МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут електронних та інформаційних технологій

Кафедра інформаційних та комп’ютерних систем

**Допущено до захисту**

Завідувач кафедри

к.е.н., доцент Базилевич В.М.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 р.

ВИПУСКНА КВАЛІФІКЦІЙНА РОБОТА

за освітньо-професійною програмою бакалавра

ІНФОРМАЦІЙНА КОМП’ЮТЕРНА СИСТЕМА   
АВТОМАТИЗОВАННОГО ДИСТАНЦІЙНОГО ТЕСТУВАННЯ

Спеціальність 123 – Комп’ютерна інженерія

Галузь знань 12 – Інформаційні технології

Виконавець:

студент гр. КІ–162

Ступак Сергій Олександрович

*(підпис)*

Керівник:

|  |  |  |
| --- | --- | --- |
| *доцент* |  | *к.т.н., доцент* |
| **–––––––––––––––––––––––––––––** |  | **–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––** |
| *(посада)* |  | *(науковий ступінь, вчене звання)* |
|  |  |  |

Бивойно Павло Георгійович

*(підпис)*

Чернігів 2020

**РЕФЕРАТ**

Кваліфікаційна робота бакалавра, с., рис., табл., джерел, додаток.

**THE ABSTRACT**

**ЗМІСТ**

**Вступ**

Основною рисою сучасних веб-технологій є кросс-платформенність та здатність надавати користувачу доступ до персоналізованого контенту на різних пристроях з різними операційними системами, зберігаючи при цьому цілісність користувацького досвіду та універсальну доступність.

Для потреб навчальних закладів було розроблено багато продуктів, призначених для тестування знань студентів. Але переважна більшість цих продуктів орієнтована на роботу в класі та мають за платформу настільні операційні системи. Недоліком такого підходу є потрібність в установці спеціалізованого програмного забезпечення, прив’язка користувача системи до певної програмної та апаратної платформи та обмеження доступності системи поза навчальним класом.

В даній дипломній роботі розробляється система тестування знань студентів, яка має розширити можливості дистанційного доступу як викладачів так і студентів до створення та проходження тестів. Це розширяє можливості викладачів та студентів, збільшує кількість користувацьких сценаріїв, дозволяє ефективніше використовувати позакласний час для підготовки та самого тестування. Це має особливу важливість для дистанційного навчання, яке в останній час набуває все більшої та більшої популярності.

Метою завдання є розробка комп’ютерної системи автоматизованого тестування знань студентів із можливістю віддаленого та кросплатформеного доступу до адміністрування та тестування.

Для досягнення мети перед розробником поставлене завдання розробити базу даних, в якій будуть зберігатися дані користувачів, тестів та результатів, службу доступу до даних, яка буде надавати клієнтським додаткам доступ до них та клієнтські додатки. Найбільш універсальною платформою, яка дозволяє охопити найбільшу кількість платформ, операційних систем та пристроїв є веб-браузер, тому до завдань передусім входить розробка браузерного клієнту. Також, враховуючи особливості мобільних пристроїв (відносно невеликих розмір екрану, сенсорний інтерфейс), поставлене завдання розробки мобільного клієнту для ОС Андроід, опціонально для ОС iOS.

Об’єктом розробки є, в широкому сенсі, процес навчання групи осіб (студентів) під керівництвом керівника та наставника (викладача), організація структури цього процесу, проміжний та кінцевий контроль результатів навчання та сертифікація отримання студентами певних знань та практичних навичок. У вузькому сенсі об’єктом розробки є контроль знань студентів за певними темами та напрямками.

Предметом розробки є система оцінювання та контролю знань студентів, яка дозволяє викладачу швидко та централізовано надавати студентам завдання по контролю знань та навичок та отримувати результати такого контролю із мінімальними затратами часу та зусиль, у зручних для викладача та студентів умовах, тобто система автоматизованого дистанційного тестування.

Основними аспектами, які мають бути вирішені при вибори методології розробки проекту є вибір ядра методології та моделі розробки. Основними ядрами, які використовуються в сучасному програмуванні є:

* Імперативне програмування
* Об’єктно орієнтоване програмування
* Функціональне програмування

Щодо методів розробки, то вони поділяються на

* Ітеративні
* Каскадні
* Змішані (спіральні)

Враховуючи те, що основним завданням проекту що розробляється є збереження та класифікація даних (завдань, рішень), то в якості оптимальної моделі для ядра розробки можна запропонувати об’єктно-орієнтоване програмування, яке дозволить відобразити структура даних, якими оперуватиме система у вигляді класів, а взаємодію даних у вигляді взаємодії між класами.

Щодо методики розробки, то, враховуючи специфіку проекту та особливості виконання була обрана ітеративна модель, в рамках якої було вирішено розробляти систему шляхом паралельної розробки бази даних, служби доступу до даних та клієнтських додатків, поступово додаючи структури да функції.

**1 АНАЛІЗ ЗАДАЧІ СТВОРЕННЯ СИСТЕМИ**

**Аналіз програм-аналогів**

**Аналіз предметної області**

Предметною областю роботи є тестування знань студентів. Тестування знань у навчальному процеси призначено, перед усім, для вирішення завдань діагностики та навчання. Діагностична функція полягає у визначенні ступеню оволодіння студентами знаннями та навичками, які вони вивчають. Ця функція тестування є основною. Навчальна функція тестування полягає у мотивації студента до більш активного та поглибленого вивчення навчального матеріалу. Тестування в якості засобу контролю знань має такі переваги:

- Стандартизація процедури контролю

- Неупередженість та об’єктивність як процедури контролю, так і оцінки її результатів

- Можливість оцінювати знання як по певній частині курсу, так і по курсу в цілому, на відміну від усного чи письмового екзамену

- Можливість кількісної оцінки знань, точнішої за традиційну чотирибальну та, навіть, дванадцятибальну шкалу.

- Економія часу та коштів завдяки можливості одночасно тестувати велику кількість студентів

Тести можуть бути класифіковані за різноманітними критеріями:

- За метою — інформаційні, діагностичні, навчальні, екзаменаційні

- За процедурою створення — стандартизовані, не стандартизовані

- За способом формування завдань — детерміновані, стохастичні, динамічні

- За технологією проведення — паперові, із ручним контролем, паперові із автоматизованим контролем за допомогою технології оптичного зчитування, апаратні аналогові (із застосуванням спеціально виготовленого обладнання), апаратні цифрові (із застосуванням програмного забезпечення)

- За формою завдань — закриті, відкриті, встановлення відповідності, впорядкування послідовності

- За інтерактивністю — традиційні та адаптивні.

Традиційний тест містить перелік питань та варіантів відповідей. Кожна відповідь оцінюється в певну кількість балів. Результат тесту залежить від кількості питань та сумарної кількості набраних студентом балів.

Як правило питання в тестах розташовуються по мірі зростання складності або, якщо тестуються знання цілого навчального курсу, групуються по темам, які входять до курсу.

Існують тести, в яких питання, які отримує студент залежать від відповідей на попередні запитання. В такому тесті кожен студент може отримувати завдання в різній послідовності чи навіть отримати різну кількість завдань.

Головна перевага комп’ютерного тестування перед традиційним тестуванням на папері це можливість динамічно формувати набір тестових завдань для кожного студента на випадковій основі чи на основі попередніх результатів тестувань. Це дозволяє покрити тестами більшу область знань, не збільшуючи при цьому розмір та тривалість самого тесту.

Архітектурно предметна область являє собою викладачів та студентів, які взаємодіють за допомогою власних клієнтських пристроїв з базою даних тестування через посередництво служби доступу до даних.

В предметній області додатку можна виділити такі бізнес-процесі

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Процес | Виконавець | Вхідні дані | | Вихідні дані | |
| Постачальник | Зміст | Споживач | Зміст |
| 1 | Додати предмет | Клієнтський додаток | Викладач | Новий предмет | База даних | Новий предмет |
| 2 | Додати курс | Клієнтський додаток | Викладач | Новий курс | База даних | Новий курс |
| 3 | Додати тест | Клієнтський додаток | Викладач | Новий тест | База даних | Новий тест |
| 4 | Змінити предмет | Клієнтський додаток | Викладач | Предмет, який підлягає зміні | База даних | Змінений предмет |
| 5 | Змінити курс | Клієнтський додаток | Викладач | Курс, який підлягає зміні | База даних | Змінений курс |
| 6 | Змінити тест | Клієнтський додаток | Викладач | Тест, який підлягає зміні | База даних | Змінений тест |
| 7 | Додати користувача | Клієнтський додаток | Адміністратор, студент | Дані нового користувача | База даних | Дані нового користувача |
| 8 | Отримати список тестів | Клієнтський додаток | База даних | Ознака, за якою мають бути відібрані тести (предмет, курс) | Викладач, студент | Список тестів |
| 9 | Тримати список питань | Клієнтський додаток | База даних | Тест, по якому треба отримати питання | Викладач/студент | Список питань |
| 9 | Додати інформацію про тестування | Клієнтський додаток | Студент | Інформація про проведене тестування | База даних | Інформація про проведене тестування |
| 10 | Переглянути результати тестування | Клієнтський додаток | База даних | Результати тестування | Викладач | Результати тестування. |

Означені бізнес-процеси використовуватимуться при проектуванні та реалізації серверної та клієнтської частин програмного продукту

**Побудова вихідної концептуально моделі даних предметної області**

Концепутальне проектування бази даних полягає у розробці концептуальної моделі бази даних, яка не враховує обрану модель даних та особливості цільової СКБД. Концепутальна модель, зазвичай представлена у вигляді сукупності типів сутностей та зв’язків між ними. Побудова концептуальнох моделі даних предметної області виконана в Microsoft Office Visio.

Можна виділити такі типи сутностей:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Назва | Опис | Категорія | Кількість екзкмплярів | Режим оновлення |
| 1 | Відповідь | Містить дані про відповідь на питання | Стрижньова | 10000 | 50% на рік |
| 2 | Питання | Містить питання до тестів | Асоціативна | 1000 | 50% на рік |
| 3. | Тест | Містить організовану колекцію питань | Асоціативна | 1000 | 50% на рік |
| 4 | Курс | Містить структуру навчальних курсів | Асоціативна | 1000 | 50% на рік |
| 5 | Предмет | Містить перелік предметів, до яких відносяться курси | Асоціативна | 100 | 10% на рік |
| 6 | Результат тестування | Містить результат проходження тесту студентом | Стрижньова | 10000 | 90% на рік |
| 7 | Викладач | Містить дані викладачів | Стрижньова | 100 | 10% на рік |
| 8 | Студент | Містить дані студента | Стрижньова | 10000 | 90% на рік |

І такі зв’язки між сутностями

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Тип сутності 1 | Тип сутності 2 | Опис | Потужність | Обов’язковість |
| 1 | Питання | Відповідь | Питання має відповідь, відповідь відноситься до питання | 1:Б | Обов’язкова (питання обов’язково має відповіді, відповідь обов’язково відноситься до питання) |
| 2 | Тест | Питання | Тест має питання, питання відноситься до тестів | 1:Б | Обов’язкова (тест обов’язково має питання, питання обов’язково відноситься до тесту) |
| 3 | Курс | Тест | Курс має тести, тест відноситься до курсу | Б:Б | Необов’язкова (курс може не мати тестів, тест може не відноситися до курсу) |
| 4 | Предмет | Курс | Предмет має курси, курс відноситься до предмету | 1:Б | Необов’язкова (предмет може не мати курсів, курс може не відноситися до предмету) |
| 5 | Предмет | Тест | Предмет має тести, тест відноситься до предмету | Б:Б | Необов’язкова(предмет може не мати тестів, тест може не відноситися до предмету |
| 6 | Студент | Тест | Студенту отримує завдання пройти тест. Тест заданий студенту | Б:Б | Необов’язкова(студенту можуть не бути задані тести, тест може бути заданий жодному студенту) |
| 7 | Студент | Курс | Студент зарахований на курс. Курс проходиться студентом | Б:Б | Необов’язкова (студент може бути не зарахований на жоден курс, курс може не мати жодного студента) |
| 8 | Результат тестування | Студент | Результат отриманий студентом, студент пройшов тест та отримав результат | Б:1 | Обов’язкова (результат обов’язково отриманий студентом) |
| 9 | Результат тестування | Тест | Результат отриманий при проходженні тесту. Тест пройдений певним результатом | Б:1 | Обов’язкова (результат обов’язково отриманий при проходженні тесту) |
| 10 | Викладач | Предмет | Викладач викладає певні предмети. Предмети викладаються викладачами | Б:Б | Необов’язкове (викладач може не мати жодного предмету, можуть бути предмети без викладача) |

Сутності мають відповідні атрибути

Атрибути сутності «Відповідь»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id відповіді | long |  | PK |
| Відповідь | Текст відповіді | String |  | Ні |
| Відсоток | Відсоток правильності відповіді | Int | 0-100 | Ні |
| Порядковий номер | Положення відповіді в списку відповідей | Int |  | Ні |
| Id питання | Id питання, до якого належить відповідь | long | FK | Ні |

Атрибути сутності «Питання»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id питання | long |  | PK |
| Питання | Текст питання | String |  | Ні |
| Порядковий номер | Положення питання в списку питань у тесті | Int |  | Ні |
| Id тесту | Id тесту, до якого належить питання | Long | FK |  |

Атрибути сутності «Тест»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id тесту | long |  | PK |
| Назва | Назва тесту | string |  | Ні |
| Id автора | Id викладача, який є власником тесту | long | FK | Ні |

Атрибути сутності «Курс»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id курсу | long |  | PK |
| Назва | Назва курсу | string |  | Ні |
| Id предмету | Id предмету, до якого належить курс | long | FK | Так |
| Id автору | Id власника курсу | long |  | Ні |

Атрибути сутності «Предмет»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id предмету | long |  | PK |
| Назва | Назва предмету | string |  | Ні |

Атрибути сутності «Результат тестування»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id результату | Long |  | PK |
| Id тесту | Id тесту, який проходив студент | long | FK | Ні |
| Id студенту | Id студенту, який проходив тест | long | FK | Ні |
| Час проходження | Дата і час проходження студентом тесту | DateTime |  | Ні |

Атрибути сутності «Відповідь студента»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id відповіді | Long |  | PK |
| Id результату | Id результату тесту, до якого відноситься відповідь | long | FK | Ні |
| Id відповіді | Id відповіді, даної студентом | long | FK | Ні |

Атрибути сутності «Викладач»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id викладача | Long |  | PK |
| Ім’я | Ім’я викладача | string |  | Ні |
| Прізвище | Прізвище викладача | string |  | Ні |
| По-батькові | По-батькові викладача | string |  | Так |
| Телефон | Телефон викладача | string |  | Так |
| Електронна пошта | Електронна пошта викладача | string |  | Ні |
| Логін | Логін викладача | string |  | Ні |
| Пароль | Пароль викладача | string |  | Ні |

Атрибути сутності «Студент»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування | Опис | Тип даних | Область допустимих значень | Можливість значення NULL |
| Id | Id студента | Long |  | PK |
| Ім’я | Ім’я студента | string |  | Ні |
| Прізвище | Прізвище студента | string |  | Ні |
| По-батькові | По-батькові студента | string |  | Так |
| Телефон | Телефон студента | string |  | Так |
| Електронна пошта | Електронна пошта студента | string |  | Ні |
| Логін | Логін студента | string |  | Ні |
| Пароль | Пароль студента | string |  | Ні |



