доступних функцій може невпинно збільшуватись, а виконуваний ними функціонал перетинатись.

Великим недоліком є використання лише GET-методів для спілкування з системою. По перше, при такогму підході втрачається симантичний зв’язок з визначенням та призначенням HTTP-методів [4] та порушує принципи визначені в REST для CRUD [5]. Це погіршує читаємість API для користувача, оскільки дія, яка виконується методом, може розпізнаватись тільки за назвою,

або пошуком по документації, якщо вона присутня. Таким чином. Також, за відсутності конвенції для найменувань, дії, які виконують схожі функції, можуть називатись по різному для різних контролерів, що ще більше ускладнює освоєння системи.

Ще одним недоліком, який випливає з використання GET-методів – це надмірність URI, які потрібно використовувати для роботи з ними. Якщо звернути увагу на стандартний маршрут (1.1), то можна зауважити, що усі параметри та їх значення, необхідні для передаються в URI. Уявімо ситуацію, що нам потрібно оновити об’єкт на стороні серверу, який містить у собі велику кількість полів. При використанні даного підходу, у нас є два варіанти: одним із параметрів передавати сереалізоване представлення об’єкту (у вигляді JSON, XML, тощо) у вигляді строки, або кожне поле передавати окремим параметром. У будь-якому випадку, потрібно проводити додаткові маніпуляції на стороні клієнту. Щодо сторони серверу, у першому варіанті нам стає недоступною повна десеріалізація об’єкту з отриманих даних, або дуже «дорогою», за допомогою техніки, званої рефлексією, операція часткового оновлення об’екту. Обидва варіанти погіршують показники швидкодії системи – один за рахунок збільшення кількості пересилаємих даних між клієнтом та сервером, другий за рахунок збільшення процесорного часу, який потрібен серверу для обробки запиту. У випадку з використанням окремого параметру для кожного поля об’єкту збільшується кількість одноманітного програмного коду, який повинен написати розробник, як зі сторони серверу, так і зі сторони клієнту, для того, щоб сформувати на клієнті та обробити на сервері такий запит. А виходячи з можливої кількості контролерів та методів у них, кількість коду, який потрібно написати при такому підході, грає не на користь системи.

Висновки до розділу

Отже у результаті проведених досліджень, можна зробити висновок, що осліджувана архітектура володіє як позитивними, так і негативними якостями.

Головною перевагою, як і недоліком, є простота організації. При невеликій кількості розробників та реалізуємого функціоналу, як буває на початку розробки, система проста в розгортанні та розширенні; вона не потребує розвинутої внутрішньої інфраструктури, а отже потрібно менше часу для її первинного розгортання.

При збільшені навантаження розробки, простота може перетворитись на проблему для людей, які займатимуться її підтримкую, оскільки в системі відсутня впорядкованість та шаблонність, закладені на рівні архітектури, що може призвести до зниження продуктивності розробки нового функціоналу та підтримки вже існуючого.

2. ПОСТАНОВКА ЗАДАЧІ

2.1. Загальний опис розроблюваної архітектури

Проаналізувавши архітектуру, яка використовується у системі «Електронний Кампус», зваживши усі її переваги та недоліки, можна спробувати сформулювати вимоги, яким повинна буде відповідати архітектура, яка є об’єктом розробки даної дослідницької роботи.

Перш за все, потрібно орієнтуватись на максимальну близькість до консепцій, закладених у REST [7]. Це дасть змогу надавати кінцевому користувачу добре структурований та зрозумілий веб-сервіс, освоєння якого не викликатиме труднощів. Також це збільшить простоту використання сервісу з популярними нині фреймворками, такими як AngularJS, KnockoutJS, тощо, які націлені на розробку клієнтських платформ.

Архітектура повинна представляти собою шаблон визначених методів, які відповідають принципам CRUD [8], та згруповані за їх призначенням, доступних за замовчуванням для кожного контролеру та розрахованих на роботу з будь-якою з доступних моделей. Це дасть змогу мінімізувати кількість програмного коду, необхідного для розгортання кожного нового контролеру у системі, мінімізувати можливість помилки, яка могла виникти при перенесені одноманітного коду з однієї частини системи у іншу, за рахунок винесення його у базовий клас.

2.2. Виокремлення окремих проблем

2.2.1. Розширюваність

Оскільки архітектура представляє собою каркас для побудови нових контролерів, то має бути передбачено механізм її інтуітивного розширення та переопреділення поведінки, в залежності від потреб. Бажаний резутьтат може бути досягнуто при винесенні «чутливої до обставин» логіки з базового класу у окремі сутності – сервіси. Кожному з типів сервісів делегується

відповідальність за реалізацію того, чи іншого функціоналу. За рахунок цього збільшується модульність та можливість повторного використання коду, стає дуступним використання значно більшого рівеня абстракції. При необхідності зміни поведінки контролеру, або додання нової поведінки, достатньо реалізувати та внедрити потрібний сервіс.

2.2.2. Робота з частковим представленням

Однією з описаних раніше проблем є робота з частковим представленням, тобто отримання, або оновлення неповного представлення моделі, а лише певної його частини. Це дає змогу більш оптимально використовувати інтернет-канал, за рахунок того, що запит включатиме в себе лише необхідні для роботи дані. Така поведінка описана у специфікації HTTP-методів GET та PATCH.

У мовах програмування з жорсткою типізацією, якою і являється використовувана мова C#, це є непростою задачею і шляхів вирішення у неї не так і багато. Стандартними є два підходи: написання хендлеру, для обробки дій над частковим представленням, окремо для кожного випадку та використання кросс-об’єктних підходів, які базуються на використанні рефлексії. Кожен з них має свої недоліки та переваги.

Для реалізації запланованої архітектури, потрібно розробити кросс-об’єктний підхід, який би мав високу продуктивність, порівняно з використанням рефлексії та надавав доступ до полів об’єкту використовуючи їхні імена. Це дасть змогу значно спростити задачу часткового отримання/оновлення даних у автоматичному режимі, підвищити загальну швидкодію системи, за рахунок знаження навантаження на канал зв’язку, між сервером та клієнтом, дозволить будувати моделі представлення даних на стороні клієнта в залежності від його потреб, що зменшує надмірність даних, які приймають участь у роботі.

2.2.3. Підтримка версійності

При розробці нового функціоналу та покращенні вже існуючого, важливо зберігати зворотню сумісність, так як на основі робочої версії системи може бути вже реалізована певна кількість програмних додатків. Якщо при внесені змін сигнатура інтерфейсу контролеру буде змінена, або зміни зачіплять повертаємі на запит дані, то велика кількість може вийти з ладу. Щоб цього не допустити, має бути передбачено можливість визначати версію методу. Таким чином, в системі існуватиме два методи, маршрути які відрізнятимуться лише версією, незалежно від їх фактичної назви.

Висновки до розділу

Вимоги, які поставлені у цьому розділі, цілком задовольняють вимоги до сучасних веб-сервісів, а саме поставлена висока планка щодо розширюваності проектованої архітектури, підтримка сучасних віянь щодо підтримки роботи з частковим представленням об’єктів. Реалізація поставлених вимог у програмному продукті дасть змогу отримати гнучкий та потужний фреймворк для побудови сучасних веб-сервісів.

3. РОЗРОБКА МОДИФІКОВАНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМИ «ЕЛЕКТРОНИЙ КАМПУС»

3.1. Визначення структури

Засновуючись на головному принципі REST - розділення API на логічні та самодостатні ресурси, якими можна маніпулювати, використовуючи HTTP запити, де метод (GET, POST, PUT, PATCH, DELETE) має певне значення. Для задоволення цієї потреби, шаблон має пропонувати розробнику увесь перелік наведених вище методів, базуючись на моделі даних та їх представлені, для яких розробляється контролер. Іншими словами, знаючи тільки модель та представлення, без внесеня додаткових змін зі сторони кінцевого розробника, API повинно надавати інтерфейс для отримання, додавання, видалення та зміни інформації, представленної моделлю даних. При цьому, перетворення інформації між представленням та моделлю, за замовчуванням, повинно відбуватись автоматично, не залежно від повноти представлення по відношенню до моделі.

Усі методи можна згрупувати, засновуючись на їх призначені, у п’ять програмніх інтерфейсів, перелік яких наведено у табл. 3.1.

Таблиця 3.1. Основні складові інтерфейсу системи

№	Назва дії	Назва інтерфейсу
1	отримання даних	IRESTGet
2	додавання нових даних	IRESTPost
3	зміна даних	IRESTPut
4	часткова зміна даних	IRESTPatch
5	видалення даних	IRESTDelete

Об’єднанням наведених інтерфейсів, став загальний інтерфейс IRESTful, який і представляє інтерфейс системи в цілому.