Концептуальна модель системи

Відповідно до поставленого завдання була побудована модель предметної області. Діаграма варіантів користування представлена на малюнку

Діаграма користувацьких сценаріїв

Після створення діаграми користувацький сценаріїв та виявлення ролей можна перейти до визначення кожного сценарію — деталізації сценаріїв. В результаті ми отримаємо деталізовані сценарії, які включають назву сценарію та його специфікації.

|  |
| --- |
| Сценарій: Додати предмет |
| ID:1 |
| Короткий опис:  Додавання нового предмету |
| Головні актори:  Викладач |
| Передумови:  Сценарій починається коли викладач вибирає опцію «Додати предмет» |
| Основний потік:   1. Викладач вводить в систему назву нового предмету 2. Система перевіряє коректність введених даних (наявність назви предмету) 3. Система зберігає предмет в базі даних 4. В базі даних зберігається новий перелік предметів |
| Післяумови:  Система оновлює список предметів |
| Альтернативні потоки: немає |

|  |
| --- |
| Сценарій: Додати курс |
| ID:2 |
| Короткий опис:  Додавання нового курсу |
| Головні актори:  Викладач |
| Передумови:  Сценарій починається коли викладач вибирає опцію «Додати курс» |
| Основний потік:   1. Викладач вводить в систему назву нового курсу 2. Система перевіряє коректність введених даних (наявність назви курсу) 3. Система зберігає курс в базі даних 4. В базі даних зберігається новий перелік курсів |
| Післяумови:  Система оновлює список курсів |
| Альтернативні потоки: немає |

|  |
| --- |
| Сценарій: Додати тест |
| ID:3 |
| Короткий опис:  Додавання нового тесту |
| Головні актори:  Викладач |
| Передумови:  Сценарій починається коли викладач вибирає опцію «Додати тест» |
| Основний потік:   1. Викладач вводить в систему назву нового тесту 2. Викладач додає до тесту принаймні одне питання 3. Система перевіряє коректність введених даних (наявність питань та назви тесту) 4. Система зберігає тест в базі даних 5. В базі даних зберігається новий перелік тестів |
| Післяумови:  Система оновлює список тестів |
| Альтернативні потоки: немає |

|  |
| --- |
| Сценарій: Додати питання |
| ID:4 |
| Короткий опис:  Додавання нового питання |
| Головні актори:  Викладач |
| Передумови:  Сценарій починається коли викладач обрав тест та обрав опцію «Додати питання» |
| Основний потік:   1. Викладач вводить в систему назву текст нового питання 2. Викладач вводить перелік варіантів відповідей та помічає вірні відповіді 3. Система перевіряє коректність введених даних (наявність питання, наявність відповідей, наявність правильних відповідей, що дають в сумі 100% правильності) 4. При виявленні некоректних даних    1. Видати повідомлення про некоректні дані 5. Інакше    1. Система додає нове питання 6. Нове питання зберігається в базі даних |
| Післяумови:  Система оновлює список питань |
| Альтернативні потоки: немає |

|  |
| --- |
| Сценарій: Пройти тест |
| ID:5 |
| Короткий опис:  Студент виконує проходження тесту |
| Головні актори:  Студент |
| Передумови: Немає |
| Основний потік:   1. Студент обирає тест для проходження. 2. Система видає студенту перелік питань тесту 3. Студент обирає відповіді на питання 4. Студент обирає опцію «Завершити тест» 5. Система перевіряє, чи не перевищила кількість обраних відповідей допустиму 6. При перебільшенні кількості допустимих відповідей    1. Система виводить повідомлення про перебільшення кількості допустимих відповідей 7. Інакше    1. Система видає на екран результати тестування |
| Післяумови:  Система зберігає результат проходження тесту д бази даних |
| Альтернативні потоки: Виконання тесту перервано |

|  |
| --- |
| Сценарій: Проходження тесту преревано |
| ID:5.1 |
| Короткий опис:  Студент відміняє проходження тесту |
| Головні актори:  Студент |
| Передумови:  Студент активує проходження тесту |
| Основний потік:   1. Студент обирає опцію «Припинити виконання тесту» 2. Система запитує у студента підтвердження припинення виконання тесту 3. Якщо студент підтвердив вибір    1. Система відкидає результати проходженя тесту 4. Інакше    1. Система повертається до виконання тесту |
| Післяумови:  Немає |
| Альтернативні потоки: немає |

Для наочного представлення поведінки сценаріїв можна використати діаграму діяльності.



Діаграма діяльності сценарію «Додати тест»



Діаграма діяльності сценарію «Проходження тесту»

Для відображення взаємодії між окресленими класами на етапі побудови моделі аналізу розглянемо діаграму послідовностей для сценарію «Додати питання»



Діаграма послідовності допомагає встановити часовий взаємозв’язок між класами, тобто показує реалізацію поведінки сценарію, але не є точним представленням кожного його кроку. Учасником діаграми є користувацький інтерфейс, який не розробляється до етапу проектування, тому на діаграмі послідовностей може бути не вказаний. В аналізі нас цікавить лише основна поведінка класів аналізу.

В даному випадку розглядається успішний розвиток сценарію, при якому викладач обрав тест, ввів вміст питання, можливі відповіді та вказав вірні. Система перевірила дані та за результатами перевірки додала нове питання до бази даних.

**Аналіз вимог до інтерфейсу системи**

Користувачами системи будуть студенти та викладачі, тобто особи, які вивчають певні області теоретичних чи практичних знань під керівництвом наставників та ці наставники.

Група викладачів складається з людей, які мають знання та навички в певній ділянці та на постійній чи епізодичній основі передають ці знання та навички іншим особам. Група студентів — це люди, які навчаються у викладачів. Викладачі є постійною та стабільною групою помірної чисельності (порядка десятків-сотень осіб), студенти ж є біль мінливою групою, чисельність студентів набагато більша і може перевищувати кількість викладачіва на 1-2 порядки (у одного викладача до кількох десятків студентів), а, враховуючи збереження даних неактивних студентів в базі — і набагато більше.

Щодо особистих характеристик, то інших особливостей у груп викладачів та студентів немає і їх можна віднести до категорії звичайних користувачів комп’ютерних та інтернет-технологій із відповідними знаннями та навичками.

В залежності від інтенсивності навчального процесу та розміру навчальної установи, яка користуватиметься системою інтенсивність користування може варіювати від помірної до високої. Також потрібно мати на увазі що, враховуючи особливості процесу навчання (періоди оволодіння знаннями змінюються періодами тестування та перевірки) навантаження на систему може бути нерівномірним у часі.

Враховуючи те, що більшість користувачів системи, найвірогідніше не матимуть особливих технічних навичок, інтерфейс системи не має містити складних елементів керування, має бути інтуітивно зрозумілим. Наявність двох клієнтів (браузерного та мобільного) вимагає проектування інтерфейсу таким чином, аби користувацький доступ на обох клієнтах був максимально наближений, тобто слід уникати технологій та елементів керування особливо специфічних для певного варіанту клієнту, або робити використання таких елементів прозорим та очевидним.

Через те, що планується використання інтерфейсу на пристроях із невеликим фізичним розміром екрану (мобільні пристрої) слід за можливістю розбити інтерфейс на сторінки із відносно невеликою кількістю даних та елементів керування на кожній, забезпечивши користувачу можливість вільної прозорої навігації між такими сторінками

Тобто в користувацькому інтерфейсі системи варто використовувати прості очевидні елементи керування (текстові поля вводу, кнопки, списки), розбивши інтерфейс на невеликі логічно-функціональні блоки. В той самий час слід уникати надмірної атомізації інтерфейсу аби користувач не мав потреби надмірної навігації між сторінками для вводу пов’язаних між собою даних та функцій.

**Аналіз вимог до апаратної частини**

Апаратна частина продукту що розробляєтся має складатися із двох основних компонентів: серверного та клієнтського, зв’язок між якими має забезпечуватися за допомогою мережи.

Основним інформаційним потоком в системі є передача даних між серверним та клієнтським компонентами. Серверна частина матиме внутрішній інформаційний потік — обмін даними між сервером бази даних на службою, яка надає ці дані клієнтам.

Аналізуючи вимоги до апаратної частини системи ми маємо враховувати, перед усім, можливий рівень навантаження на базу даних та мережу. Цей рівень залежить від кількості студентів, викладачів, навчальних курсів та частоти і графіку тестування, що плануються, тому окреслити ці вимоги універсально неможливо. Вони мають розроблятися індивідуально при імплементації системи конкретним користувачем.

**Постановка задачі на розробку**

Отже метою розробки є розробка комп’ютерної системи автоматизованого тестування знань студентів із можливістю віддаленого та кросплатформеного доступу до адміністрування та тестування.

Для реалізації цієї мети перед розробником стоїть завдання розробити базу даних, службу для обміну даних між базою та клієнтськими додатками та клієнтські додатки.

Програмна частина розробки має забезпечувати наступні функціональні вимоги:

* Можливість викладача створювати та змінювати тести
* Можливість викладача переглядати результати тестів, пройдених студентами
* Можливість студента проходити тести
* Можливість студента переглядати результати власних тестів

та наступні нефункціональні вимоги

* Захист даних в базі, який би унеможливлював доступ користувачів до даних, які не призначені для його ролі
* Надійність доступу та збереження даних
* Можливість клієнтських додатків забезпечувати прийнятний користувацький досвід на пристроях із невисокими технічними характеристиками.

Вимоги до апаратної системи полягають в наданні стабільного доступу клієнтських додатків до бази даних на прийнятній швидкості. Конкретні технічні характеристики серверної частини можуть залежати від запланованого конкретним користувачем навантаження на неї.

**Розробка**

**Вибір технічних засобів побудови системи**

При виборі програмної платформи для системи ми виділяємо такі бажані властивості:

* Ефективність технології та наявність в ній розвинених та зручних інструментів для досягнення мети та завдань розробки.
* Придатність технології для підтримки
* Здатність технології підтримувати кросплатформенінсть та відкриті стандарти.
* Легка та ефективна сумісність технологій, обраних для різних платформ між собою
* Максимальна уніфікація та повторне використання програмного коду в різних компонентах продукту.

Врахувавши всі ці вимоги для розробки були обрані наступні програмні платформи і технології:

1. СКБД MS SQL Server 2019
2. .NET Core
3. Мова програмування C#
4. ORM Entity Framework Core
5. Веб-фреймворк ASP MVC Core
6. Фреймворк для браузерних додатків Blazor wasm
7. Фреймворк для створення мобільних додатків Xamarin.Forms

СКБД MS SQL Server 2019 обрана за гарну сумісність, кросплатформеність (підтримується Linux та Windows), наявність безкоштовної версії, доступної для комерційного використання.

Програмна платформа .NET Core обрана як універсальна програмна платформа для серверної частини системи через кросплатформеність (Windows, Linux, MacOS), безкоштовність, наявність зрілих, розвинених, ефективних та зручних у використанні інструментів для всіх задач, які поставлені перед розробниками.

В якості мови програмування була обрана мова C# як розвинена об’єктно-орієнтована мова програмування, повністю підтримувана обраними платформами та технологіями.

Для взаємодії із базою даних була обрана ORM Entity Framework Core. Рішення про використання ORM, а не прямого програмування бази даних будо прийняте з міркувань уніфікації коду (уникнення використання другої мови програмування — SQL) та ефективного представлення сутностей бази даних у вигляді об'єктів у коді C#. ORM Entity Framework Core була обрана за добру сумісність з обраною платформою технологіями, обраними для інших компонентів системи та наявності потужного інструментарію для керування, створення та модифікації обраної СУБД

Веб-фреймворк ASP MVC Core обраний у якості технології веб серверу через ефективність, наявність інструментів для всіх задач проекту (створення API, аутентифікація та авторизація користувачів)

Означені програмні інструменти широко використовуються при розробці ПО. Компоненти ж обрані для клієнтської частини, напроти, є набагато менш вживаними в розробці.

Нативними мовами програмування для платформи Android є Java та Kotlin. Це розвинені мови з багатим інструментарієм та цілком сумісні із запланованими технологіями серверного API. Вибір на користь фреймворка Xamarin був зроблений виключно з метою уніфікації розробки, використання єдиної мови програмування (C#), повторного використання коду, використання нативних технологій валідації даних та локалізації платформи .NET. Платформа Xamarin.Forms (замість спеціалізовано платформи Xamarin.Android) буда обрана через можливість при потребі використати розроблений код для створення додатку для iOS.

Особливої уваги заслуговує вибір технології Blazor wasm для браузерного додатку. Враховуючи складність та багатокомпонентність клієнтського браузерного додатку було прийняте рішення розробляти браузерний додаток за ідеологією односторінкового додатку (SPA). Односторінковий додаток це веб-додаток або веб-сайт, який взаємодіє із сервером шляхом асинхронних запитів, динамічно переписуючи контент веб-сторінки замість повного її перезавантаження. Це наближує досвід користування браузерним додатком до досвіду користування десктопним додатком, зменшує мережевий трафік, період очікування отримання даних із серверу та покращує користувацький досвід завдяки тому, що користувач у період завантаження може продовжувати працювати із іншими компонентами додатку.

До недавнього часу єдиною платформою для створення односторінкових додатків та для браузерного програмування взагалі була мова JavaScript. Ця мова, з одного боку має потужний інструментарій для вирішення різноманітних завдань браузерного програмування, підтримується усіма сучасними бразуерами, для неї існують потужні фреймворкі для створення односторінкових додатків (Angular, Vue.js, React.js). С іншого боку вона має певні недоліки. В нашому випадку основним недоліком є те, що використання JavaScript призведе до включення до проекту ще однієї мови програмування із своїм стеком технологій, що зробить неможливим повторне використання коду серверної частини та мобільного клієнту у браузерному клієнті та викличе потребу дублювати значну частину вже розробленого коду іншою мовою програмування. Запобігти цьому ми можемо викоритсовуючи фреймворк Blazor.

Blazor це розроблений компанією Microsoft фреймворк, який дозволяє розробляти веб-додатки із застосуванням мови C# та комбінованого синтаксису Razor.

Існує два різновиди фреймворку Blazor, Blazor Server та Blazor Wasm. В фреймворку Blazor Server код C# виконується на сервері, а взаємодія із браузером відбувається дистанційно через технологію SignalR.

Для браузерного клієнту був обраний другий варіант — Blazor Wasm.

Сьогодні для браузерного програмування існує альтернатива JavaScript. Це стандарт WebAssembly або Wasm —портативний формат двоїчного коду для виконуваних програм та пов’язаний з ним асемблер. Стандарт був анонсований у 2015 році, у 2017 році відбулася перша демонстрація його можливостей, а в грудні 2019 року World Wide Web Consortium рекомендував його як обов’язковий у сучасних браузерах на додачу до HTML, CSS та JavaScript.

Завдяки стандарту WebAssembly з’явилася можливість виконувати у браузері скомпільований код і, таким чином, програмувати браузерні додатки іншими мовами програмування, крім JavaScript.

На сьогоднішній день Wasm підтримується усіма сучасними браузерами, крім Internet Explorer.

Фреймворк Blazor Wasm використовує стандарт WebAssembly для виконання скомпільованого JavaScript коду. Таким чином ми отримуємо всі переваги класичного JavaScript веб-додатку (мінімізація мережевого трафіку, покращання користувацького досвіду) отримуючи при цьому можливість повноцінного використання мови C# та інструментів платформи .NET.

В якості платформи програмування обрана операційна система Windows 10 та IDE Visual Studio 2019. IDE Visual Studio обрана заврядки потужній підтримці платформи .NET та мови C#, обрання операційної системи Windows 10 було вторинним по відношенню до вибору IDE, бо існуючі інструменти програмування для платформи .NET Core для Linux та MacOs, хоча і пропонують також багатий інструментарій, але все ж таки поступаються Visual Studio Windows.

**Архітектура системи**

Сама мета розробки та задачі, поставлені перед розробником диктують обрання класичної клієнт-серверної архітектури, в якій центральний компонент системи (сервер) здійснює взаємодію з клієнтами, зберігає дані, введені клієнтами, надає клієнтам доступ до цих даних, регулює права та можливості клієнтів по відношенню до центрального компоненту. З метою забезпечення безпеки, універсальності клієнтського інтерфейсу, виключення залежності клієнтських додатків від конкретної реалізації постачальника даних між базою даних та клієнтськими додатками використовується посередник у вигляді веб-служби.



Діаграма архітектури системи

Основою системи є постачальних даних, який зберігає дані про користувачів, тести та результати тестування та надає їх за запитами клієнтів. В рамках проекту у якості постачальника даних використовується реляційна база даних MS SQL Server.

Зв’язок між базою даних та клієнтськими додатками здійснює веб-служба, яка представляє клієнтам публічний API з архітектурою REST.

Включення веб-служби у шлях передачі даних від бази до клієнтів було зроблене з наступних міркувань:

* Безпека: єдиним клієнтом, який здійснює безпосереднє підключення до бази даних є веб-служба. Таким чином база даних не доступна в глобальній мережі, що значно знижує ризик несанкціонованого підключення до неї
* Відсутність залежності: при прямому підключенні клієнтів до бази даних, довелося б використовувати в них технології доступу: команди SQL або ORM. При підключенні через службу вся логіка з’єднання з базою даних знаходиться на сервері, клієнти ж отримують дані у вигляді готових об’єктів не маючи потреби в знаннях про структуру бази та перетворення записів у базі на об’єкти C#
* Гнучкість: публічний Api, який надає дані у форматі JSON може бути утилізований клієнтом на практично будь-якій сучасній технології і не потребує ніякого коду, специфічного для бази даних чи будь-якого іншого постачальника даних.

Зв’язок між базою даних та веб-службою здійснюється за допомогою ORM Entity Framework Core. Для розробки та створення бази даних був обраний підхід Code First, при якому сутності бази даних розробляються у вигляді класів C#, включаються в структуру класу контексту бази даних, за допомогою анотацій та Fluent Api вказується структура бази, та певні характеристики окремих полів і після цього за допомогою інструментів Entity Framework Core cli створюється клас міграції, який застосовується до бази даних або вручну, також за допомогою інструментів командного рядку, або автоматично при старті додатку, який утилізує клас контексту.

Такий підхід було обрано по-перше для того, щоб зберегти єдність програмування та використовувати в процесі розробки лише одну мову C# (запобігти використанню мови SQL). По друге, такий підхід найкраще підходить до обраної ітеративно-інкрементної моделі розробки, при якій за цикл розробляється певний набір взаємопов’язаних компонентів бази даних, служби та клієнту. Він дозволяє максимально зручно та ефективно поступово додавати до бази даних таблиці та модифікувати існуючі таблиці по мірі розробки.

Основою класу контексту бази даних є набір класів моделі, які визначають таблиці бази, їх поля та характеристики. Класи моделі виділені в окремий проект в рамках рішення для того, аби їх можна будо вільно використовувати у різних частинах проекту, що розробляється.

Клас контексту бази даних також виділений в окремих проект. Цей проект, крім класу контексту включає ще класи міграцій, які додаються по мірі розробки бази даних.

Для того, щоб запобігти жорсткій залежності веб-служби від постачальника даних, уможливити зміну постачальника без необхідності змінювати код веб-служби та полегшення тестування, було визначено інтерфейс репозіторію бази даних, який надає доступ до постачальника даних не розкриваючи його внутрішню реалізацію. Класи, які реалізують інтерфейс вносяться до класів веб-служби за допомогою механізму ін’єкції залежностей.

Веб-служба використовує технологію ASP MVC Core. На url web-служби надходять HTTP запити і веб-служба за допомогою проміжного модуля роутінгу, вбудованого у ASP MVC Core, на основі url запиту та типу запиту визначає, який клас та метод веб-служби мають обробити веб-запит. Параметри методів, якщо вони потрібні веб-служба також отримує із url або із тіла запиту. Дані в тілі запиту передаються у форматі JSON.

Веб-служба отримує запит, десеріалізує при необхідності отримані дані у об’єкт C# та робить запит до бази даних через посередництво класу-репозиторію, введеного до класу контроллера за допомогою ін’єкції залежностей. Репозиторій, в свою чергу, робить запит до бази даних за допомогою технології EF Core (в реалізації, що розробляється), отримує дані від бази даних, та передає їх до веб-служби. Отримавші ці дані, служба серіалізує їх та повертає клієнтові.

Таким чином досягається максимальне розділення задач. Завданням бази даних є збереження та відтворення даних за запитом. Вона не має ніяких відомостей про клієнтів та їх особливості. Завданням репозиторію є транстформація запитів мовою C# клієнтів до бази у команди бази даних та обратна трансформація відповідей бази. Завданням веб-служби є отримання веб-запитів від клієнтів, трансформація їх у об’єкти C#, виклик методів репозиторію, обратна трансформація відповідей репозиторію у веб-формат на надсилання відповідей клієнтам. Кожен рівень працює незалежно від особливостей реалізації своїх користувачів і може бути модифікований чи замінений при умові збереження публічного програмного інтерфейсу.

База даних та веб-служба можуть бути розміщені як на одному фізичному сервері так і на двох, пов’язаних між собою за допомогою мережі. Перше рішення має переваги з точки зору безпеки (відсутній зовнішній доступ до бази даних), друге з точки зору надійності та швидкості роботи системи.

Останньою ланкою архітектури проекту є клієнти. Розробляється два клієнта: браузерний клієнт та мобільний клієнт. Враховуючи те, що обрані технології дають можливість розробляти обидва клієнти мовою C#, то код, розроблений для серверної частини (модель даних) і стандартні вбудовані рішення для валідації даних та локалізації можуть бути використані в обох клієнтах.

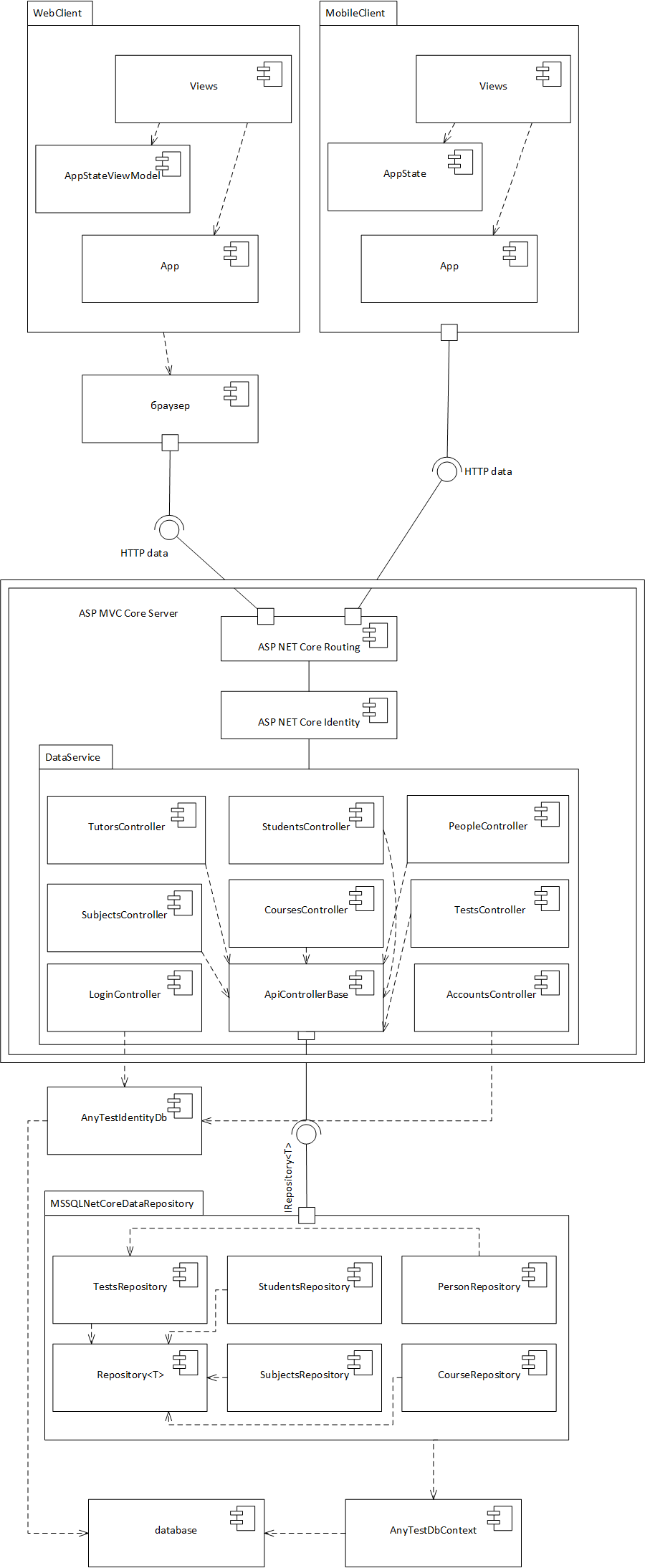
Основою обох клієнтів є графічний інтерфейс користувача, з яким користувач взаємодіє за допомогою клавіатури та миші або сенсорного екрану. Обидва клієнти побудовані за архітектурою MVVM, яка включає в себе модель даних (основану на моделі, створеній для бази даних), користувацький інтерфейс (вид) та програмну основу, яка пов’язує між собою інтерфейс та модель (ViewModel), використовуючи механізм «прив’язки», встановлюючи динамічні зв’язки між елементами інтерфейсу та певними класами моделі.

Взаємодія клієнтських додатків із центральним компонентом відбувається за допомогою запитів HTTP, які формуються моделлю представлення у відповідь на певні дії користувача. Дані запиту серіалізуютьс у формат JSON та відправляються на сервер. Відповідь на запит отримується також у форматі JSON, десеріалізується на використовується моделлю представлення для зміни стану користувацького інтерфейсу.

Окремо потрібно зупинитися на забезпеченні безпеки. До роботи в системі допускаються лише аутентифіковані та авторизовані користувачі. Кожен запит до центрального компоненту має бути аутентифікований даними конкретного користувача. Для аутентифікації та авторизації користувачів використовується технологія ASP MVC Core Identity, інтегрована у фреймфорк ASP MVC Core. Вона додає до конвеєру обробки HTTP запиту проміжний компонент, який перевіряє аутентифікацію користувачів та визначає, чи авторизовані вони для доступу до певного класу та методу контроллеру. Для зберігання облікових даних користувачів використовується окрема база даних, що підсилює безпеку та надійність аутентифікації і авторизації.

Методом ідентифікації користувачів було обрано JWT токени із обмеженим терміном дії. Вибір на користь JWT токенів було зроблено через їх універсальність: при бажанні їх легко можна використати із будь-яким типом клієнта не змінюючи серверний компонент, методики аутентифікації та авторизації і не порушуючи роботу раніше розроблених клієнтів.

**Розробка програмної підсистеми**



Структура програмної підсистеми

**Розробка та реалізація класів сутностей системи.**

У якості сутностей для використання в системі та для створення таблиці бази даних за принципом Code First були розроблені наступні основні класи моделі:

**public** class Person

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterFirstName**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**FirstName**))]**

**public** string FirstName **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterFamilyName**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**FamilyName**))]**

**public** string FamilyName **{** **get;** **set;** **}**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Patronimic**))]**

**public** string Patronimic **{** **get;** **set;** **}**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Phone**))]**

**public** string Phone **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterEmail**))]**

**[**EmailAddress**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EmailInvalid**))]**

**[**Display**(**Name **=** "e-mail"**)]**

**public** string Email **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Tutor Tutor **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Student Student **{** **get;** **set;** **}**

**}**

**public** class Tutor

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**None**)]**

**public** long Id **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**Id**))]**

**public** **virtual** Person Person **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**Test**>** TestsCreated **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**Course**>** CoursesCreated **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TutorTest**>** TestsOwned **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TutorCourse**>** CoursesOwned **{** **get;** **set;** **}**

**}**

**public** class Student

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**None**)]**

**public** long Id **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**Id**))]**

**public** **virtual** Person Person **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**StudentTest**>** Tests **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**StudentCourse**>** Courses **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestPass**>** Passes **{** **get;** **set;** **}**

**}**

**public** class Subject

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterName**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Name**))]**

**public** string Name **{** **get;** **set;** **}**

**public** bool Changed **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**Course**>** Courses **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestSubject**>** Tests **{** **get;** **set;** **}**

**}**

**public** class Course

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterName**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Name**))]**

**public** string Name **{** **get;** **set;** **}**

**public** long**?** SubjectId **{** **get;** **set;** **}**

**public** long AuthorId **{** **get;** **set;** **}**

**public** bool Changed **{** **get;** **set;** **}**

**[**Timestamp**]**

**public** byte**[]** RowVersion **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**SubjectId**))]**

**public** **virtual** Subject Subject **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**AuthorId**))]**

**public** **virtual** Tutor Author **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TutorCourse**>** Owners **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**StudentCourse**>** Students **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestCourse**>** Tests **{** **get;** **set;** **}**

**}**

**public** class Test

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterName**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Name**))]**

**public** **virtual** string Name **{** **get;** **set;** **}**

**public** long AuthorId **{** **get;** **set;** **}**

**public** bool Changed **{** **get;** **set;** **}**

**[**Timestamp**]**

**public** byte**[]** RowVerstion **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestQuestion**>** TestQuestions **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**AuthorId**))]**

**public** **virtual** Tutor Author **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TutorTest**>** Owners **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**StudentTest**>** Students **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestSubject**>** Subjects **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestCourse**>** Courses **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestPass**>** Passes **{** **get;** **set;** **}**

**}**

**public** class TestQuestion

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterQuestion**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Question**))]**

**public** **virtual** string Question **{** **get;** **set;** **}**

**[**Required**]**

**public** long TestId **{** **get;** **set;** **}**

**[**Required**]**

**public** int OrderNo **{** **get;** **set;** **}**

**[**Timestamp**]**

**public** byte**[]** RowVersion **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**[**ForeignKey**(**nameof**(**TestId**))]**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**TestAnswer**>** TestAnswers **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**AnswerPass**>** Answered **{** **get;** **set;** **}**

**}**

**public** class TestAnswer

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**(**ErrorMessageResourceType **=** **typeof(**Resources**),** ErrorMessageResourceName **=** nameof**(**Resources**.**EnterAnswer**))]**