Засновуючись на наведеному переліку методів сформовано прототипи маршрутів, які відображають інтерфейс API для користувача. Докладніше перелік маршрутів, їх визначення та опис наведено у пункті 3.5.1.

Для забезпечення простоти розширюваності розроблено інфраструктуру для створення, та вмонтовування сервісів у шаблон. Для використання пропонується набір інтерфейсів, які наведених у табл. 3.2.

Таблиця 3.2. Основні складові інфраструктури підтримки сервісів

№	Призначення інтерфейсу	Назва інтерфейсу
1	Базовий інтерфейс для сервісів. Використовується для визначення внутрішніх сервісів.	IService
2	Є розширенням інтерфейсу IService. Використовується для визначення та проведення ін’єкцій користувацьких сервісів.	IInjectionService
3	Визначає необхідний набір функціоналу, необхідних для роботи з сервісами.	IServiceProvider

Інтеграція сервісів виконується за допомогою використання техніки «Dependency Injection» [9] та реалізації шаблону проектування «Service Locator» [10].

Dependency Injection представляє собою техніку, яка демонструє, як створити слабосвязані класи. За рахунок відсутності жорсткого зв’язку забезпечується простота впровадження різних реалізацій сервісу для різних контролерів. Ін’єкція проводиться базуючить на визначеній розробником зв’язці інтерфейсу сервісу і його конкретної реалізації в анотації класу.

3.2. Налаштування

Інструмент пропонує досить широкий набір налаштувань, які доступні розробнику, для забезпечення гнучкої поведінки.

Для забезпечення простоти конфігурування обрано використання анотації за допомогою атрибутів. Атрибути забезпечують ефективний спосіб зв'язування метаданих або декларативної інформації з кодом (збірками, типами, методами, властивостями, тощо). Після того як атрибут буде пов'язаний з програмною сутністю, він може бути запитаний під час виконання за допомогою рефлексії.

Атрибути мають наступні параметри:

- Атрибути додають у програму метадані. Метадані являють собою відомості про типи, визначених в програмі.
- Один або кілька атрибутів можуть застосовуватися до збірок, модулів або більш дрібних програмних елементів, таких як класи і властивості.
- Атрибути можуть приймати аргументи точно так само, як методи і властивості.

Зважаючи на наведений список властивостей, переваги такого підходу стають зробумілими. Перш за все відпадає необхідність використання конфігураційних файлів, які є не дуже читаємими для людини і є більш підходящими для машинної обробки. Замість цього використовується декорування елементу програми атрибутами, кожен з яких відповідає за певне налаштування. За рахунок цього простіше знаходити інформацію про налаштування, оскільки вони знаходяться безпосередньо біля асоційованого з нею елементом.

Оскільки атрибути є частиною компілюємого коду, розробнику значно ваще допустити помилку при налаштувані, так як при некоректності даних додаток не скомпілюється і буде виведено повідомлення з вказання детальної інформації про допущену помилку.

3.3. Маршрутизація

3.3.1. Побудова маршрутів

Маршрутизація забезпечує адресацію методів контролерів у системі. Для виконання поставлених вимог необхідно визначати маршрути безпосередньо у батьківському класі. Такий функціонал не підтримується фреймворком Web API, тому запропоновано особливий підхід до маршрутизації.

Стандартний підхід до маршрутизації полягає у визначені загального шаблону (1.1), який задає поведінку для усіх контролерів та методів у них. Такий підхід не задовольняв потреби розроблюваного фреймворку, тому вирішено формувати маршрут для кожного контролеру і методу у ньому. Для цього використано підхід «Strong-typed routing». Це дало оминути проблему, з якою зіштовхується стандартний провайдер маршрутів – він віддає перший маршрут, по якому можна відправити отриманий запит, навіть якщо у системі існує більш підходящий маршрут, який знаходиться далі у таблиці маршрутизації. А при використанні побудови прямих маршрутів, кожен запит відправляється на метод, сігнатура якого найбільше відповідає запиту.

3.3.2. Анотація маршрутів

Маршрути є одними із параметрів, які задаються атрибутами. Сигнатура атрибуту маршрутизації має наступний вигляд: (рис. 3.1)

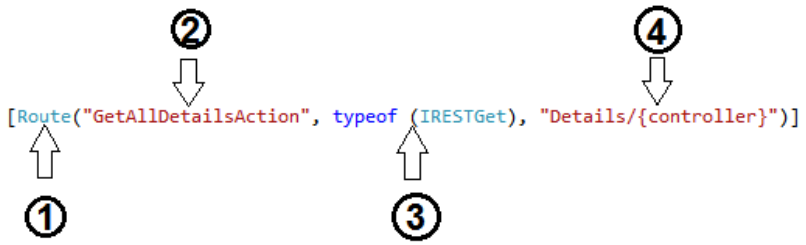

Рис. 3.1. Атрибут маршрутизації. Метод.

1) Назва атрибуту;
2) Назва маршруту. Може не співпадати з назвою методу, але має однозначно ідентифікувати його у таблиці маршрутизації, тобто бути унікальною;
3) Тип інтерфейсу (дії), до якого відноситься метод;
4) Маршрут, який буде використовуватись для навігації на позначений метод, відносно контролера, у якому він міститься.

Атрибут є наслідуваним, тому немає необхідності вказувати його постійно для кожного нового контролера. При необхідності зміни маршруту для методу, потрібно переопріділити його у дочірньому контролері і помітити його атрибутом маршруту, який задовольняє новим потребвам.

При застосуванні атрибуту до контролера, його сигнатура матиме наступний вигляд: (рис. 3.2)

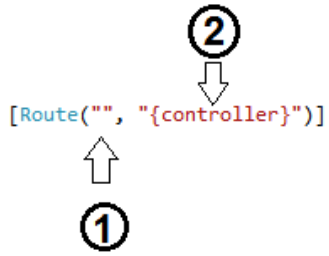

Рис. 3.2. Атрибут маршрутизації. Контролер.

1) Ім’я маршруту. Якщо значення рівне порожнє – буде використано назву класу;
2) Маршрут контролера. Використовується як префікс до маршрутів методів у контролері;

3.3.3. Версійність

Для впровадження версійності потрібно використовувати окремий атрибут, сигнатура якого наведено на рис. 3.3.

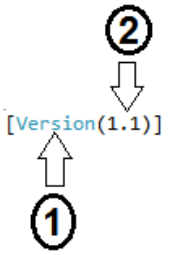

Рис. 3.3. Атрибут версії.

1) Назва атрибуту;
2) Версія. Представляє собою десяткове дробове число, дробова частина якого може бути рівна нулю.

Атрибутом може помічатись як окремий метод, так і цілий клас. Якщо атрибут застосовується до класу, він перекриває атрибут методу. Маршрут з версією має наступний вигляд:

{api}/{controller}/{v{version}}/... (3.1)

3.3.4. Відімкнення маршрутів

При проектуванні API, часто виникаєть випадки, коли ті чи інші операції не повинні бути доступними кінцевому користувачеві.

У випадку коли усі методи взаємодії описувались вручну для кожного випадку, можна просто не описувати методи, які не повинні бути доступними. Але у запропонованому рішенні вже визначено базовий набір методів і за замовчуванням усі вони підгружаються у таблицю маршрутизації.

На такий випадок фреймворком запропонований атрибут (рис. 3.4), який повідомляє маршрутизатору, що вказаний тип дії не повинен включатись до доступних маршрутів. Атрибутом помічається клас контролеру. Ним можна

відмикати будь-яку групу методів використовуючи для цього інтерфейс, який представляє ці методи.

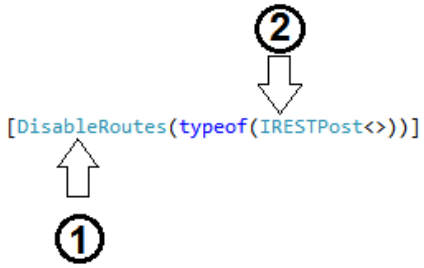

Рис. 3.4. Атрибут відімкнення маршрутів

1) Назва атрибуту;
2) Інтерфейс, який визначає відімкнені методи.

Якщо тип дії видімкнено, то система поводитиме себе так, ніби такий маршрут не існує та поверне помилку 404 (запрошений ресурс неіснує).

3.4. Робота з частковим представленням

3.4.1. Загальний опис підходу

Для того, щоб робота з частковим представленням являлась можливою, необхідно реалізувати підхід, який би дозволям логічно поєднати дві, не поєднані до цього, сутності, які на програмному рівні – це два окремих класи, які нічого не знають одне про одного – модель та її представлення. Для цього потрібно задати відповідність кожного поля представлення до поля моделі, при цьому кількість полів, імена та типи полів можуть не співпадати. Отже, лише розробник може визначити логічний зв’язок, між цими сутностями, який він закладає у представлення.

Зазвичай, для вирішення такої задачі використовується рефлексія – механізм, що дозволяє отримувати, змінювати та маніпулювати даними об’єкта на основі його метаданих.

3.4.2. Проблеми використання рефлексії

Перш за все, використання рефлексії є дуже повільним, за рахунок отримання метадати об’єктів у виконуваному коді, що відбувається при кожному її використанні, будь то виклик методу чи отримання значення поля.

В контексті розглядуваної задачі, використання рефлексії зводиться до отримання метаданих класу, пошуку поля за іменем та отриманню/зміні значення поля. Отже, для кожного поля, яке присутнє в перетворені моделі на її представлення, буде відбуватись один і той самий процес, який по своїй суті є повільним, так як за кулісами постійно має відбуватись аналіз поля-донора та поля-реципієнта, так як їх типи на етапі компіляції не є відомими.

3.4.3. Альтернативний підхід

Вирішення проблеми лежить у деталях реалізації сластивостей у .NET, а саме те, що вони являють собою за кулісами. Властивості є нічим іншим, як синтаксичним цукром. На ділі, компілятором генеруються два методи, сигнатура яких є незмінною, які відповідають за отримання та зміну значення, і при звернені до властивості викликаються саме ці методи.

Іншою складовою є поняття делегатів та делегування виклику у .NET. За допомогою використання цієї техніки можна «помістити» метод у змінну, а потім за допомогою неї його викликати. При цьому існує два варіанти, коли змінна представлена базовим типом *“Delegate”*, що означає відсутність відомостей про метод, окрім того, що він метод, або одним із двох узагальнених делегатів *“Func”* та *“Action”*, які містять у собі відомості про його сигнатуру. У прешому з них, виклик методу можливий лише використовуючи метод *“DynamicInvoke”*, який вже за кулісами збирає усі необхідні метадані для його виклику, що є досить повільною операцією. У другому випадку, виклик методу відбувається дуже швидко, оскільки вже відома сигнатура та посилання на нього у таблиці методів. Недоліком є те, що на стадії компіляції потрібно знати типи, з якими працює метод, щоб мати

повну його сигнатуру. Така проблема може бути вирішена комбінаціїю шаблонних методів, класів та рефлексією.

На основі цього можна побудувати рішення, яке буде представляти собою гібрид стандартного жорстко визначеного для кожного випадку мапінгу, де все відомо на етапі компіляції, з використанням рефлексії, який буде мати усі переваги нежорсткої типізації рефлексії, але матиме значно більшу швидкодію.

В гібридному підході конкрутні типи невідомі на етапі компіляції, але жорстко визначаються на етапі виконання, коли кешуються метадані про властивості та йде побудова делегатів доступу. Таким чином, у проксі-об’єкті кожній властивості цільового класу буду відповідати пара типізованих делегатів, які являтимуться посиланнями на оригінальні методи властивості, доступдо яких відбуватиметься індексатором, ключем до якого є ім’я властивості.

За рахунок того, що у запропонованій реалізації все сприймається як виклик методу, то це дає додаткову можливість до розширення за рахунок визначення не стандартних методів, які можуть приймати участь у побудові відображення.

Мапінг між об’єктами виконується за заданим маршрутом, який нагадує XPath у XML, де кожен наступний елемент маршруту є властивістю попереднього, або одним з зареєстрованих додаткових методів. За рахунок такого рішення складна по своїй структурі модель, може бути з легкістю відображена у набагато більш спрощене представлення, що полегшує роботу з даними.

3.4.4. Порівняння підходів

Після побудови та реалізації алгоритму відображення моделі у представлення, проведенно бенчмарк швидкості для простих тестових даних, які не міслять у собі вкладених об’єктів, а лише властивості простих типів. У

тесті приймали участь розроблений алгоритм, рефлексія, популярний на сьогоднішній день маппер, для відображення конкурентоспроможності на сучасному ринку та ручний маппінг.