**[**Display**(**ResourceType **=** **typeof(**Resources**),** Name **=** nameof**(**Resources**.**Answer**))]**

**public** **virtual** string Answer **{** **get;** **set;** **}**

**[**Range**(**0**,** 100**)]**

**public** **virtual** int Percent **{** **get;** **set;** **}**

**[**Required**]**

**public** int OrderNo **{** **get;** **set;** **}**

**[**Required**]**

**public** long TestQuestionId **{** **get;** **set;** **}**

**[**Timestamp**]**

**public** byte**[]** RowVersion **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**[**ForeignKey**(**nameof**(**TestQuestionId**))]**

**public** **virtual** TestQuestion TestQuestion **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**AnswerPass**>** Answered **{** **get;** **set;** **}**

**}**

**public** class TestPass

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**]**

**public** long TestId **{** **get;** **set;** **}**

**[**Required**]**

**public** long StudentId **{** **get;** **set;** **}**

**[**Required**]**

**public** DateTime PassedAt **{** **get;** **set;** **}**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**public** **virtual** ICollection**<**AnswerPass**>** Answers **{** **get;** **set;** **}**

**}**

**public** class TestSubject

**{**

**public** long TestId **{** **get;** **set;** **}**

**public** long SubjectId **{** **get;** **set;** **}**

**public** bool Deleted **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Subject Subject **{** **get;** **set;** **}**

**}**

**public** class AnswerPass

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**[**Required**]**

**public** long PassId **{** **get;** **set;** **}**

**[**Required**]**

**public** long QuestionId **{** **get;** **set;** **}**

**[**Required**]**

**public** long AnswerId **{** **get;** **set;** **}**

**public** **virtual** TestPass Pass **{** **get;** **set;** **}**

**public** **virtual** TestQuestion Question **{** **get;** **set;** **}**

**[**ForeignKey**(**nameof**(**AnswerId**))]**

**public** **virtual** TestAnswer Answer **{** **get;** **set;** **}**

**}**

**public** class TestCourse

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** long Id **{** **get;** **set;** **}**

**public** long TestId **{** **get;** **set;** **}**

**public** long CourseId **{** **get;** **set;** **}**

**public** uint OrderNo **{** **get;** **set;** **}**

**public** bool Deleted **{** **get;** **set;** **}**

**public** bool Restricted **{** **get;** **set;** **}**

**[**Range**(**0**,** 100**)]**

**public** int PassScore **{** **get;** **set;** **}** **=** 50**;**

**public** int TriesPermitted **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Course Course **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** ICollection**<**Precondition**>** Preconditions **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** ICollection**<**Precondition**>** Dependent **{** **get;** **set;** **}**

**}**

**public** class Precondition

**{**

**public** long TestId **{** **get;** **set;** **}**

**public** long CourseId **{** **get;** **set;** **}**

**public** long PreconditionId **{** **get;** **set;** **}**

**public** **virtual** TestCourse Test **{** **get;** **set;** **}**

**public** **virtual** TestCourse PreconditionTest **{** **get;** **set;** **}**

**}**

**public** class TutorCourse

**{**

**public** long TutorId **{** **get;** **set;** **}**

**public** long CourseId **{** **get;** **set;** **}**

**public** **virtual** Tutor Tutor **{** **get;** **set;** **}**

**public** **virtual** Course Course **{** **get;** **set;** **}**

**}**

**public** class TutorTest

**{**

**public** long TutorId **{** **get;** **set;** **}**

**public** long TestId **{** **get;** **set;** **}**

**public** **virtual** Tutor Tutor **{** **get;** **set;** **}**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**}**

**public** class StudentCourse

**{**

**public** long StudentId **{** **get;** **set;** **}**

**public** long CourseId **{** **get;** **set;** **}**

**public** bool Removed **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Student Student **{** **get;** **set;** **}**

**[**JsonIgnore**]**

**public** **virtual** Course Course **{** **get;** **set;** **}**

**}**

**public** class StudentTest

**{**

**public** long StudentId **{** **get;** **set;** **}**

**public** long TestId **{** **get;** **set;** **}**

**public** **virtual** Student Student **{** **get;** **set;** **}**

**public** **virtual** Test Test **{** **get;** **set;** **}**

**}**

Також для створення бази даних та додавання до бази властивостей, які неможливо задати за допомогою анотацій за допомогою Fluent Api створений таки клас контексту бази даних:

**public** class AnyTestDbContext **:** DbContext

**{**

**public** AnyTestDbContext**(**DbContextOptions**<**AnyTestDbContext**>** options**)** **:** **base(**options**)** **{** **}**

**public** **virtual** DbSet**<**Person**>** People **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Tutor**>** Tutors **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Student**>** Students **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Subject**>** Subjects **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Test**>** Tests **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Course**>** Courses **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TestQuestion**>** TestQuestions **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TestAnswer**>** TestAnswers **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TutorCourse**>** TutorCourses **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TutorTest**>** TutorTests **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**StudentTest**>** StudentTests **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**StudentCourse**>** StudentCourses **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TestSubject**>** TestSubjects **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TestCourse**>** TestCourses **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**Precondition**>** Preconditions **{** **get;** **set;** **}**

**public** **virtual** DbSet**<**TestPass**>** TestPasses **{** **get;** **set;** **}**

**public** DbSet**<**AnswerPass**>** AnswerPasses **{** **get;** **set;** **}**

**protected** **override** void OnConfiguring**(**DbContextOptionsBuilder optionsBuilder**)** **{** **}**

**protected** **override** void OnModelCreating**(**ModelBuilder model**)**

**{**

model**.**Entity**<**Person**>().**HasIndex**(**p **=>** p**.**Email**).**IsUnique**();**

model**.**Entity**<**TutorTest**>().**HasKey**(**tt **=>** **new** **{** tt**.**TutorId**,** tt**.**TestId **});**

model**.**Entity**<**TutorTest**>().**HasOne**(**tt **=>** tt**.**Tutor**).**WithMany**(**t **=>** t**.**TestsOwned**).**HasForeignKey**(**tt **=>** tt**.**TutorId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TutorTest**>().**HasOne**(**tt **=>** tt**.**Test**).**WithMany**(**t **=>** t**.**Owners**).**HasForeignKey**(**tt **=>** tt**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TutorCourse**>().**HasKey**(**tc **=>** **new** **{** tc**.**TutorId**,** tc**.**CourseId **});**

model**.**Entity**<**TutorCourse**>().**HasOne**(**tc **=>** tc**.**Tutor**).**WithMany**(**t **=>** t**.**CoursesOwned**).**HasForeignKey**(**tc **=>** tc**.**TutorId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TutorCourse**>().**HasOne**(**tc **=>** tc**.**Course**).**WithMany**(**c **=>** c**.**Owners**).**HasForeignKey**(**tc **=>** tc**.**CourseId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**StudentTest**>().**HasKey**(**st **=>** **new** **{** st**.**StudentId**,** st**.**TestId **});**

model**.**Entity**<**StudentTest**>().**HasOne**(**st **=>** st**.**Student**).**WithMany**(**s **=>** s**.**Tests**).**HasForeignKey**(**st **=>** st**.**StudentId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**StudentTest**>().**HasOne**(**st **=>** st**.**Test**).**WithMany**(**t **=>** t**.**Students**).**HasForeignKey**(**st **=>** st**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**StudentCourse**>().**HasKey**(**sc **=>** **new** **{** sc**.**StudentId**,** sc**.**CourseId **});**

model**.**Entity**<**StudentCourse**>().**HasOne**(**cs **=>** cs**.**Student**).**WithMany**(**s **=>** s**.**Courses**).**HasForeignKey**(**sc **=>** sc**.**StudentId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**StudentCourse**>().**HasOne**(**sc **=>** sc**.**Course**).**WithMany**(**c **=>** c**.**Students**).**HasForeignKey**(**sc **=>** sc**.**CourseId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestSubject**>().**HasKey**(**ts **=>** **new** **{** ts**.**TestId**,** ts**.**SubjectId **});**

model**.**Entity**<**TestSubject**>().**HasOne**(**ts **=>** ts**.**Test**).**WithMany**(**t **=>** t**.**Subjects**).**HasForeignKey**(**ts **=>** ts**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestSubject**>().**HasOne**(**ts **=>** ts**.**Subject**).**WithMany**(**s **=>** s**.**Tests**).**HasForeignKey**(**ts **=>** ts**.**SubjectId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestCourse**>().**HasOne**(**tc **=>** tc**.**Test**).**WithMany**(**t **=>** t**.**Courses**).**HasForeignKey**(**tc **=>** tc**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestCourse**>().**HasOne**(**tc **=>** tc**.**Course**).**WithMany**(**c **=>** c**.**Tests**).**HasForeignKey**(**tc **=>** tc**.**CourseId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestCourse**>().**HasIndex**(**tc **=>** tc**.**CourseId**);**

model**.**Entity**<**TestCourse**>().**Property**(**tc **=>** tc**.**PassScore**).**HasDefaultValue**(**50**);**

model**.**Entity**<**Precondition**>().**HasKey**(**p **=>** **new** **{** p**.**TestId**,** p**.**CourseId**,** p**.**PreconditionId **});**

model**.**Entity**<**Precondition**>().**HasOne**(**p **=>** p**.**Test**).**WithMany**(**t **=>** t**.**Preconditions**).**HasForeignKey**(**p **=>** p**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**Precondition**>().**HasOne**(**p **=>** p**.**PreconditionTest**).**WithMany**(**t **=>** t**.**Dependent**).**HasForeignKey**(**p **=>** p**.**PreconditionId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestPass**>().**HasOne**(**p **=>** p**.**Test**).**WithMany**(**t **=>** t**.**Passes**).**HasForeignKey**(**p **=>** p**.**TestId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**TestPass**>().**HasIndex**(**p **=>** p**.**StudentId**);**

model**.**Entity**<**AnswerPass**>().**HasOne**(**a **=>** a**.**Pass**).**WithMany**(**p **=>** p**.**Answers**).**HasForeignKey**(**a **=>** a**.**PassId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**AnswerPass**>().**HasOne**(**a **=>** a**.**Question**).**WithMany**(**q **=>** q**.**Answered**).**HasForeignKey**(**a **=>** a**.**QuestionId**).**OnDelete**(**DeleteBehavior**.**NoAction**);**

model**.**Entity**<**AnswerPass**>().**HasIndex**(**a **=>** a**.**PassId**);**

**}**

**}**

**Реалізація бази даних**

Після створення приведених класів, створення міграції за допомогою інструментів командного рядку EF Core, та застосування міграції до бази даних ми отримуємо наступну структура бази даних:



Структура бази даних

Таблиці:

|  |
| --- |
| [dbo].[People] |

Columns

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |
|  | First­Name | nvarchar(max) | max | NOT NULL |  |
|  | Family­Name | nvarchar(max) | max | NOT NULL |  |
|  | Phone | nvarchar(max) | max | NULL allowed |  |
|  | Email | nvarchar(450) | 900 | NOT NULL |  |
|  | Patronimic | nvarchar(max) | max | NULL allowed |  |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­People | Id | True |
|  | IX\_­People\_­Email | Email | True |

|  |
| --- |
| [dbo].[Tutors] |

Columns

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability |
|  | Id | bigint | 8 | NOT NULL |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Tutors | Id | True |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Tutors\_­People\_­Id | Cascade | Id->[[dbo].[People].[Id]](#56aDLBRUJLgVZYEphzQA6pzf9AY=) |

|  |
| --- |
| [dbo].[Students] |

Columns

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability |
|  | Id | bigint | 8 | NOT NULL |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Students | Id | True |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Students\_­People\_­Id | Cascade | Id->[[dbo].[People].[Id]](#56aDLBRUJLgVZYEphzQA6pzf9AY=) |

|  |
| --- |
| [dbo].[Subjects] |

Columns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity | Default |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |  |
|  | Name | nvarchar(max) | max | NOT NULL |  |  |
|  | Changed | bit | 1 | NOT NULL |  | (CONVERT([bit],(0))) |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Subjects | Id | True |

|  |
| --- |
| [dbo].[Courses] |

Columns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity | Default |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |  |
|  | Name | nvarchar(max) | max | NOT NULL |  |  |
|  | Subject­Id | bigint | 8 | NULL allowed |  |  |
|  | Author­Id | bigint | 8 | NOT NULL |  |  |
|  | Row­Version | timestamp | 8 | NULL allowed |  |  |
|  | Changed | bit | 1 | NOT NULL |  | (CONVERT([bit],(0))) |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Courses | Id | True |
|  | IX\_­Courses\_­Author­Id | Author­Id |  |
|  | IX\_­Courses\_­Subject­Id | Subject­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Courses\_­Subjects\_­Subject­Id |  | Subject­Id->[[dbo].[Subjects].[Id]](#8M6ey8t0rjYKrRWbl1nv6gIt3Yk=) |
| FK\_­Courses\_­Tutors\_­Author­Id | Cascade | Author­Id->[[dbo].[Tutors].[Id]](#U4sEBJbkTGlOC8/YQ/AsPNPHqwE=) |

|  |
| --- |
| [dbo].[Tests] |

Columns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity | Default |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |  |
|  | Name | nvarchar(max) | max | NOT NULL |  |  |
|  | Author­Id | bigint | 8 | NOT NULL |  |  |
|  | Row­Verstion | timestamp | 8 | NULL allowed |  |  |
|  | Changed | bit | 1 | NOT NULL |  | (CONVERT([bit],(0))) |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Tests | Id | True |
|  | IX\_­Tests\_­Author­Id | Author­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Tests\_­Tutors\_­Author­Id | Cascade | Author­Id->[[dbo].[Tutors].[Id]](#U4sEBJbkTGlOC8/YQ/AsPNPHqwE=) |

|  |
| --- |
| [dbo].[Test­Questions] |

Columns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity | Default |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |  |
|  | Question | nvarchar(max) | max | NOT NULL |  |  |
|  | Test­Id | bigint | 8 | NOT NULL |  |  |
|  | Row­Version | timestamp | 8 | NULL allowed |  |  |
|  | Order­No | int | 4 | NOT NULL |  | ((0)) |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Test­Questions | Id | True |
|  | IX\_­Test­Questions\_­Test­Id | Test­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Test­Questions\_­Tests\_­Test­Id | Cascade | Test­Id->[[dbo].[Tests].[Id]](#XNIg6KaWVxnMfBXdvtB1JFNPuL4=) |

|  |
| --- |
| [dbo].[Test­Answers] |

Columns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity | Default |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |  |
|  | Answer | nvarchar(max) | max | NOT NULL |  |  |
|  | Percent | int | 4 | NOT NULL |  |  |
|  | Test­Question­Id | bigint | 8 | NOT NULL |  |  |
|  | Row­Version | timestamp | 8 | NULL allowed |  |  |
|  | Order­No | int | 4 | NOT NULL |  | ((0)) |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Test­Answers | Id | True |
|  | IX\_­Test­Answers\_­Test­Question­Id | Test­Question­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Test­Answers\_­Test­Questions\_­Test­Question­Id | Cascade | Test­Question­Id->[[dbo].[Test­Questions].[Id]](#Qqow505sP8RpiA4pks8L93hVlzY=) |

|  |
| --- |
| [dbo].[Test­Passes] |

Columns

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |
|  | Test­Id | bigint | 8 | NOT NULL |  |
|  | Student­Id | bigint | 8 | NOT NULL |  |
|  | Passed­At | datetime2 | 8 | NOT NULL |  |