Нище наведені структури моделі та відображення:

```
public class Model
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public int NumberOfOrders { get; set; }
}

public class ViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

Результати проведенних тестів зібрано до табл. 3.3.

Таблиця 3.3. Результати тестів продуктивності

№	Кількість елементів	Розроблене рішення (мс)	Reflection (мс)	AutoMapper [11] (мс)	Manual (мс)
1	1	4	1	23	1
2	10	0	0	0	0
3	100	0	0	0	0
4	1000	1	2	3	0
5	10000	12	24	34	0
6	100000	145	245	319	7
7	1000000	1445	2843	3193	20

З отриманих результатів можна побачити, що розроблена система маппінгу є майже у два рази продуктивнішою, ніж просте використання рефлексії і більш ніж у два рази продуктивнішою за популярний на ринку маппер. Отже можна прийти до висновку, що поставлена мета досягнута.

3.5. Визначення членів інтерфейсу IRESTful

3.5.1. IRESTGet

Інтерфейс IRESTGet відповідає за впровадження методів, пов’язаних з отриманням інформації. У табл. 3.4 наведено перелік прототипів маршрутів для цього інтерфейсу.

Таблиця 3.4. Перелік маршрутів IRESTGet

№	Маршрут	Опис
1	2	3
1	*{api}/{controller}/*	Отримання переліку усіх даних, які представлені контролером {controller}. Виводяться повні представлення об’єктів.
2	*{api}/Ids/{controller}/*	Отримання переліку усіх даних, які представлені контролером {controller}. Виводяться лише ідентифікатори об’єктів.
3	*{api}/{controller}/ ?sort={order}property1*	Отримання переліку усіх даних, які представлені контролером {controller} відсортованих за значенням поля property1 і напрямком {order}, який задається «+» чи «-». Виводяться повні представлення об’єктів.
4	*{api}/{controller}/{id}*	Отримання інформації про конкретний об’єкт з ідентифікатором {id} , який представлений контролером {controller}
5	*{api}/{controller}/{id}?fields= {property1},{property2}*	Отримання значенн конкретних полів {propertyN} вказаних у параметрі fields об’єкту з ідентифікатором {id

Закінчення таблиці 3.4

6	*{api}/{controller}/{id}?fields= {property1}.fields={inner_property1}*	Отриманння значеннь конкретних дочірних полів вказаних полів {propertyN} вказаних у параметрі fields об'єкту з ідентифікатором {id}
7	*{api}/Search/{controller}? {property1}={value1}& {property2}={value2}*	Пошук об'єктів, які задовольняють вказаним вимогам , які представлені контролером {controller}. Виводиться повний зміст об'єктів.
8	*{api}/Search/Ids/{controller}? {property1}={value1}& {property2}={value2}*	Пошук об'єктів, які задовольняють вказаним вимогам, які представлені контролером {controller}. Виводяться лише ідентифікатори об'єктів.

Наведений перелік є базовим і його компоненти можна комбінувати. Наприклад, для того, щоб отримати лише конкретні поля для результатів пошуку, що у значній мірі може знизити об'єм передаваємих даних, прототип маршруту виглядатиме так:

{api}/Search/{controller}?{property1}={value1}&{property2}={value2}
&fields={property2},{property3} (3.2)

Якщо результат пошуку потрібно додатково відсортувати за спаданням, запит виглядатиме наступним чином:

{api}/Search/{controller}?{property1}={value1}&{property2}={value2}
&fields={property2},{property3}&sort=-{property1} (3.3)

Такий широкий набір різноманітних варіацій запитів для отримання даних представляє собою потужний інструмент, за допомогою якого можна будувати конструкції, які забезпечать результати найбільш підходящі до потреб користувача API, на основі яких можна збудувати простіші та більш інформативн моделі, які підходитимуть кожному окремому користувачеві, та

забезпечить зменшиння об’ємів надлишково використовуваного трафіку на передачу збиткової інформації.

3.5.2. IRESTPost

Інтерфейс IRESTPost відповідає за впровадження методів, пов’язаних з доданням інформації. У табл. 3.5 наведено перелік прототипів маршрутів для цього інтерфейсу.

Таблиця 3.5. Перелік маршрутів інтерфейсу IRESTPost

№	Маршрут	Опис
1	2	3
1	*{api}/{controller}/(single json object)*	Додає об’єкт, який передано у тілі запиту, до бази даних.
2	*{api}/{controller}/(multiple json objects)*	Додає об’єкти, які передано у тілі запиту, до бази даних.

Як видно з наведених маршрутів, які обслуговують HTTP-метод POST, вхідна точка у них однакова, але вони відрізняються вхідними даними.

Використання можливості додавання декількох об’єктів одночасно може значно зменшити кількість запитів до бази даних, за рахунок відправлення не кожного об’єкту окремо, а одразу певної їх кількості, оскільки час на ініціалізацію операції буде витрачатись один раз для всіх об’єктів, а не для кожного окремо.

У відповідь на запит, користувачеві відправляються додані ним об’єкти, з оновленими необхідними полями в залежності від реалізації, наприклад з присвоєними id. Це потрібно для того, щоб користувач мав актуальний стан об’єктів після їх додання.

3.5.3. IRESTPut

Інтерфейс IRESTPut відповідає за впровадження методів, пов’язаних із повною зміною інформації. У табл. 3.6 наведено перелік прототипів маршрутів для цього інтерфейсу.

Таблиця 3.6. Перелік маршрутів інтерфейсу IRESTPut

№	Маршрут	Опис
1	2	3
1	*{api}/{controller}/{id}* *(single json object)*	Замінює об’єкт, який відповідає ідентифікатору {id} на переданий у тілі запиту об’єктом.
2	*{api}/{controller}/* *(single/multiple json object(s))*	Замінює об’єкт(и), інентифікатор якого, у переданому об’єкті, відповідає існуючому у БД

Маршрути, які обслуговують HTTP-метод PUT, мають різні вхідні точки. Це зобумовлено простотою спрощенням використання. При такій побудові маршрутів доступно обслуговування одного і тільки одного об’єкту, якщо використовується маршрут 1, так як він потребує чіткого вказання його ідентифікатору, або обслуговування невизначеної кількості об’єктів, при використанні маршруту 2, який приймає як один об’єкт представлення, так і їх масив, представлені у вигляді JSON.

3.5.4. IRESTPatch

Інтерфейс IRESTPatch відповідає за впровадження методів, пов’язаних із частковою зміною інформації. У табл. 3.7 наведено перелік прототипів маршрутів для цього інтерфейсу.

Таблиця 3.7. Перелік маршрутів інтерфейсу IRESTPatch

№	Маршрут	Опис
1	2	3
1	*{api}/{controller}/{id}* *(single json object)*	Вносить передані у тілі запиту зміни до об’єкту, ідентифікатор якого відповідає {id}.
2	*{api}/{controller}/* *(single/multiple json changes object(s))*	Замінює об’єкт(и) на основі переданих у тілі змін, інентифікатор якого, у переданому об’єкті, відповідає існуючому у БД

HTTP-метод PATCH та маршрути, які його обслуговуються, доповлюють модель роботи з частковим представленням, яка наведена у розділі 3.5.1, так як без них неможливо працювати з неповною моделлю представлення, яку можна отримати при вказанні конкретних полів для отримання, оскільки HTTP-метод PUT надає можливість лише повного оновлення даних.

3.5.5. IRESTDelete

Інтерфейс IRESTDelete відповідає за впровадження методів, пов’язаних із частковою зміною інформації. У табл. 3.8 наведено перелік прототипів маршрутів для цього інтерфейсу.

Таблиця 3.8. Перелік маршрутів інтерфейсу IRESTDelete

№	Маршрут	Опис
1	2	3
1	*{api}/{controller}/{id}*	Видаляє об’єкт, ідентифікатор якого відповідає {id}.
2	*{api}/Ids/{controller}/* *(array of ids in json object)*	Видаляє об’єкти на основі переданих у тілі ідентифікаторів.

Закінчення таблиці 3.8

З	*{api}/{controller}/ (single/multiple json object(s))*	Видаляє об’єкт(и) на основі переданого у тілі представлення.

HTTP-метод DELETE та маршрути, які його обслуговують, надають можливість видаляти дані з БД, при цьому не обмежуючи форму, як задаються ці дані.

3.6. Розширюваність.

3.6.1. Базові можливості розширення

Як і будь-який додаток, який написано в об’єктно-орієнтованому стилі, в запропонованному рішені доступна можливість перевизначення віртуальник членів класу у класі-наслідникy. За допомогою цього механізму можна докорінно змінити логіку поведінки контролера. Це може бути корисно, якщо функціонал, який реалізовується, потребує цілком інакшої логіки обробки, ніж запропоновано у базовому класі. Такі зміни можуть порушити роботу інфраструктури в цілому, якщо перевизначення не виконано правильно, тому розширювати та змінювати поведінку батьківського класу таким чином рекомендується лише розробникам, які маєть високий рівень ознайомлення зі структурою тулкіту.

Іншим важливим моментом є те, що можна створювати свої базові класи, які будуть засновані на запропонованому, але будуть розширені необхідним набором логіки, яка необхідна у цільових контролерах розроблюваної системи, а одже зменшиться кількість одноманітного коду за рахунок винесення його у батьківський клас. Це може бути корисним у ситуаціях, коли до певної кількості цільових контролерів приміняється один і той самий шаблон налаштувань і для того, щоб не дублювати усі атрибути на усіх контролерах, можна зробити лише один батьківський клас, у якому будуть

виконані усі налаштування, які будуть автоматично перейняті класами-наслідниками.

3.6.2. Ін’єкції у методи

Іншим способом розширення функціоналу базових методів одного із інтерфейсів є використання ін’єкцій.

Ін’єкція – вбудовування додаткової логіки у існуючий метод без його перевизначення. Інфраструктура для виконання ін’єкцій закладена як на рівні реалізації кожного окремого методу базового інтерфейсу, так і на рівні архітектури в цілому. Ланцюжок виконання операцій будується окремо для кожного базового інтерфейсу на етапі ініціалізації системи. За рахунок цього значно зменшуються витрати на виконання ін’єкції безпосередньо на етапі виконання операцій контролера.