Indexes

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Test­Passes | Id | True |
|  | IX\_­Test­Passes\_­Student­Id | Student­Id |  |
|  | IX\_­Test­Passes\_­Test­Id | Test­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Test­Passes\_­Students\_­Student­Id | Cascade | Student­Id->[[dbo].[Students].[Id]](#d3aEVDAIMWyOlbbtGq8j0SZq5Hc=) |
| FK\_­Test­Passes\_­Tests\_­Test­Id |  | Test­Id->[[dbo].[Tests].[Id]](#XNIg6KaWVxnMfBXdvtB1JFNPuL4=) |

|  |
| --- |
| [dbo].[Answer­Passes] |

Columns

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability | Identity |
|  | Id | bigint | 8 | NOT NULL | 1 - 1 |
|  | Pass­Id | bigint | 8 | NOT NULL |  |
|  | Question­Id | bigint | 8 | NOT NULL |  |
|  | Answer­Id | bigint | 8 | NOT NULL |  |

Indexes

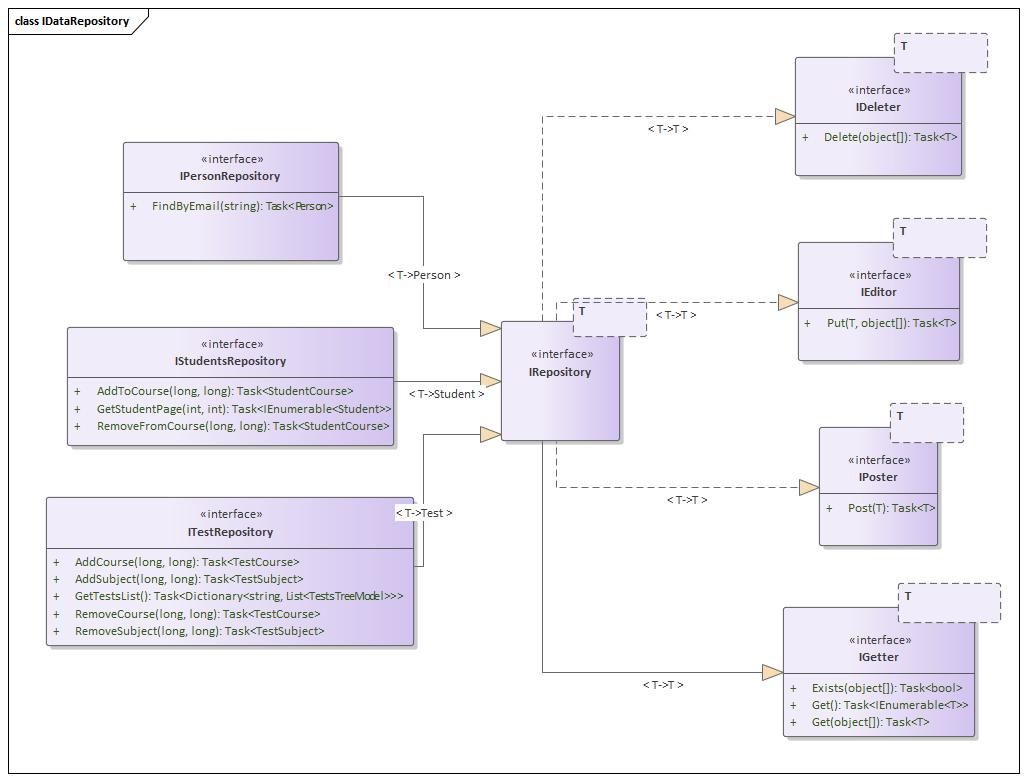
|  |  |  |  |
| --- | --- | --- | --- |
| Key | Name | Key Columns | Unique |
|  | PK\_­Answer­Passes | Id | True |
|  | IX\_­Answer­Passes\_­Answer­Id | Answer­Id |  |
|  | IX\_­Answer­Passes\_­Pass­Id | Pass­Id |  |
|  | IX\_­Answer­Passes\_­Question­Id | Question­Id |  |

Foreign Keys

|  |  |  |
| --- | --- | --- |
| Name | Delete | Columns |
| FK\_­Answer­Passes\_­Test­Answers\_­Answer­Id | Cascade | Answer­Id->[[dbo].[Test­Answers].[Id]](#x+pOhTFildfc365IDui37rCPHps=) |
| FK\_­Answer­Passes\_­Test­Passes\_­Pass­Id |  | Pass­Id->[[dbo].[Test­Passes].[Id]](#HEquW4jjl35v39mfRn1BNmCizPQ=) |
| FK\_­Answer­Passes\_­Test­Questions\_­Question­Id |  | Question­Id->[[dbo].[Test­Questions].[Id]](#Qqow505sP8RpiA4pks8L93hVlzY=) |

**Розробка та реалізація інтерфейсу доступу до бази даних**

З метою запобігання залежності компонентів системи взаємодія служби обміну даними з клієнтськими додатками із постачальником даних відбувається через абстрактний інтерфейс. Основою інтерфейсу є генерічний клас IRepository<T>, який надає основні методи операції з даними. Від цього класу успадковують класи, які додають до нього специфічні методи, необхідні для роботи з конкретними сутностями бази даних. Інтерфейсом передбачений асінхроний доступ до постачальника, тому усі методи інтерфейсу повертають об’єкти класу Task. З метою моживості гнучкого налаштування різних компонентів інтерфейсу, він розбитий на кілька абстрактний інтерфейсів, які описують окремі методи.



Діаграма класів інтерфейсу доступу до постачальника даних.

Похідний код інтерфейсу:

**public** **interface** IGetter**<**T**>**

**{**

Task**<**IEnumerable**<**T**>>** Get**();**

Task**<**T**>** Get**(params** **object[]** key**);**

Task**<**bool**>** Exists**(params** **object[]** key**);**

**}**

**public** **interface** IPoster**<**T**>**

**{**

Task**<**T**>** Post**(**T item**);**

**}**

**public** **interface** IEditor**<**T**>**

**{**

Task**<**T**>** Put**(**T item**,** **params** **object[]** key**);**

**}**

**public** **interface** IDeleter**<**T**>**

**{**

Task**<**T**>** Delete**(params** **object[]** key**);**

**}**

**public** **interface** IRepository**<**T**>:**IGetter**<**T**>,** IPoster**<**T**>,** IEditor**<**T**>,** IDeleter**<**T**>**

**{**

**}**

**public** **interface** IPersonRepository **:** IRepository**<**Person**>**

**{**

Task**<**Person**>** FindByEmail**(**string email**);**

**}**

**public** **interface** IStudentsRepository**:**IRepository**<**Student**>**

**{**

Task**<**IEnumerable**<**Student**>>** GetStudentPage**(**int pageNubmer**,** int pageSize**);**

Task**<**StudentCourse**>** AddToCourse**(**long studentId**,** long courseId**);**

Task**<**StudentCourse**>** RemoveFromCourse**(**long studentId**,** long courseId**);**

**}**

**public** **interface** ITestRepository **:** IRepository**<**Test**>**

**{**

**public** Task**<**Dictionary**<**string**,** List**<**TestsTreeModel**>>>** GetTestsList**();**

**public** Task**<**TestSubject**>** AddSubject**(**long testId**,** long subjectId**);**

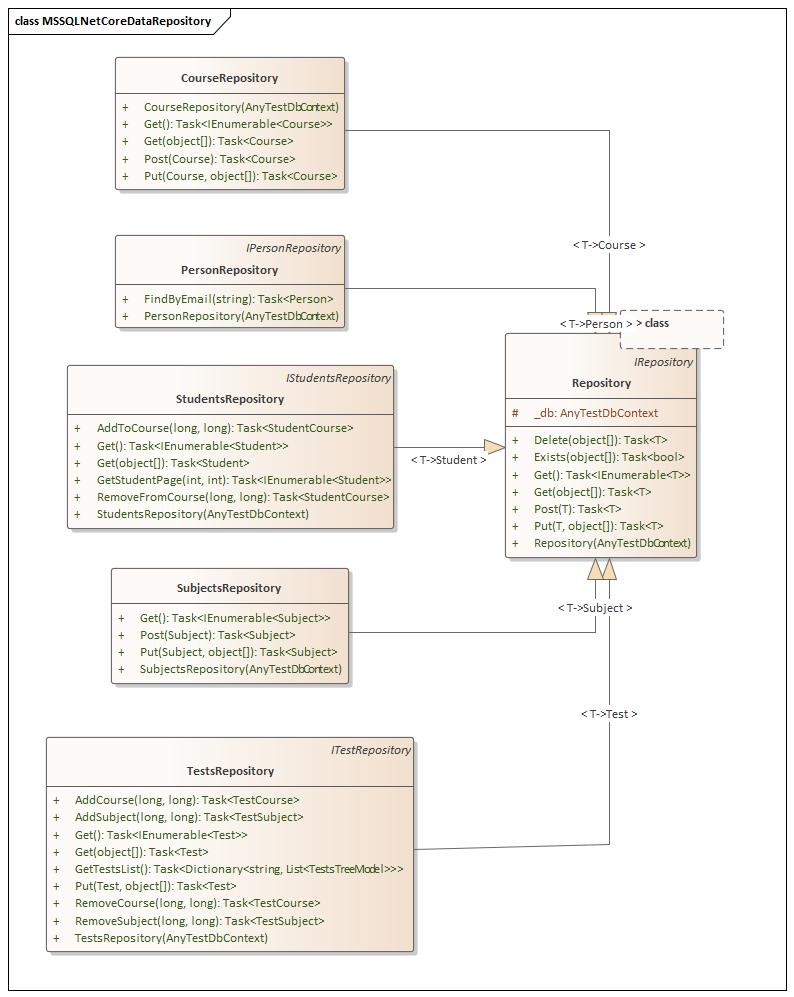
**public** Task**<**TestSubject**>** RemoveSubject**(**long testId**,** long subjectId**);**

**public** Task**<**TestCourse**>** AddCourse**(**long testId**,** long courseId**);**

**public** Task**<**TestCourse**>** RemoveCourse**(**long testId**,** long courseId**);**

**}**

Далі ми визначаємо конкретну реалізацію інтерфейсів, в яких постачальником даних виступає СКБД MS SQL Server і дані отримуються за допомогою ORM фреймворку EF Core. Для реалізації ми також визначаємо клас-генерік Repository<T>, класи, які успадковують від нього з метою специфічної реалізації певних методів та класи, які успадковують від нього і від інтерфейсів, успадковуючих від IRepository<T>. Структура класів реалізації інтерфейсу має наступний вигляд:



Конкретна реалізація інтерфейсу визначається наступними класами:

**public** class Repository**<**T**>** **:** IRepository**<**T**>** **where** T**:**class

**{**

**protected** AnyTestDbContext \_db**;**

**public** Repository**(**AnyTestDbContext db**)** **=>** \_db **=** db**;**

**public** **virtual** **async** Task**<**T**>** Delete**(params** **object[]** key**)**

**{**

T item **=** **await** \_db**.**Set**<**T**>().**FindAsync**(**key**);**

**if(**item **!=** **null)**

**{**

\_db**.**Remove**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

**}**

**return** item**;**

**}**

**public** **virtual** **async** Task**<**bool**>** Exists**(params** **object[]** key**)**

**{**

var found **=** **await** \_db**.**Set**<**T**>().**FindAsync**(**key**);**

**if(**found **==** **null)**

**{**

**return** **false;**

**}**

**else**

**{**

\_db**.**Entry**<**T**>(**found**).**State **=** EntityState**.**Detached**;**

**return** **true;**

**}**

**}**

**public** **virtual** **async** Task**<**IEnumerable**<**T**>>** Get**()**

**{**

**return** **await** \_db**.**Set**<**T**>().**AsNoTracking**().**ToListAsync**();**

**}**

**public** **virtual** **async** Task**<**T**>** Get**(params** **object[]** key**)**

**{**

var item **=** **await** \_db**.**Set**<**T**>().**FindAsync**(**key**);**

**if(**item **!=** **null)**

**{**

\_db**.**Entry**<**T**>(**item**).**State **=** EntityState**.**Detached**;**

**}**

**return** item**;**

**}**

**public** **virtual** **async** Task**<**T**>** Post**(**T item**)**

**{**

\_db**.**Add**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

\_db**.**Entry**(**item**).**State **=** EntityState**.**Detached**;**

**return** item**;**

**}**

**public** **virtual** **async** Task**<**T**>** Put**(**T item**,** **params** **object[]** key**)**

**{**

\_db**.**Update**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

\_db**.**Entry**(**item**).**State **=** EntityState**.**Detached**;**

**return** item**;**

**}**

**}**

**public** class PersonRepository **:** Repository**<**Person**>,** IPersonRepository

**{**

**public** PersonRepository**(**AnyTestDbContext db**)** **:** **base(**db**)**

**{**

**}**

**public** **async** Task**<**Person**>** FindByEmail**(**string email**)** **=>** **await** \_db**.**Set**<**Person**>().**AsNoTracking**().**SingleOrDefaultAsync**(**p **=>** p**.**Email **==** email**);**

**}**

**public** class SubjectsRepository **:** Repository**<**Subject**>**

**{**

**public** SubjectsRepository**(**AnyTestDbContext db**)** **:** **base(**db**)**

**{**

**}**

**public** **override** **async** Task**<**IEnumerable**<**Subject**>>** Get**()** **=>**

**await** \_db**.**Subjects**.**Where**(**s **=>** **!**s**.**Changed**)**

**.**Include**(**s **=>** s**.**Tests**)**

**.**AsNoTracking**().**ToListAsync**();**

**public** **override** **async** Task**<**Subject**>** Post**(**Subject item**)** **=>** **await** **base.**Put**(**item**);**

**public** **override** **async** Task**<**Subject**>** Put**(**Subject item**,** **params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

var old **=** **await** \_db**.**Subjects**.**FindAsync**(**id**);**

**if** **(**old **==** **null)** **throw** **new** ArgumentException**(**"No subject with such id"**);**

old**.**Changed **=** **true;**

\_db**.**Update**(**old**);**

item**.**Id **=** 0**;**

\_db**.**Add**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

**return** item**;**

**}**

**else** **throw** **new** ArgumentException**(**"Subject id must be of type long"**);**

**}**

**}**

**public** class CourseRepository **:** Repository**<**Course**>**

**{**

**public** CourseRepository**(**AnyTestDbContext db**)** **:** **base(**db**)**

**{**

**}**

**public** **override** **async** Task**<**IEnumerable**<**Course**>>** Get**()** **=>** **await** \_db**.**Courses

**.**Where**(**c **=>** **!**c**.**Changed**)**

**.**Include**(**c **=>** c**.**Tests**)**

**.**Include**(**c **=>** c**.**Owners**)**

**.**AsNoTracking**().**ToListAsync**();**

**public** **override** **async** Task**<**Course**>** Get**(params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

**return** **await** \_db**.**Courses**.**Where**(**c **=>** **!**c**.**Changed**)**

**.**Include**(**c **=>** c**.**Tests**)**

**.**Include**(**c **=>** c**.**Owners**)**

**.**AsNoTracking**().**SingleOrDefaultAsync**(**c **=>** c**.**Id **==** id**);**

**}**

**else** **throw** **new** ArgumentException**(**"Course Id must be of type long"**);**

**}**

**public** **override** **async** Task**<**Course**>** Post**(**Course item**)** **=>** **await** **base.**Put**(**item**);**