Ін’єкції мають жорстко визначену сигнатуру, якій має відповідати метод, який буде використовуватись як тіло ін’єкції. Сигнатури методів наведено у табл. 3.9.

Таблиця 3.9. Сигнатура методів-ін’єкцій

№	Сигнатура	Інтерфейс
1	2	3
1	+ MethodName (data : ActionContainer<IQueryable<T>>) : ActionContainer<IQueryable<T>>	IRESTGet
2	+ MethodName (data : ActionContainer<T>) : ActionContainer<T>	IRESTPost
3	+ MethodName (data : ActionContainer<T>) : ActionContainer<T>	IRESTPut
4	+ MethodName (data : ActionContainer<T>) : ActionContainer<T>	IRESTPatch

Закінчення таблиці 3.9

5	+ MethodName (data : ActionContainer<T>) : ActionContainer<T>	IRESTDelete

Для того, щоб сказати системі, що метод повинен використовуватись у якості ін’єкції, його потрібно відмітити спеціальним атрибутом (рис. 3.5)

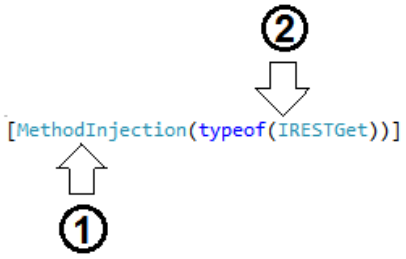

Рис. 3.5. Атрибут ін’єкції

1) Назва атрибуту;
2) Базовий інтерфейс, для методів якого буде проводитись ін’єкція.

Ін’єкції є самим підходящим місцем для введення додаткових (користувацьких) сервісів у систему, так як система гарантує виконання усіх ін’єкцій, але не гарантує їх порядок, що, в принципі, не є важливим для даної задачі.

Ін’єкції є корисними тоді, вносимі зміни є унікальними для кожного контролеру. У іншому випадку використання сервісів, які реалізують інтерфейс IInjectionService є доцільнішим.

3.6.3. Сервіси

Найбільш потужним інструментом для розширення на кастомізації системи являються сервіси. За їх допомоги можна побудувати потрібний функціонал з мінімальною кількістю дублювання програмного коду, що збільшує простоту подальшого використання та розширення.

У системі представлено три основних інтерфейси (табл. 3.2), які обслуговують інфраструктуру сервісів. Інтерфейс *IServiceProvider* є службовим і не буде розглядатись у даному розділі. Розгляданню підлягають лише перші два інтерфейси у таблиці. Основним з них є перший, *IService*, так як він є базовим для усіх інших інтерфейсів у інфраструктурі. На його основі побудовано чотири інтерфейси, які є прототипами основного функціоналу, який повинен бути присутнім у значній більшості контролерів. Ці сервіси наведено у табл. 3.10.

Таблиця 3.10. Інтерфейси-прототипи

№	Призначення інтерфейсу	Назва інтерфейсу
1	2	3
1	Інтерфейс, який представляє функціонал авторизації. Реалізація інтерфейсу повинна описувати процес авторизації кожного окремого користувача у системі, його права доступу та інше. Збурігає у собі авторизаційні дані в період одної операції.	IAuthService
2	Реалізації інтерфейсу є відповідальними за візуальне подання відповіді контролеру. Через цей інтерфейс проходять усі потоки виводу, тому його реалізація повинна бути максимально простою та швидкою.	IDataPresentationService
3	Являє собою сервіс перетворення інформації з моделі даних у її представлення, а також у зворотньому напрямку. Є центральним і найбільш навантаженим сервісом системи.	IDataService
4	Являє собою базовий інтерфейс для реалізації інтерфейсів та сервісів, які відповідатимуть за роботу з запитами фільтрації та вибірки.	IQueryService

Закінчення таблиці 3.10

5	Виконує схожу роль, як і *MethodInjection* атрибут, який описано у розділі 3.6.2, але на рівні сервісу, а не на рівні класу, що забезпечує більший простір для повторного використання програмного коду.	IInjectionService

Для забезпечення простоти розгортання деякі з базових інтерфейсів мають реалізацію за замовчуванням, яка представляє собою реалізацію базової поведінки, яка забезпечить коректну роботу при відсутності необхідності персоналізувати ті чи інші аспекти поведінки системи. У табл. 3.11 наведено список класів та їх опис, які представляють собою реалізацію базових інтерфейсів.

Таблиця 3.11. Базові реалізації інтерфейсів-прототипів

№	Реалізована поведінка	Назва інтерфейсу	Назва класу
1	2	3	
1	Клас являє собою заглушку, яка забезпечує можливість використання контролеру без авторизації. Це необхідно для зберігання цілісності системи незалежно від того, чи проводиться авторизація.	IAuthService	DefaultAuthService
2	Реалізація класу забезпечує маппінг між моделлю та її представленням на основі розробленої та описаної у розділі 3.4 технології. Це забезпечує максимально просту та загальну імплементацію, яка дає змогу працювати маппінгу одразу після розгортання системи.	IDataService	DefaultDataService

Закінчення таблиці 3.11

3	Клас являє собою заглушку, яка забезпечує подачу даних без зміни їх вигляду.	IDataPresentationService	DefaultPresentation-Service

Для втілення власних реалізацій базових інтерфейсів можна використовувати реалізації за замовчуванням, у випадках коли не є необхідним реалізація з нуля, і переопреділити в них необхідні методи чи властивості.

Висновки до розділу

Реалізована архітектура відповідає вимогам поставленим у розділі 2. Розроблено гнучку та легко-розширювану інфраструктуру, яка підтримує різноманітні варіанти розширення та втілення необхідної бізнес-логіки. Система реалізована з урахуванням більшості кращих практик реалізації REST сервісів, що робить системи, розроблені з її допомогою, легкими у використанні для кінцевих користувачів.

4. ПРАКТИЧНЕ ЗАСТОСУВАННЯ

4.1. Введення

Для розробки підсистеми «Розклад» для системи «Електронний Кампус» засновуючись на розробленому, в ході виконання дипломної роботи, фреймворку потрібно сфокусуватись на розробці моделі предметної області. Перш за все потрібно проаналізувати потоки даних, які будуть присутні у підсистемі.

Розклад представляє собою набір статичних та нестатичних даних. До статичних відносяться дані, які представляють сам розклад зайнять та не мають прив’язки до дати, а лише до номеру навчального тижня, який визначено деканатами факультетів і майже не підлягає змінам напротязі усього навчального семестру. Нестатичні дані являють собою користувацькі дані, такі як персональні зміни, внесені викладачами або студентами, які накладаються та комбінуються зі статичними даними. Різні типи даних потребують різних підходів до їх зберігання. Для збереження статичних даних є достатнім використовувати одноразово заповнене сховище даних без представлення інтерфейсу для їх редагування зі сторони користувача API. Замість цього інтерфейс API повинен представляти інтерфейс, заснований на врев’язці до конкретної дати, який надаватиме можливість створювати, змінювати та видаляти динамічні дані, при цьому при необхідності імітувати зміни до статичних даних у випадку, коли вносяться зміни, які стосуються навчального розкладу, але не охоплюють період усього семестру, а лише певний його проміжок. Внесені зміни можуть стосуватись як окремого користувача, як наприклад персональна подія, так і цілих груп користувачів, як наприклад при перенесенні зайняття з однієї аудиторії в іншу.

Оскільки філософія REST пропагує роботу з ресурсом в цілому, а не з набором визначених методів, то потрібно зосередитись на побудові моделей, які б цілком задовольняли описані потоки даних, та зв’язку цих моделей з їх

представленням, з яким безпосередньо працюватиме користувач API. Представлення повинно приховувати від кінцевого користувача деталі реалізації структури даних, які лежать у основі системи, для спрощення освоєння та використання API.

4.2. Розробка моделі даних

Як сховище даних використовується реляційна база даних SQL Server. Для зберігання даних пов’язаних із розкладом, знадобиться чотири таблиці, які зображено на рис. 4.1.

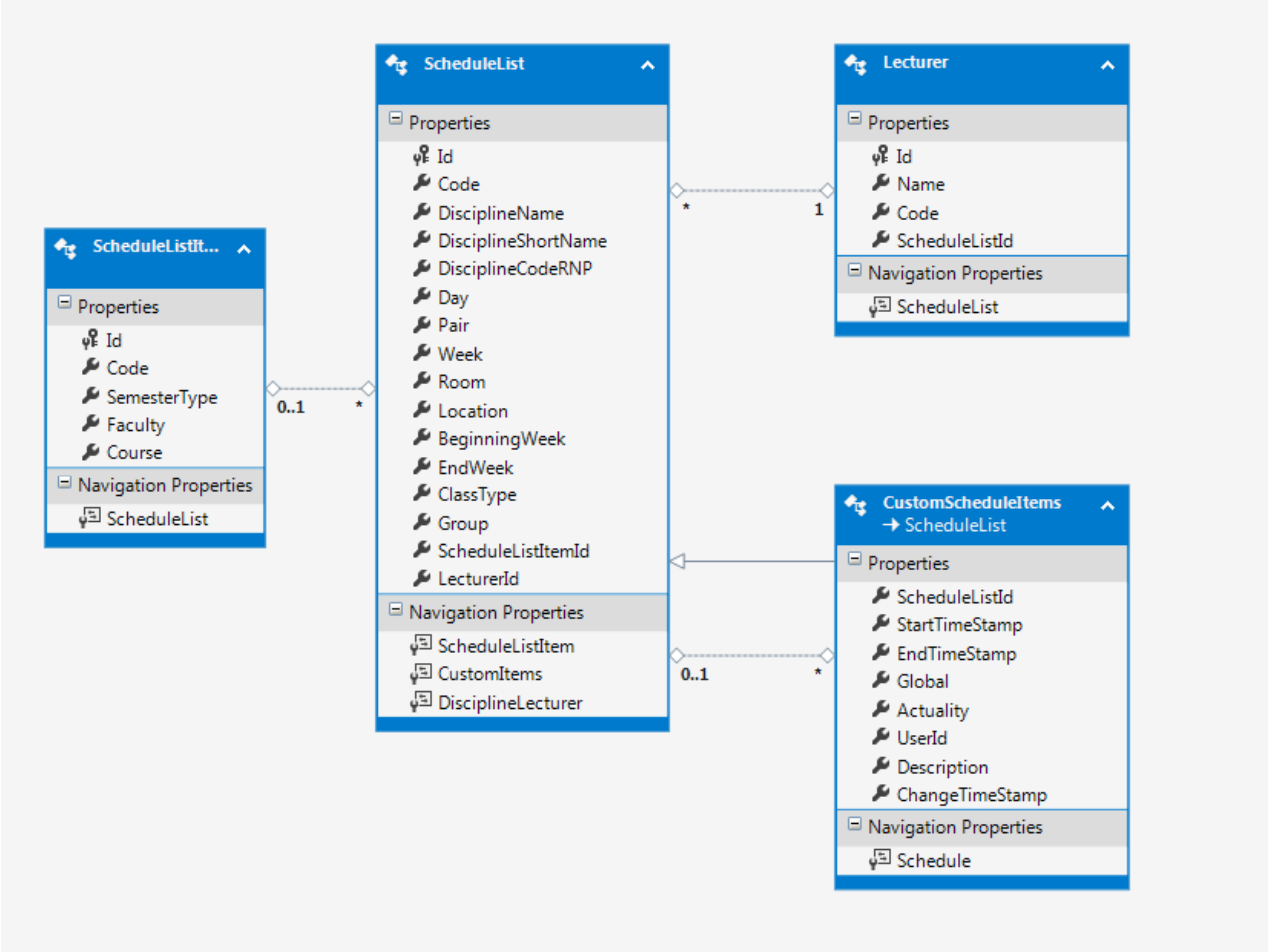

Рис. 4.1. Модель даних для підсистеми «Розклад»

4.3. Розробка представлення

Після визначення моделі даних, потрібно визначити представлення, яке буде використовуватись для доступу до моделі.

Перш за все, потрібно визначити обмеження, які накладаються на вивід результатів в залежності від користувача, який виконав запит, базуючись на авторизаційних даних. У системі «Електронний Кампус» існує два типи користувачів – студент та викладач. В залежності від типу користувача потрібно накладати різні фільтри на результати запиту, адже кожному користувачеві доступні лише пов’язані з ним дані.

Якщо користувача визначено як студента, то з даних про нього можна визначити факультет, кафедру га группу у якій навчається цей студент, а отже з бази даних можна здійснювати вибірку розкладу для цієї групи. У випадку авторизації викладача, можна визначити факультети та кафедри, на яких він викладає та вивести усі елементи розкладу, які пов’язано з ним. Отже інтерфейс контролера не повинен турбуватись про визначення даних про користувача, оскільки це вже зроблено на рівні авторизації. Одним із додаткових фільтрів є відсіювання неактуальних для обраної дати подій, засновуючись на полях, які визначають дати початку та кінця події, а також флаг актуальності. Остаточне представлення подано на рис. 4.2.

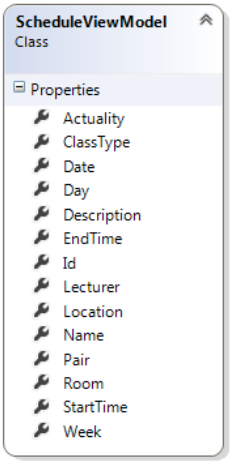

Рис. 4.2. Представлення моделі даних

4.4. Визначення зв’язку моделі з представленням

Так як модель та представлення вже визначені, можна перейти по опису зв’язку між ними. Для виконання маппінгу вирішено використовувати розроблену систему, описану у розділі 3.4, так як вона поєднує у собі простоту конфігурації та відносно високу продуктивність. Після побудови маппінгу отримано наступні додаткові зв’язки, які необхідно сконфігурувати по причині неспівпадання маршрутів у моделі та представлені:

```
Mapper.Create<CustomScheduleItems, ScheduleViewModel>()
         .Remap("DisciplineName", "Name")
         .Remap("DisciplineLecturer/Name", "Lecturer");

      Mapper.Create<ScheduleViewModel, CustomScheduleItems>()
          .Remap("Name", "DisciplineName")
           .Remap("Lecturer", "DisciplineLecturer/Name");
```

4.5. Результуючий контролер для роботи з підсистемою «Розклад»

Використовуючи фреймворк, який розроблено в ході виконання даної дипломної роботи, отримано контролер, який забезпечує весь спектр необхідного функціоналу необхідного для розгортання підсистеми «Розклад», програмний код якого представлено нище:

```
[Route("Schedule", "Schedule")]
[AuthService(typeof(CampusAuthService))]
public class ScheduleController :
         CRUDProvider<ScheduleViewModel, CustomScheduleItems>
{
    private readonly Repository _repository = new Repository();
    protected override IRepository Repository
    {
        get { return _repository; }
    }

    static ScheduleController()
    {
        // mapping model to its viewmodel
        Mapper.Create<CustomScheduleItems, ScheduleViewModel>()
           .Remap("DisciplineName", "Name")
           .Remap("DisciplineLecturer/Name", "Lecturer");

        // mapping viewmodel back to model
        Mapper.Create<ScheduleViewModel, CustomScheduleItems>()
            .Remap("Name", "DisciplineName")
            .Remap("Lecturer", "DisciplineLecturer/Name");
    }
```

```
        [System.Web.Http.AcceptVerbs("GET")]
        [Route("GetByDateAction", typeof(IRESTGet), "/{controller}/{date}")]
        public virtual HttpResponseMessage GetPropertyAction(HttpRequestMessage request,
                                                string date)
        {
            var d = Convert.ToDateTime(date);