**public** **override** **async** Task**<**Course**>** Put**(**Course item**,** **params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

var old **=** **await** \_db**.**Courses**.**FindAsync**(**id**);**

**if** **(**old **==** **null)** **throw** **new** ArgumentException**(**"No course with such id"**);**

old**.**Changed **=** **true;**

\_db**.**Update**(**old**);**

item**.**Id **=** 0**;**

\_db**.**Add**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

**return** item**;**

**}**

**else** **throw** **new** ArgumentException**(**"Course id must be of type long"**);**

**}**

**}**

**public** class StudentsRepository **:** Repository**<**Student**>,** IStudentsRepository

**{**

**public** StudentsRepository**(**AnyTestDbContext db**)** **:** **base(**db**)**

**{**

**}**

**public** **override** **async** Task**<**IEnumerable**<**Student**>>** Get**()** **=>** **await** \_db**.**Students

**.**Include**(**s **=>** s**.**Person**)**

**.**Include**(**s **=>** s**.**Courses**)**

**.**AsNoTracking**().**ToListAsync**();**

**public** **override** **async** Task**<**Student**>** Get**(params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

**return** **await** \_db**.**Students

**.**Include**(**s **=>** s**.**Person**)**

**.**Include**(**s **=>** s**.**Courses**)**

**.**AsNoTracking**().**SingleOrDefaultAsync**(**s **=>** s**.**Id **==** id**);**

**}**

**else**

**throw** **new** ArgumentException**(**"Student Id must be of type long"**);**

**}**

**public** **async** Task**<**IEnumerable**<**Student**>>** GetStudentPage**(**int pageNubmer**,** int pageSize**)** **=>**

**await** \_db**.**Students**.**Include**(**s **=>** s**.**Person**)**

**.**Include**(**s **=>** s**.**Courses**)**

**.**OrderBy**(**s **=>** s**.**Person**.**FirstName**)**

**.**Skip**(**pageSize **\*** **(**pageNubmer **-** 1**))**

**.**Take**(**pageSize**)**

**.**AsNoTracking**().**ToListAsync**();**

**public** **async** Task**<**StudentCourse**>** AddToCourse**(**long studentId**,** long courseId**)**

**{**

var studentCourse **=** \_db**.**StudentCourses**.**Find**(**studentId**,** courseId**);**

**if(**studentCourse **==** **null)**

**{**

studentCourse **=** **new** StudentCourse **{** StudentId **=** studentId**,** CourseId **=** courseId **};**

\_db**.**Add**(**studentCourse**);**

**}**

**else** **if(**studentCourse**.**Removed**)**

**{**

studentCourse**.**Removed **=** **false;**

\_db**.**Update**(**studentCourse**);**

**}**

**await** \_db**.**SaveChangesAsync**();**

\_db**.**Entry**(**studentCourse**).**State **=** EntityState**.**Detached**;**

**return** studentCourse**;**

**}**

**public** **async** Task**<**StudentCourse**>** RemoveFromCourse**(**long studentId**,** long courseId**)**

**{**

var studentCoures **=** \_db**.**StudentCourses**.**Find**(**studentId**,** courseId**);**

studentCoures**.**Removed **=** **true;**

\_db**.**Update**(**studentCoures**);**

**await** \_db**.**SaveChangesAsync**();**

**return** studentCoures**;**

**}**

**}**

**public** class TestsRepository **:** Repository**<**Test**>,** ITestRepository

**{**

**public** TestsRepository**(**AnyTestDbContext db**)** **:** **base(**db**)**

**{**

**}**

**public** **override** **async** Task**<**IEnumerable**<**Test**>>** Get**()** **=>**

**await** \_db**.**Tests**.**Include**(**t **=>** t**.**Subjects**)**

**.**Where**(**t **=>** **!**t**.**Changed**)**

**.**Include**(**t **=>** t**.**Courses**)**

**.**Include**(**t **=>** t**.**TestQuestions**).**ThenInclude**(**q **=>** q**.**TestAnswers**)**

**.**AsNoTracking**().**ToListAsync**();**

**public** **override** **async** Task**<**Test**>** Get**(params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

**return** **await** \_db**.**Tests

**.**Include**(**t **=>** t**.**Subjects**)**

**.**Include**(**t **=>** t**.**Courses**)**

**.**Include**(**t **=>** t**.**TestQuestions**).**ThenInclude**(**q **=>** q**.**TestAnswers**)**

**.**AsNoTracking**().**SingleOrDefaultAsync**(**t **=>** t**.**Id **==** id**);**

**}**

**else** **throw** **new** ArgumentException**(**"Test Id must be of type long"**);**

**}**

**public** **override** **async** Task**<**Test**>** Put**(**Test item**,** **params** **object[]** key**)**

**{**

**if** **(**key**.**Length **>** 0 **&&** key**[**0**]** **is** long id**)**

**{**

var old **=** **await** \_db**.**Tests**.**FindAsync**(**id**);**

**if** **(**old **==** **null)** **throw** **new** ArgumentException**(**"No test with such id"**);**

old**.**Changed **=** **true;**

\_db**.**Update**(**old**);**

item**.**Id **=** 0**;**

**foreach** **(**var question **in** item**.**TestQuestions**)**

**{**

question**.**Id **=** 0**;**

question**.**TestId **=** 0**;**

**foreach(**var answer **in** question**.**TestAnswers**)**

**{**

answer**.**Id **=** 0**;**

answer**.**TestQuestionId **=** 0**;**

**}**

**}**

**foreach(**var subject **in** item**.**Subjects**)** subject**.**TestId **=** 0**;**

**foreach(**var course **in** item**.**Courses**)** course**.**TestId **=** 0**;**

\_db**.**Add**(**item**);**

**await** \_db**.**SaveChangesAsync**();**

**return** item**;**

**}**

**else** **throw** **new** ArgumentException**(**"Test id must be of type long"**);**

**}**

**public** **async** Task**<**Dictionary**<**string**,** List**<**TestsTreeModel**>>>** GetTestsList**()**

**{**

var result **=** **new** Dictionary**<**string**,** List**<**TestsTreeModel**>>**

**{**

**{**"subjects"**,** **new** List**<**TestsTreeModel**>()** **},**

**{**"courses"**,** **new** List**<**TestsTreeModel**>()** **},**

**{**"tests"**,** **new** List**<**TestsTreeModel**>()** **}**

**};**

var subjects **=** **await** \_db**.**Subjects**.**Where**(**s **=>** **!**s**.**Changed**).**Include**(**s **=>** s**.**Courses**).**ThenInclude**(**c **=>** c**.**Tests**).**ThenInclude**(**t **=>** t**.**Test**)**

**.**Include**(**s **=>** s**.**Tests**).**ThenInclude**(**t **=>** t**.**Test**).**AsNoTracking**().**ToListAsync**();**

result**[**"subjects"**]** **=** subjects**.**Select**(**s **=>** **new** TestsTreeModel

**{**

Id **=** s**.**Id**,**

Name **=** s**.**Name**,**

Url **=** $"subjects/{s.Id}"**,**

Children **=** s**.**Courses**.**Select**(**c **=>** **new** TestsTreeModel

**{**

Id **=** c**.**Id**,**

Name **=** c**.**Name**,**

Url **=** $"courses/{c.Id}"**,**

Children **=** c**.**Tests**.**Select**(**t **=>** **new** TestsTreeModel

**{**

Id **=** t**.**TestId**,**

Name **=** t**.**Test**.**Name**,**

Url **=** $"tests/{t.TestId}"

**})**

**}).**Concat**(**s**.**Tests**.**Where**(**t **=>** **!**s**.**Courses**.**SelectMany**(**c **=>** c**.**Tests**).**Any**(**tc **=>** tc**.**TestId **==** t**.**TestId**)).**Select**(**tc **=>** **new** TestsTreeModel

**{**

Id **=** tc**.**TestId**,**

Name **=** tc**.**Test**.**Name**,**

Url **=** $"tests/{tc.TestId}"

**}))**

**}).**ToList**();**

var courses **=** **await** \_db**.**Courses**.**Where**(**c **=>** **!**c**.**Changed**).**Include**(**c **=>** c**.**Tests**).**ThenInclude**(**t **=>** t**.**Test**).**AsNoTracking**().**ToListAsync**();**

result**[**"courses"**]** **=** courses**.**Select**(**c **=>** **new** TestsTreeModel

**{**

Id **=** c**.**Id**,**

Name **=** c**.**Name**,**

Url **=** $"courses/{c.Id}"**,**

Children **=** c**.**Tests**.**Select**(**t **=>** **new** TestsTreeModel

**{**

Id **=** t**.**TestId**,**

Name **=** t**.**Test**.**Name**,**

Url **=** $"tests/{t.TestId}"

**})**

**}).**ToList**();**

var tests **=** **await** \_db**.**Tests**.**Where**(**t **=>** **!**t**.**Changed**).**AsNoTracking**().**ToListAsync**();**

result**[**"tests"**]** **=** tests**.**Select**(**t **=>** **new** TestsTreeModel

**{**

Id **=** t**.**Id**,**

Name **=** t**.**Name**,**

Url **=** $"tests/{t.Id}"

**}).**ToList**();**

**return** result**;**

**}**

**public** **async** Task**<**TestSubject**>** AddSubject**(**long testId**,** long subjectId**)**

**{**

var testSubject **=** \_db**.**TestSubjects**.**Find**(**testId**,** subjectId**);**

**if(**testSubject **==** **null)**

**{**

testSubject **=** **new** TestSubject **{** TestId **=** testId**,** SubjectId **=** subjectId **};**

\_db**.**Add**(**testSubject**);**

**await** \_db**.**SaveChangesAsync**();**

**}**

**else** **if** **(**testSubject**.**Deleted**)**

**{**

testSubject**.**Deleted **=** **false;**

\_db**.**Update**(**testSubject**);**

**await** \_db**.**SaveChangesAsync**();**

**}**

\_db**.**Entry**(**testSubject**).**State **=** EntityState**.**Detached**;**

**return** testSubject**;**

**}**

**public** **async** Task**<**TestSubject**>** RemoveSubject**(**long testId**,** long subjectId**)**

**{**

var testSubject **=** \_db**.**TestSubjects**.**Find**(**testId**,** subjectId**);**

**if(**testSubject **!=** **null)**

**{**

testSubject**.**Deleted **=** **true;**

\_db**.**Update**(**testSubject**);**

**await** \_db**.**SaveChangesAsync**();**

\_db**.**Entry**(**testSubject**).**State **=** EntityState**.**Detached**;**

**}**

**return** testSubject**;**

**}**

**public** **async** Task**<**TestCourse**>** AddCourse**(**long testId**,** long courseId**)**

**{**

var testCourse **=** \_db**.**TestCourses**.**Find**(**testId**,** courseId**);**

**if** **(**testCourse **==** **null)**

**{**

testCourse **=** **new** TestCourse **{** TestId **=** testId**,** CourseId **=** courseId **};**

\_db**.**Add**(**testCourse**);**

**await** \_db**.**SaveChangesAsync**();**

**}**

**else** **if** **(**testCourse**.**Deleted**)**

**{**

testCourse**.**Deleted **=** **false;**

\_db**.**Update**(**testCourse**);**

**await** \_db**.**SaveChangesAsync**();**

**}**

\_db**.**Entry**(**testCourse**).**State **=** EntityState**.**Detached**;**

**return** testCourse**;**

**}**

**public** **async** Task**<**TestCourse**>** RemoveCourse**(**long testId**,** long courseId**)**

**{**

var testCourse **=** \_db**.**TestCourses**.**Find**(**testId**,** courseId**);**

**if** **(**testCourse **!=** **null)**

**{**

testCourse**.**Deleted **=** **true;**

\_db**.**Update**(**testCourse**);**

**await** \_db**.**SaveChangesAsync**();**

\_db**.**Entry**(**testCourse**).**State **=** EntityState**.**Detached**;**

**}**

**return** testCourse**;**

**}**

**}**

Об’єкт контексту бази даних є залежністю класу Repository<T> та споживається цим класом за механізмом ін’єкції залежності.

**Розробка та реалізація веб-служби**

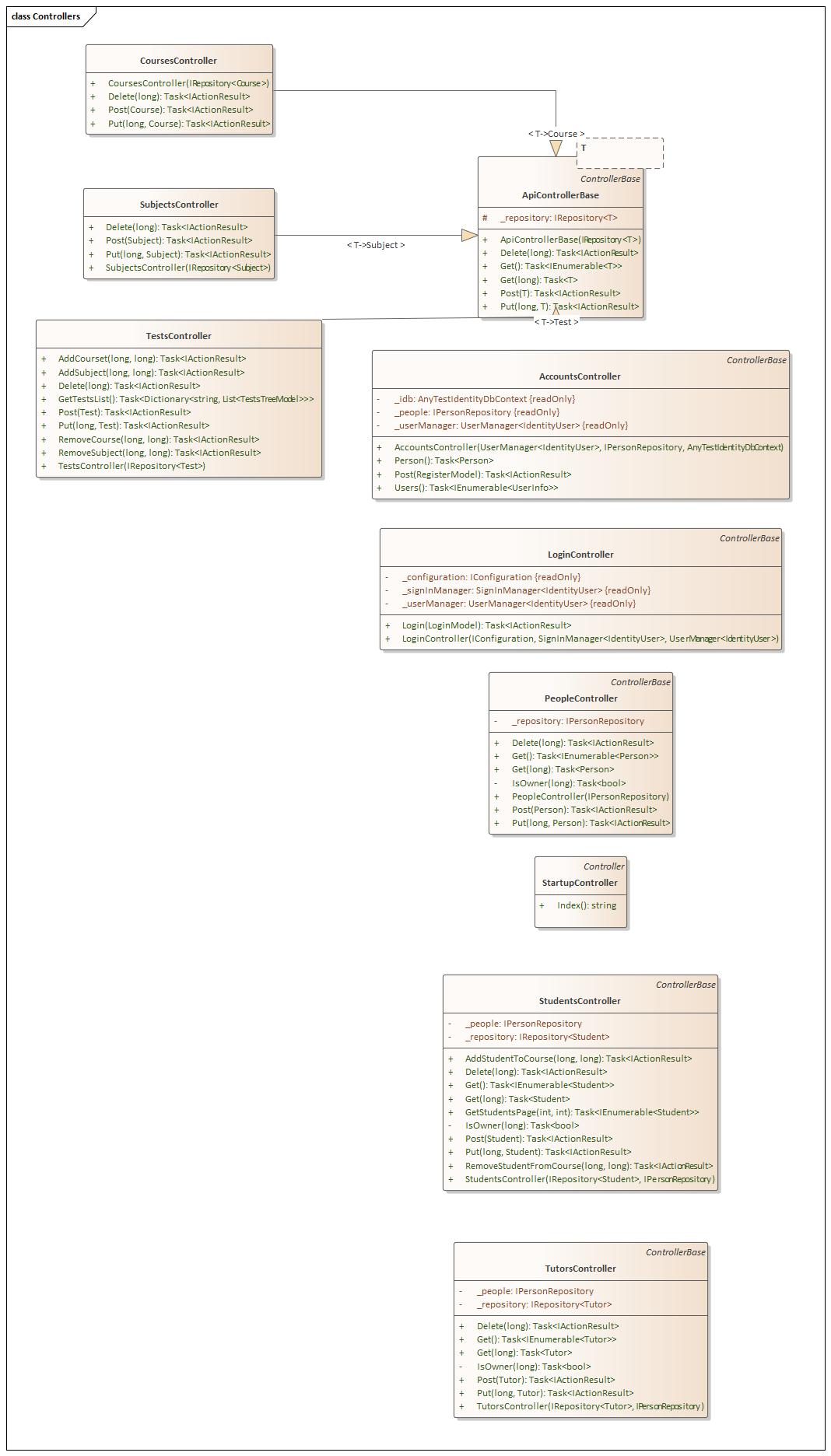
Завданням веб-служби є отримання HTTP-запитів від клієнтський додатків, обробка їх та надання клієнтським додаткам потрібних даних від постачальника даних. Також веб-служба відповідає за аутентифікацію, авторизацію користувачів та контроль доступу користувачів до даних та функцій.