            return ValidateAndInvoke(request,
                (ActionContainer<IQueryable<ScheduleViewModel>> container) =>
                    _get(CreateContainer(DataProvider.AsQueryable())
                        .Apply(o => o.Errors = container.Errors)
                        .Apply(o => o.Exceptions = container.Exceptions))
                        .Where(o => d.Date >= o.StartDate.Date &&
                                    d.Date <= o.EndDate.Date)
                    .AsEnumerable()
                    .Select(DataService.Map<CustomScheduleItems, ScheduleViewModel>)
                    .AsQueryable(),
                HttpStatusCode.OK,
                null,
                InterfaceTypes[Interface.Get]);
        }
    }
```

Як видно, контролер має досить невелику кількість програмного коду, але при цьому реалізовує великий спектр можливостей для маніпулювання пов’язаними з ним даними (див. розділ 3.5), а час, затрачений на його розробку, незрівнянно менший, ніж якби довелося розроблювати усі методи власноруч.

Висновки до розділу

Реалізація підсистеми «Розклад» є гарним прикладом використання розробленої архітектури контролеру для веб-сервісу, як за її допомогою у короткий термін та з використанням малої кількості програмного коду можна розгорнути повноцінну частину веб-сервісу, яка містить у собі усі необхідні інструменти для роботи з даними. Це дасть змогу розробникам концентруватись на виконанні корисної роботи, а не витрачати час на написання одноманітного коду для роботи з базою даних, що допоможе зробити розроблюваний продукт кращим.

5. ОХОРОНА ПРАЦІ

Дипломна робота на тему «Розробка інформаційного та програмного забезпечення підсистеми Електронного Кампусу "Розклад" з підтримкою мобільних платформ. Розробка модифікованої архітектури серверної частини» напряму пов’язана з комп’ютерним моделюванням та розробкою програмного забезпечення.

Суб’єктом в даному розділі є інженер, який виконує проектування веб-сервісів з використанням персонального комп’ютера. Передбачається, що місцем його роботи буде комп’ютерна лабораторія на підприємстві, що розрахована на 3х осіб.

Робота на комп’ютері може мати негативний вплив фізичних чинників, призвести до серйозних проблем фізичного та психологічного стану.

Метою цього розділу є аналіз умов безпеки праці на обраному робочому місці, виявлення шкідливих і небезпечних факторів виробничого середовища і порівняння їх з діючими нормативами, а також розробка заходів, націлених на утворення умов праці, що відповідають вимогам усіх нормативно-правових актів з охорони праці.

Впровадження заходів з охорони праці дозволить гарантувати працівнику збереження його здоров’я та працездатності.

5.1. Характеристика об'єкту та умови його експлуатації

Робоче місце суб’єкта знаходиться в одній із комп’ютерних лабораторій підприємства, яка обладнана для роботи трьох інженерів. Лінійні розміри становлять 7м×5,5м, висота стелі 2,8м. У приміщенні, використовується змішане освітлення. Стіни пофарбовані в світло-жовтий колір а на підлозі лежить світлий паркет. Спрощений план приміщення приведено на рис. 5.1.

Як основні характеристики приміщення приймаються його геометричні розміри і кількість працюючих у ньому людей. Розміри аналізованого приміщення приведені в табл. 5.1.

Таблиця 5.1. Розміри приміщення

Найменування	Позначення	Значення, м
Довжина	A	7
Ширина	B	5,5
Висота	H	2,8

Таблиця 5.2. Площа та обсяг приміщення, на одного працюючого

Геометрична характеристика	Одиниця виміру	Нормативне Значення	Фактичне значення
Площа, *S*	$м^2$	не менш 6.0	9,6
Обсяг, *V*	$м^3$	не менш 20	26,95

За даними, приведеним у табл. 5.2, можна зробити висновок, що геометричні розміри приміщення відповідають нормативним вимогам згідно ДНАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

Основним робочим положенням є положення сидячи. Головними елементами робочого місця є письмовий стіл, крісло і комп’ютер.

З меблів в лабораторії знаходяться чотири столи для комп’ютерів, чотири крісла, дві шафи з документацією, та стіл для розташування іншої техніки.

З техніки тут розташовані чотири персональні комп’ютери, принтер та факс.

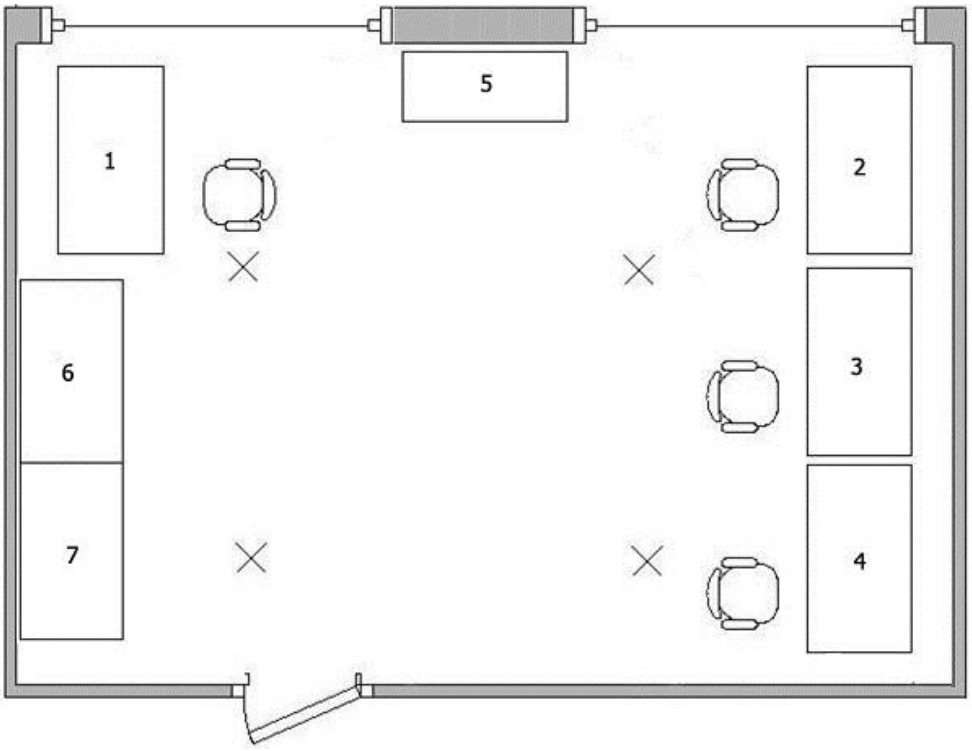

Рис. 5.1 Спрощений план приміщення

1,2,3,4 – робочі місця з комп’ютерами; 5 – стіл для принтера та факсу;

6,7 – шафи.

Розглянемо робоче місце користувача ПК з точки зору оцінки впливу шкідливих виробничих факторів відповідно до Гігієнічного нормативу ГН 3.3.5-8-6.6.1-2002 «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу». Відповідно до цього документу, на працівника, який працює з комп’ютером діють такі шкідливі виробничі чинники:

1. Мікроклімат робочої зони;
2. Недостатність штучного освітлення;
3. Виробничий шум;
4. Виробничі випромінювання;
5. Пожежонебезпека.

5.1.1. Мікроклімат робочої зони

Відповідно до встановлених гігієнічно-санітарних вимог (ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень») роботодавець зобов'язаний забезпечити в приміщеннях для даного типу роботи (категорія Легка – 1а) оптимальні параметри виробничого середовища. Параметри мікроклімату згідно з нормами повинні бути наступними:

Таблиця 5.3. Норми мікроклімату для приміщень з ЕОМ

Пора року	Параметр мікроклімату	Оптимальне значення	Фактичне значення
Холодна	Температура повітря	22-24°С	23°С
	Відносна вологість повітря	60 - 40%	31%
	Швидкість руху повітря	0,1м/с	0,1м/с
Тепла	Температура повітря	23 - 25°С	24,5° С
	Відносна вологість повітря	60 - 40%	58%
	Швидкість руху повітря	0,1 м/с	0,1м/с

Причиною підвищеної температури робочої зони можуть бути освітлювальні пристрої, величина тепловиділення яких становить 35-60 Вт/м^2, а також комп’ютер, середня величина тепловиділення якого становить 310 Вт/м^2.

5.1.2. Освітлення

Причиною недостатньості природного освітлення може бути неправильно спроектоване розміщення робочого місця відносно джерел природного освітлення або ж слабке світлопроникнення вікон черех їх забрудненість.

Основним документом, який регламентує норми освітленості є ДБН.В.2.5-28-2006 «Природне і штучне освітлення». Освітлення у приміщеннях, де знаходиться робоче місце працівника, використовується змішане.

У якості природнього в даному приміщені представлене одностороннє бокове освітлення через два вікна розміром 2,5м×1,5м . Напрямок розміщення вікон східний. Коефіцієнт природної освітленості ~ 1,7%.

Для штучного освітлення в подібних приміщеннях необхідно використовувати джерела світла з досить великим ККД у світильниках, які розташовуються над робочими поверхнями у рівномірно – прямокутному порядку. Найкраще підходять в таких приміщень світлодіодні(LED) лампи, які мають один з найвищих показників світловіддачі. У нашому випадку використовуються чотири світильника зі звичайними люмінесцентними лампами.

Штучне освітлення повинно забезпечити на робочих подібних місцях освітленість 300 – 500 лк. На робочому місці,що розглядається, фактичне значення освітленості становить 200-250 лк. Це пов’язано з нерівномірністю розміщення світильників.

При правильно розрахованому і виконаному освітленні виробничих приміщень, очі працюючого на протязі тривалого часу зберігають здатність добре розрізняти предмети, не стомлюючись. Такі умови сприяють зниженню виробничого травматизму і професійного захворювання очей. Раціональне освітлення має задовольняти ряду вимог і умов. Воно має бути:

– достатнім, щоб очі без напруги могли розрізняти предмети;

– постійним у часі, для цього напруга в мережі живлення не повинне коливатися більш ніж на 4%;

– рівномірно розподіленим по робочих поверхнях, щоб очам не приходилося попадати з дуже темного місця у світле і навпаки;

– таким, що не здійснює сліпучу дію на око людини як від самого джерела світла, так і від поверхонь, що віддзеркалюють його та знаходяться в полі зору працюючого. Зменшення сліпучої дії джерел досягається застосуванням світильників які розсіюють світло;

– не викликати різких тіней на робочих місцях. Цього можна уникнути при правильному розташуванні світильників.

Згідно ДБН В.2.5-28-2006 «Природне і штучне освітлення» для даних робіт встановлена необхідна освітленість робочого місця $E_{\text{н}} = 300$ лк.

5.1.3. Виробничий шум

На комп'ютеризованих робочих місцях основними джерелами шуму є вентилятори системного блоку, принтери. Сильний шум викликає труднощі з розпізнаванням колірних сигналів, знижує швидкість сприйняття кольорів, гостроту зору, зорову адаптацію, порушує сприйняття візуальної інформації, зменшує на 5-12% продуктивність праці.

На даному робочому місці основними джерелами шуму є вентилятори системи охолодження системного блоку комп'ютера, а також принтери та факс. Також варто врахувати шум, що надходить ззовні, і який ліквідується використанням акустичних поглиначів звуку, а також вікон, що щільно закриваються.

Для покращення робочої обстановки необхідне технічне вдосконалення та періодичне обслуговування системних систем охолодження комп'ютерів. А принтери перемістити за межі лабораторії, або помістити в звукоізоляційну коробку.

Методи вимірювання шуму регламентовано ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».

Розрахуємо рівень шуму в приміщені.

Таблиця 5.4. Значення рівня шуму для типових джерел

Джерело шуму	Рівень шуму, дБА
Жорсткий диск	30
Вентилятор	45
Монітор	15
Клавіатура	8
Принтер	40
Факс	45

Максимальний час роботи принтера за один день – 1,5 години.

Робочий день Т = 8 годин.

$$L_{\Sigma} = 10\,lg(10^{3} + 10^{4,5} + 10^{1,7} + 10^{0,8} + 10^{4} + 10^{4}) = 48{,}7 \text{ дБА}$$