Веб служба складається із серверу, який отримує запити від користувачів та передає їх до веб-служби, проміжних компонентів (middleware), які виконують попередній аналіз та обробку запитів, зокрема аутентифікацію та авторизацію користувачів, розподіленя запитів по класам та методам контроллерів та десеріалізацію даних із запитів в об’єкти C#.



Діаграма компонентів веб-служби

Тому що кожен контролер надає стандартний набір методів для отримання та зміни даних в базі, утилізуючи методи інтерфейсу репозиторію був створений абстрактний клас стандартної утилізації інтерфейсу, від якого успадковують всі контролери базової моделі додатку крім контролерів, які відповідають за вхід користувача у систему



Діаграма класів контролерів веб-служби

Класи контролерів були розроблені наступним чином:

**[**ApiController**]**

**public** class ApiControllerBase**<**T**>** **:** ControllerBase

**{**

**protected** IRepository**<**T**>** \_repository**;**

**public** ApiControllerBase**(**IRepository**<**T**>** repository**)** **=>** \_repository **=** repository**;**

**[**HttpGet**]**

**public** **virtual** **async** Task**<**IEnumerable**<**T**>>** Get**()** **=>** **await** \_repository**.**Get**();**

**[**HttpGet**(**"{id:long}"**)]**

**public** **virtual** **async** Task**<**T**>** Get**(**long id**)** **=>** **await** \_repository**.**Get**(**id**);**

**[**HttpPost**]**

**public** **virtual** **async** Task**<**IActionResult**>** Post**([**FromBody**]**T item**)**

**{**

**if** **(!**ModelState**.**IsValid**)**

**{**

**return** **(**BadRequest**(**ModelState**));**

**}**

**return** Ok**(await** \_repository**.**Post**(**item**));**

**}**

**[**HttpPut**(**"{id:long}"**)]**

**public** **virtual** **async** Task**<**IActionResult**>** Put**(**long id**,** **[**FromBody**]** T item**)**

**{**

**if** **(!**ModelState**.**IsValid**)**

**{**

**return** BadRequest**(**ModelState**);**

**}**

**if** **(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"Tutor does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Put**(**item**,** id**));**

**}**

**[**HttpDelete**(**"{id:long}"**)]**

**public** **virtual** **async** Task**<**IActionResult**>** Delete **(**long id**)**

**{**

**if** **(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"User does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Delete**(**id**));**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** class PeopleController **:** ControllerBase

**{**

**private** IPersonRepository \_repository**;**

**public** PeopleController**(**IPersonRepository repository**)** **=>** \_repository **=** repository**;**

**[**HttpGet**]**

**[**Authorize**(**Roles**=**"Administrator"**)]**

**public** **async** Task**<**IEnumerable**<**Person**>>** Get**()** **=>** **await** \_repository**.**Get**();**

**[**HttpGet**(**"{id:long}"**,** Name **=** "Get"**)]**

**[**Authorize**]**

**public** **async** Task**<**Person**>** Get**(**long id**)** **=>** **await** \_repository**.**Get**(**id**);**

**[**HttpPost**]**

**public** **async** Task**<**IActionResult**>** Post**([**FromBody**]** Person person**)**

**{**

**if(!**ModelState**.**IsValid**)**

**{**

**return** BadRequest**(**ModelState**);**

**}**

**return** Ok**(await** \_repository**.**Post**(**person**));**

**}**

**[**HttpPut**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**IActionResult**>** Put**(**long id**,** **[**FromBody**]** Person person**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!**ModelState**.**IsValid**)**

**{**

**return** BadRequest**(**ModelState**);**

**}**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"User does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Put**(**person**));**

**}**

**[**HttpDelete**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**IActionResult**>** Delete**(**long id**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"User does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Delete**(**id**));**

**}**

**private** **async** Task**<**bool**>** IsOwner**(**long id**)**

**{**

var userEmail **=** **(**HttpContext**.**User**.**Identity **as** ClaimsIdentity**).**FindFirst**(**ClaimTypes**.**Email**).**Value**;**

var personEmail **=** **(await** \_repository**.**Get**(**id**)).**Email**;**

**return** userEmail **==** personEmail**;**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**[**Authorize**]**

**public** class SubjectsController **:** ApiControllerBase**<**Subject**>**

**{**

**public** SubjectsController**(**IRepository**<**Subject**>** repository**)** **:** **base(**repository**)** **{** **}**

**[**Authorize**(**Roles **=**"Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Post**(**Subject item**)** **=>** **await** **base.**Post**(**item**);**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Put**(**long id**,** Subject item**)** **=>** **await** **base.**Put**(**id**,** item**);**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Delete**(**long id**)** **=>** **await** **base.**Delete**(**id**);**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**[**Authorize**]**

**public** class CoursesController **:** ApiControllerBase**<**Course**>**

**{**

**public** CoursesController**(**IRepository**<**Course**>** repository**)** **:** **base(**repository**)** **{** **}**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Post**(**Course item**)** **=>** **await** **base.**Post**(**item**);**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Put**(**long id**,** Course item**)** **=>** **await** **base.**Put**(**id**,** item**);**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Delete**(**long id**)** **=>** **await** **base.**Delete**(**id**);**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**[**Authorize**]**

**public** class TestsController **:** ApiControllerBase**<**Test**>**

**{**

**public** TestsController**(**IRepository**<**Test**>** repository**)** **:** **base(**repository**)**

**{**

**}**

**[**Authorize**(**Roles **=** "Administratir, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Post**([**FromBody**]** Test item**)** **=>** **await** **base.**Post**(**item**);**

**[**Authorize**(**Roles **=**"Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Put**(**long id**,** **[**FromBody**]** Test item**)** **=>** **await** **base.**Put**(**id**,** item**);**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **override** **async** Task**<**IActionResult**>** Delete**(**long id**)** **=>** **await** **base.**Delete**(**id**);**

**[**HttpGet**(**"list"**)]**

**public** **async** Task**<**Dictionary**<**string**,** List**<**TestsTreeModel**>>>** GetTestsList**()** **=>** **await** **(**\_repository **as** ITestRepository**).**GetTestsList**();**

**[**HttpPost**(**"{testId:long}/subjects/{subjectId:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **async** Task**<**IActionResult**>** AddSubject**(**long testId**,** long subjectId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**testId**))** **return** BadRequest**(**"Such test does not exist"**);**

**return** Ok**(await** **(**\_repository **as** ITestRepository**).**AddSubject**(**testId**,** subjectId**));**

**}**

**[**HttpDelete**(**"{testId:long}/subjects/{subjectId:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **async** Task**<**IActionResult**>** RemoveSubject**(**long testId**,** long subjectId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**testId**))** **return** BadRequest**(**"Such test does not exist"**);**

var deleted **=** **await** **(**\_repository **as** ITestRepository**).**RemoveSubject**(**testId**,** subjectId**);**

**if** **(!**deleted**.**Deleted**)** **return** BadRequest**(**"The test does not belong to such subject"**);**

**return** Ok**(**deleted**);**

**}**

**[**HttpPost**(**"{testId:long}/courses/{courseId:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **async** Task**<**IActionResult**>** AddCourset**(**long testId**,** long courseId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**testId**))** **return** BadRequest**(**"Such test does not exist"**);**

**return** Ok**(await** **(**\_repository **as** ITestRepository**).**AddCourse**(**testId**,** courseId**));**

**}**

**[**HttpDelete**(**"{testId:long}/courses/{subjectId:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **async** Task**<**IActionResult**>** RemoveCourse**(**long testId**,** long courseId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**testId**))** **return** BadRequest**(**"Such test does not exist"**);**

var deleted **=** **await** **(**\_repository **as** ITestRepository**).**RemoveCourse**(**testId**,** courseId**);**

**if** **(!**deleted**.**Deleted**)** **return** BadRequest**(**"The test does not belong to such course"**);**

**return** Ok**(**deleted**);**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** class TutorsController **:** ControllerBase

**{**

**private** IRepository**<**Tutor**>** \_repository**;**

**private** IPersonRepository \_people**;**

**public** TutorsController**(**IRepository**<**Tutor**>** repository**,** IPersonRepository people**)** **=>** **(**\_repository**,** \_people**)** **=** **(**repository**,** people**);**

**[**HttpGet**]**

**[**Authorize**(**Roles**=**"Administrator"**)]**

**public** **async** Task**<**IEnumerable**<**Tutor**>>** Get**()** **=>** **await** \_repository**.**Get**();**

**[**HttpGet**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**Tutor**>** Get**(**long id**)** **=>** **await** \_repository**.**Get**(**id**);**

**[**HttpPost**]**

**[**Authorize**(**Roles**=**"Tutor, Administrator"**)]**

**public** **async** Task**<**IActionResult**>** Post**(**Tutor tutor**)**

**{**

**if(!**ModelState**.**IsValid**)**

**{**

**return(**BadRequest**(**ModelState**));**

**}**

**return** Ok**(await** \_repository**.**Post**(**tutor**));**

**}**

**[**HttpPut**(**"{id:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Tutor"**)]**

**public** **async** Task**<**IActionResult**>** Put**(**long id**,** **[**FromBody**]** Tutor tutor**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!**ModelState**.**IsValid**)**

**{**

**return** BadRequest**(**ModelState**);**

**}**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"Tutor does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Put**(**tutor**));**

**}**

**[**HttpDelete**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**IActionResult**>** Delete**(**long id**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"User does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Delete**(**id**));**

**}**

**private** **async** Task**<**bool**>** IsOwner**(**long id**)**

**{**

var userEmail **=** **(**HttpContext**.**User**.**Identity **as** ClaimsIdentity**).**FindFirst**(**ClaimTypes**.**Email**).**Value**;**

var personEmail **=** **(await** \_people**.**Get**(**id**)).**Email**;**

**return** userEmail **==** personEmail**;**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** class StudentsController **:** ControllerBase

**{**

**private** IRepository**<**Student**>** \_repository**;**

**private** IPersonRepository \_people**;**

**public** StudentsController**(**IRepository**<**Student**>** repository**,** IPersonRepository people**)** **=>** **(**\_repository**,** \_people**)** **=** **(**repository**,** people**);**

**[**HttpGet**]**

**[**Authorize**(**Roles**=**"Administrator, Tutor"**)]**

**public** **async** Task**<**IEnumerable**<**Student**>>** Get**()** **=>** **await** \_repository**.**Get**();**

**[**HttpGet**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**Student**>** Get**(**long id**)** **=>** **await** \_repository**.**Get**(**id**);**

**[**HttpPost**]**

**[**Authorize**]**

**public** **async** Task**<**IActionResult**>** Post**(**Student Student**)**

**{**

**if(!**ModelState**.**IsValid**)**

**{**

**return(**BadRequest**(**ModelState**));**

**}**

**return** Ok**(await** \_repository**.**Post**(**Student**));**

**}**

**[**HttpPut**(**"{id:long}"**)]**

**[**Authorize**(**Roles **=** "Administrator, Student"**)]**

**public** **async** Task**<**IActionResult**>** Put**(**long id**,** **[**FromBody**]** Student Student**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!**ModelState**.**IsValid**)**

**{**

**return** BadRequest**(**ModelState**);**

**}**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"Student does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Put**(**Student**));**

**}**

**[**HttpDelete**(**"{id:long}"**)]**

**[**Authorize**]**

**public** **async** Task**<**IActionResult**>** Delete**(**long id**)**

**{**

**if** **(!await** IsOwner**(**id**)** **&&** **!**HttpContext**.**User**.**IsInRole**(**"Administrator"**))** **return** Unauthorized**();**

**if(!await** \_repository**.**Exists**(**id**))**

**{**

**return** BadRequest**(**"User does not exist"**);**

**}**

**return** Ok**(await** \_repository**.**Delete**(**id**));**

**}**

**[**HttpGet**(**"page/{pageNumber:int}/{pageSize:int}"**)]**

**public** **async** Task**<**IEnumerable**<**Student**>>** GetStudentsPage**(**int pageNumber**,** int pageSize**)** **=>** **await** **(**\_repository **as** IStudentsRepository**).**GetStudentPage**(**pageNumber**,** pageSize**);**

**[**HttpPost**(**"{id:long}/courses/{courseId:long}"**)]**

**public** **async** Task**<**IActionResult**>** AddStudentToCourse**(**long id**,** long courseId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**id**))** **return** BadRequest**();**

**return** Ok**(await** **(**\_repository **as** IStudentsRepository**).**AddToCourse**(**id**,** courseId**));**

**}**

**[**HttpDelete**(**"{id:long}/courses/{courseId:long}"**)]**

**public** **async** Task**<**IActionResult**>** RemoveStudentFromCourse**(**long id**,** long courseId**)**

**{**

**if** **(!await** \_repository**.**Exists**(**id**))** **return** BadRequest**();**

**return** **(**Ok**(await** **(**\_repository **as** IStudentsRepository**).**RemoveFromCourse**(**id**,** courseId**)));**

**}**

**private** **async** Task**<**bool**>** IsOwner**(**long id**)**

**{**

var userEmail **=** **(**HttpContext**.**User**.**Identity **as** ClaimsIdentity**).**FindFirst**(**ClaimTypes**.**Email**).**Value**;**

var personEmail **=** **(await** \_people**.**Get**(**id**)).**Email**;**

**return** userEmail **==** personEmail**;**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** class LoginController **:** ControllerBase

**{**

**private** **readonly** IConfiguration \_configuration**;**

**private** **readonly** SignInManager**<**IdentityUser**>** \_signInManager**;**

**private** **readonly** UserManager**<**IdentityUser**>** \_userManager**;**

**public** LoginController**(**IConfiguration configuration**,** SignInManager**<**IdentityUser**>** signInManager**,** UserManager**<**IdentityUser**>** userManager**)**

**{**

\_configuration **=** configuration**;**

\_signInManager **=** signInManager**;**

\_userManager **=** userManager**;**

**}**

**[**HttpPost**]**

**public** **async** Task**<**IActionResult**>** Login**([**FromBody**]** LoginModel login**)**

**{**

var result **=** **await** \_signInManager**.**PasswordSignInAsync**(**login**.**UserName**,** login**.**Password**,** **false,** **false);**

**if(!**result**.**Succeeded**)**

**{**

**return** BadRequest**(new** LoginResult **{** Sussessfull **=** **false,** Error **=** Resources**.**UsernameOrPasswordInvalid **});**