При роботі на комп’ютері рівень шуму відповідно до постанови ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» не повинен перевищувати 50 дБА, а фактичний 48,7 дБА. Отже, наше приміщення відповідає діючим санітарним нормам.

5.1.4. Виробничі випромінювання

Обладнання робочого місця за усіма вимогами і наявність сертифікованого персонального комп’ютера не дає повної гарантії електромагнітної безпеки користувача, навіть за умови використання сучасного рідкокристалічного монітора. Однак для підвищення захищеності користувача комп’ютера від прямого та опосередкованого впливу

електромагнітних полів та випромінювань необхідно розглянути увесь інформаційно-технічний комплекс з точки зору надійності функціонування.

Вимірювання полів від персональних комп'ютерів показали, що сучасні сертифіковані відео монітори в основному відповідають вимогам чинних нормативних актів з електромагнітної безпеки. Однак при невірному взаємному розташуванні моніторів рівні полів на робочих місцях можуть перевищувати гранично допустимі.

Сумарне електромагнітне поле з боку сучасних рідкокристалічних моніторів значно менше за гранично допустиме. Проте у діапазоні 2-400 кГц мають місце досить великі рівні полів на частотах 150-200кГц.

Крім електромагнітних полів та випромінювань безпосередньо від монітора, на користувача додатково впливають так звані фонові поля – поля від сторонніх джерел, які знаходяться у приміщенні або поблизу від нього. Такими джерелами є мережі живлення і освітлення, побутові прилади (кондиціонер, обігрівач), мобільні телефони, бездротова мережа тощо. Досліди показали, що рівні напруженості полів на робочих місцях, розташованих, наприклад, поблизу працюючих кондиціонерів, збільшується на 15-20%. Значне зростання полів також спостерігається у просторах між масивними металевими предметами та комп'ютерами.

5.1.5. Пожежна безпека

Пожежну та вибухову безпеку регламентують Закон України «Про пожежну безпеку», а також ОНТП 24-86 «Визначення категорій приміщень і будівель по вибухопожежній і пожежній небезпеці»), які є обов'язковим для виконання всіма підприємствами незалежно від форми власності. Правила встановлюють загальні вимоги з пожежної безпеки.

Безпека людей має здійснюватися при виникненні пожежі в будь-якому місці виробничої будівлі, споруди або територій підприємства. При виникненні пожежі на людей можуть впливати небезпечні чинники: відкритий

вогонь та іскри; підвищена температура повітря, предметів, обладнання; токсичні продукти горіння, дим; знижена концентрація кисню; обвалення і пошкодження будівель, споруд, установок, вибух.

Основними причинами пожежі та вибуху на підприємствах є наступні:

- несправність виробничого обладнання;
- несправність та перенавантаження електричного обладнання;
- необережне ставлення до вогню (паління, використання відкритого вогню в недозволених місцях, залишання без нагляду електрообладнання);
- порушення правил пожежної безпеки.

В приміщені класу «В», що розглядається, повинно бути встановлена система пожежної сигналізації з димовими пожежними сповіщувачами (з розрахунку 2 шт. на кожні 20 м^2 площі приміщення) та переносні порошкові вогнегасники (для даного приміщення достатньо одного на лабораторію).

У сучасних комп'ютерах повинні бути передбачені всі захисні заходи щодо пожежі:

– монітори захищені від вибуху та займання;

– на процесорах стоять захисники, що виключає можливість займання процесора;

– заземлення.

Отже, пожежна та вибухова безпека забезпечується:

– використанням методів та пристроїв запобігання іскріння;

– своєчасним контролем за справним станом обладнання;

– систематичною очисткою вентиляційних каналів від пилу і перевіркою системи вентиляції;

– підтримкою чистоти та порядку всередині приміщення;

– відсутністю всередині приміщення легкозаймистих та вибухових речовин;

– застосуванням запобіжників;

– дотриманням протипожежних вимог до електрообладнання;
– захистом від блискавки будинку і устаткування відповідно;
– використанням пожежної сигналізації.

5.2. Інструкції з техніки безпеки

Дана інструкція діє для персоналу, що експлуатує комп'ютери та периферійне обладнання, а також побутові електроприлади (електрочайники, кавоварки тощо). Інструкція містить загальні вказівки щодо безпечного застосування електрообладнання в організації. Вимоги даної інструкції є обов'язковими, будь-які відхилення від неї є недопустимими. До самостійної експлуатації комп'ютерів та електроапаратури допускається лише спеціально навчений персонал.

Вимоги до безпеки:

- Перед початком роботи переконайтесь у:
  - справності електропроводки вимикачів
  - справності штепсельних розеток, за допомогою яких обладнання підключається до мережі
  - наявності заземлення комп'ютера
  - справності комп'ютера та периферійних засобів.
- Задля запобігання пошкодження ізоляції проводів та виникнення короткого замикання забороняється: вішати будь-що на дроти, засовувати дроти та шнури за водопровідні труби, за батареї опалювальної системи, висмикувати штепсельну вилку з розетки за дріт (зусилля мають бути прикладені до корпусу вилки).
- Щоб запобігти враженню електричним струмом забороняється:
  - часто вмикати та вимикати комп'ютер без необхідності;
  - торкатись деталей комп'ютера та периферійного обладнання вологими руками;

- працювати з комп’ютером та периферійними пристроями, якщо вони мають порушення цілісності корпусу, порушення ізоляції дротів, несправну індикацію живлення, з ознаками електричної напруги на корпусі;
- під напругою проводити ремонт комп’ютерів та периферійного обладнання. Ремонт електроапаратури проводять тільки спеціалісти-техніки з дотриманням необхідних технічних вимог;
- очищувати від пилу та забруднення електрообладнання, коли воно знаходиться під напругою;
- перевіряти справність електрообладнання в непристосованих для експлуатації приміщеннях зі струмопровідною підлогами, вологих, таких, що не дозволяють заземлити доступні металеві частини;
- при користуванні електроприладами торкатись одночасно трубопроводів, батарей опалення, металевих конструкцій, що з’єднані з землею.

- Після закінчення роботи необхідно знеструмити всі комп’ютери, периферійне обладнання та електроприлади.
- У випадку неперервного виробничого процесу дозволяється залишити ввімкненим тільки необхідне обладнання.

Висновок по розділу

У результаті проведеного аналізу умов безпеки праці на робочому місці працівника виявлені шкідливі і небезпечні фактори, а також визначені та запропоновані варіанти вирішення його недоліків.

Так для покращення освітлення запропоновані нові та значно ефективніші світлодіодні лампи, яскравість яких складає 2520 лм, що нічим не гірше від люмінесцентних ламп, але строк служби яких при цьому довший в 5

– 10 разів. До того ж вони стійкі до механічних пошкоджень та низьковольтні, а значить – безпечніші.

Також проведено розрахунок рівня шуму в приміщені та встановлено, що він задовольняє норми.

Окрім цього розглянуті інструкції з охорони праці, питання пожежної безпеки та визначено необхідні умови для її забезпечення.

ВИСНОВКИ

Запропонована структура веб-сервісів є достатньо гнучкою для використання у різноманітних сферах. Вона може легко інтегруватись у вже існуючу систему. Також її легко налаштувати під специфічні потреби розроблюваної системи.

У рамках розробки вдосконаленої архітектури для системи «Електронний Кампус» створено гнучкий, легкий у вивченні фреймворк для побудови REST сервісів на базі ASP.NET Web API.

Для прикладу розроблено підсистему «Розклад», в ході побудови якої доведено ефективність створеного інструментарію.

Потенціальними користувачами є розробники програмного забезпечення, які займаються проектуванням та розробкою різноманітних веб-орієнтованих сервісів, яким в час жорсткої конкуренції потрібно швидко додавати нові можливості та покращення, при цьому мінімізувати затрати на розробку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Application Programming Interface [Електронний ресурс] : https://en.wikipedia.org/wiki/Application_programming_interface
2. Representation State Transfer [Електронний ресурс] : https://en.wikipedia.org/wiki/Representational_state_transfer
3. campus:core — пример библиотеки для реализации API в системе с унаследованным кодом [Електронний ресурс] : http://habrahabr.ru/post/222909/
4. HTTP/1.1: Method Definitions [Електронний ресурс] : http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
5. Using HTTP Methods for RESTful Services [Електронний ресурс] : http://www.restapitutorial.com/lessons/httpmethods.html
6. Reflection (C# and Visual Basic) [Електронний ресурс] : https://msdn.microsoft.com/en-us/library/ms173183.aspx
7. A Guide to Designing and Building RESTful Web Services [Електронний ресурс] : https://msdn.microsoft.com/en-us/library/dd203052.aspx
8. Create, read, update and delete [Електронний ресурс] : http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
9. Dependency Injection Pattern [Електронний ресурс] : https://msdn.microsoft.com/en-us/library/ff921152.aspx
10. Service Locator Pattern [Електронний ресурс] : https://msdn.microsoft.com/ru-ru/library/ff648968.aspx
11. AutoMapper – Offical site [Електронний ресурс] : http://automapper.org
12. «Комплекс методичних вказівок до виконання дипломних проектів»: підручник / [авт.кільк.: М.М. Поліщук, М.М.Ткач, В.П. Пасько, О.І. Чумаченко, О.І. Лісовиченко, О.А. Стенін], - Київ: Дорадо-Друк.2014.