**}**

var user **=** **await** \_userManager**.**FindByNameAsync**(**login**.**UserName**);**

var roles **=** **await** \_userManager**.**GetRolesAsync**(**user**);**

var claims **=** **new[]**

**{**

**new** Claim**(**ClaimTypes**.**Name**,** login**.**UserName**),**

**new** Claim**(**ClaimTypes**.**Email**,** user**.**Email**)**

**};**

var identity **=** **new** ClaimsIdentity**(**claims**,** "Token"**);**

identity**.**AddClaims**(**roles**.**Select**(**r **=>** **new** Claim**(**ClaimTypes**.**Role**,** r**)));**

var key **=** **new** SymmetricSecurityKey**(**Encoding**.**UTF8**.**GetBytes**(**\_configuration**[**"JwtSettings:JwtSecurityKey"**]));**

var creds **=** **new** SigningCredentials**(**key**,** SecurityAlgorithms**.**HmacSha256**);**

var expiry **=** DateTime**.**Now**.**AddDays**(**Convert**.**ToInt32**(**\_configuration**[**"JwtSettings:JwtExpiryDays"**]));**

var token **=** **new** JwtSecurityToken **(null,** **null,** identity**.**Claims**,** expires**:**expiry**,** signingCredentials**:** creds**);**

**return** Ok**(new** LoginResult **{** Sussessfull **=** **true,** Token **=** **new** JwtSecurityTokenHandler**().**WriteToken**(**token**)** **});**

**}**

**}**

**[**Route**(**"api/[controller]"**)]**

**[**ApiController**]**

**public** class AccountsController **:** ControllerBase

**{**

**private** **readonly** UserManager**<**IdentityUser**>** \_userManager**;**

**private** **readonly** AnyTestIdentityDbContext \_idb**;**

**private** **readonly** IPersonRepository \_people**;**

**public** AccountsController**(**UserManager**<**IdentityUser**>** userManager**,** IPersonRepository people**,** AnyTestIdentityDbContext idb**)**

**{**

\_userManager **=** userManager**;**

\_people **=** people**;**

\_idb **=** idb**;**

**}**

**[**HttpPost**]**

**public** **async** Task**<**IActionResult**>** Post**([**FromBody**]** RegisterModel model**)**

**{**

var newUser **=** **new** IdentityUser **{** UserName **=** model**.**UserName**,** Email **=** model**.**Email **};**

var result **=** **await** \_userManager**.**CreateAsync**(**newUser**,** model**.**Password**);**

**if(!**result**.**Succeeded**)**

**{**

var errors **=** result**.**Errors**.**Select**(**x **=>** x**.**Description**);**

**return** Ok**(new** RegisterResult **{** Successful **=** **false,** Errors **=** errors **});**

**}**

**foreach(**var role **in** model**.**Roles**)**

**{**

**if(**AuthService**.**Roles**.**Contains**(**role**))** **await** \_userManager**.**AddToRoleAsync**(**newUser**,** role**);**

**}**

**return** Ok**(new** RegisterResult **{** Successful **=** **true** **});**

**}**

**[**HttpGet**(**"person"**)]**

**[**Authorize**]**

**public** **async** Task**<**Person**>** Person **()**

**{**

Person person **=** **new** Person**();**

**if** **(**HttpContext**.**User**.**Identity **is** ClaimsIdentity identity**)**

**{**

var email **=** identity**.**FindFirst**(**ClaimTypes**.**Email**).**Value**;**

person **=** **await** \_people**.**FindByEmail**(**email**)** **??** **new** Person **{** Email **=** email**};**

**}**

**return** person**;**

**}**

**[**HttpGet**(**"users"**)]**

**[**Authorize**(**Roles **=** "Administrator"**)]**

**public** **async** Task**<**IEnumerable**<**UserInfo**>>** Users**()**

**{**

var identityUserInfo **=** **await** \_idb**.**UserInfos**();**

var personalInfo **=** **await** \_people**.**Get**();**

var result **=** **from** user **in** identityUserInfo

**join** person **in** personalInfo on user**.**Email equals person**.**Email **into** grp

**from** pordef **in** grp**.**DefaultIfEmpty**()**

**select** **new** UserInfo

**{**

User **=** user**.**User**,**

Email **=** user**.**Email**,**

Name **=** $"{pordef?.FamilyName ?? ""} {pordef?.FirstName ?? ""} {pordef?.Patronimic ?? ""}"**.**Trim**(),**

UserPersonId **=** pordef**?.**Id**,**

Roles **=** user**.**Roles

**};**

**return** result**;**

**}**

**}**

Веб-служба визначає для клієнтський додатків наступний Api:

**Accounts**

**POST​/api​/Accounts**

**GET​/api​/Accounts​/person**

**GET​/api​/Accounts​/users**

**Courses**

**POST​/api​/Courses**

**GET​/api​/Courses**

**PUT​/api​/Courses​/{id}**

**DELETE​/api​/Courses​/{id}**

**GET​/api​/Courses​/{id}**

**Login**

**POST​/api​/Login**

**People**

**GET​/api​/People**

**POST​/api​/People**

**GET​/api​/People​/{id}**

**PUT​/api​/People​/{id}**

**DELETE​/api​/People​/{id}**

**Students**

**GET​/api​/Students**

**POST​/api​/Students**

**GET​/api​/Students​/{id}**

**PUT​/api​/Students​/{id}**

**DELETE​/api​/Students​/{id}**

**GET​/api​/Students​/page​/{pageNumber}​/{pageSize}**

**POST​/api​/Students​/{id}​/courses​/{courseId}**

**DELETE​/api​/Students​/{id}​/courses​/{courseId}**

**Subjects**

**POST​/api​/Subjects**

**GET​/api​/Subjects**

**PUT​/api​/Subjects​/{id}**

**DELETE​/api​/Subjects​/{id}**

**GET​/api​/Subjects​/{id}**

**Tests**

**POST​/api​/Tests**

**GET​/api​/Tests**

**PUT​/api​/Tests​/{id}**

**DELETE​/api​/Tests​/{id}**

**GET​/api​/Tests​/{id}**

**GET​/api​/Tests​/list**

**POST​/api​/Tests​/{testId}​/subjects​/{subjectId}**

**DELETE​/api​/Tests​/{testId}​/subjects​/{subjectId}**

**POST​/api​/Tests​/{testId}​/courses​/{courseId}**

**DELETE​/api​/Tests​/{testId}​/courses​/{subjectId}**

**Tutors**

**GET​/api​/Tutors**

**POST​/api​/Tutors**

**GET​/api​/Tutors​/{id}**

**PUT​/api​/Tutors​/{id}**

**DELETE​/api​/Tutors​/{id}**

**Аутентифікація та авторизація користувачів**

Для аутентифікації та авторизації користувачів використана інтегрована з ASP MVC Core система ASP.NET Identity.

Система за замовчуванням зберігає свої дані в базі даних MS SQL та взаємодіє з нею за допомогою Entity Framework. В проекті був використаний клас контексту, який успадковує від класу IdentityDbContext. Зміни до бази даних користувачів протягом розробки проекту не передбачувалися, тому для створення її не використовується технологія міграцій.

Лістінг класу AnyTestIdentityDbContext

**public** class AnyTestIdentityDbContext **:** IdentityDbContext**<**IdentityUser**>**

**{**

**public** AnyTestIdentityDbContext**(**DbContextOptions**<**AnyTestIdentityDbContext**>** options**):** **base(**options**)** **{** **}**

**public** static **async** void SeedData**(**IApplicationBuilder app**)**

**{**

**using** var scope **=** app**.**ApplicationServices**.**CreateScope**();**

var roleManager **=** scope**.**ServiceProvider**.**GetRequiredService**<**RoleManager**<**IdentityRole**>>();**

var userManager **=** scope**.**ServiceProvider**.**GetRequiredService**<**UserManager**<**IdentityUser**>>();**

**if(!await** roleManager**.**RoleExistsAsync**(**"Administrator"**))**

**{**

**foreach(**var role **in** AuthService**.**Roles**)**

**{**

**await** roleManager**.**CreateAsync**(new** IdentityRole**(**role**));**

**}**

var defaultAdmin **=** **new** IdentityUser**{**UserName **=** "admin"**,** Email **=** "admin@admin.com" **};**

var createDefaultAdmin **=** **await** userManager**.**CreateAsync**(**defaultAdmin**,** "Admin\_1"**);**

**if** **(**createDefaultAdmin**.**Succeeded**)**

**{**

**await** userManager**.**AddToRoleAsync**(**defaultAdmin**,** "Administrator"**);**

**}**

**}**

**}**

**public** **async** Task**<**IEnumerable**<**UserInfo**>>** UserInfos**()**

**{**

string query **=**

@"SELECT U.UserName, U.Email, R.Name AS Role

FROM AspNetUsers AS U

INNER JOIN AspNetUserRoles AS UR ON U.Id = UR.UserId

INNER JOIN AspNetRoles AS R ON UR.RoleId = R.Id"**;**

**using** var connection **=** Database**.**GetDbConnection**();**

**using** var command **=** connection**.**CreateCommand**();**

command**.**CommandType **=** System**.**Data**.**CommandType**.**Text**;**

command**.**CommandText **=** query**;**

var identityQueryResult **=** **new** List**<(**string User**,** string Email**,** string Role**)>();**

**await** connection**.**OpenAsync**();**

**using** var reader **=** **await** command**.**ExecuteReaderAsync**();**

**while** **(await** reader**.**ReadAsync**())**

**{**

identityQueryResult**.**Add

**((**

User**:** reader**[**"UserName"**].**ToString**(),**

Email**:** reader**[**"Email"**].**ToString**(),**

Role**:** reader**[**"Role"**].**ToString**()**

**));**

**}**

var result **=** identityQueryResult

**.**GroupBy**(**iqr **=>** **new** **{** iqr**.**User**,** iqr**.**Email **})**

**.**Select**(**gr **=>** **new** UserInfo

**{**

User **=** gr**.**Key**.**User**,**

Email **=** gr**.**Key**.**Email**,**

Roles **=** gr**.**Select**(**r **=>** r**.**Role**)**

**});**

**return** result**;**

**}**

**}**



Діаграма бази даних Identity

База даних Identity

|  |
| --- |
| [dbo].[Asp­Net­Users] |

Columns

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Name | Data Type | Max Length (Bytes) | Nullability |
|  | Id | nvarchar(450) | 900 | NOT NULL |
|  | User­Name | nvarchar(256) | 512 | NULL allowed |
|  | Normalized­User­Name | nvarchar(256) | 512 | NULL allowed |
|  | Email | nvarchar(256) | 512 | NULL allowed |
|  | Normalized­Email | nvarchar(256) | 512 | NULL allowed |
|  | Email­Confirmed | bit | 1 | NOT NULL |
|  | Password­Hash | nvarchar(max) | max | NULL allowed |
|  | Security­Stamp | nvarchar(max) | max | NULL allowed |
|  | Concurrency­Stamp | nvarchar(max) | max | NULL allowed |
|  | Phone­Number | nvarchar(max) | max | NULL allowed |
|  | Phone­Number­Confirmed | bit | 1 | NOT NULL |
|  | Two­Factor­Enabled | bit | 1 | NOT NULL |
|  | Lockout­End | datetimeoffset | 10 | NULL allowed |
|  | Lockout­Enabled | bit | 1 | NOT NULL |
|  | Access­Failed­Count | int | 4 | NOT NULL |

Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Key | Name | Key Columns | Unique | Filtered |
|  | PK\_­Asp­Net­Users | Id | True |  |
|  | User­Name­Index | Normalized­User­Name | True | True |
|  | Email­Index | Normalized­Email |  |  |

|  |
| --- |
| **[dbo].[Asp­Net­Roles]** |

**Columns**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** |
|  | Id | nvarchar(450) | 900 | NOT NULL |
|  | Name | nvarchar(256) | 512 | NULL allowed |
|  | Normalized­Name | nvarchar(256) | 512 | NULL allowed |
|  | Concurrency­Stamp | nvarchar(max) | max | NULL allowed |

**Indexes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** | **Filtered** |
|  | PK\_­Asp­Net­Roles | Id | True |  |
|  | Role­Name­Index | Normalized­Name | True | True |

|  |
| --- |
| **[dbo].[Asp­Net­User­Roles]** |

**Columns**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** |
|  | User­Id | nvarchar(450) | 900 | NOT NULL |
|  | Role­Id | nvarchar(450) | 900 | NOT NULL |

**Indexes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** |
|  | PK\_­Asp­Net­User­Roles | User­Id, Role­Id | True |
|  | IX\_­Asp­Net­User­Roles\_­Role­Id | Role­Id |  |

**Foreign Keys**

|  |  |  |
| --- | --- | --- |
| **Name** | **Delete** | **Columns** |
| FK\_­Asp­Net­User­Roles\_­Asp­Net­Roles\_­Role­Id | Cascade | Role­Id->[[dbo].[Asp­Net­Roles].[Id]](#AGV9rr9uYQr2m2G5vvJJ6d1Q0qo=) |
| FK\_­Asp­Net­User­Roles\_­Asp­Net­Users\_­User­Id | Cascade | User­Id->[[dbo].[Asp­Net­Users].[Id]](#gcvxlQQF5rMm3Wm5XuoFLYhZ6Ho=) |

|  |
| --- |
| **[dbo].[Asp­Net­User­Logins]** |

**Columns**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** |
|  | Login­Provider | nvarchar(450) | 900 | NOT NULL |
|  | Provider­Key | nvarchar(450) | 900 | NOT NULL |
|  | Provider­Display­Name | nvarchar(max) | max | NULL allowed |
|  | User­Id | nvarchar(450) | 900 | NOT NULL |

**Indexes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** |
|  | PK\_­Asp­Net­User­Logins | Login­Provider, Provider­Key | True |
|  | IX\_­Asp­Net­User­Logins\_­User­Id | User­Id |  |

**Foreign Keys**

|  |  |  |
| --- | --- | --- |
| **Name** | **Delete** | **Columns** |
| FK\_­Asp­Net­User­Logins\_­Asp­Net­Users\_­User­Id | Cascade | User­Id->[[dbo].[Asp­Net­Users].[Id]](#gcvxlQQF5rMm3Wm5XuoFLYhZ6Ho=) |

|  |
| --- |
| **[dbo].[Asp­Net­User­Tokens]** |

**Columns**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** |
|  | User­Id | nvarchar(450) | 900 | NOT NULL |
|  | Login­Provider | nvarchar(450) | 900 | NOT NULL |
|  | Name | nvarchar(450) | 900 | NOT NULL |
|  | Value | nvarchar(max) | max | NULL allowed |

**Indexes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** |
|  | PK\_­Asp­Net­User­Tokens | User­Id, Login­Provider, Name | True |

**Foreign Keys**

|  |  |  |
| --- | --- | --- |
| **Name** | **Delete** | **Columns** |
| FK\_­Asp­Net­User­Tokens\_­Asp­Net­Users\_­User­Id | Cascade | User­Id->[[dbo].[Asp­Net­Users].[Id]](#gcvxlQQF5rMm3Wm5XuoFLYhZ6Ho=) |

|  |
| --- |
| **[dbo].[Asp­Net­User­Claims]** |

**Columns**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** | **Identity** |
|  | Id | int | 4 | NOT NULL | 1 - 1 |
|  | User­Id | nvarchar(450) | 900 | NOT NULL |  |
|  | Claim­Type | nvarchar(max) | max | NULL allowed |  |
|  | Claim­Value | nvarchar(max) | max | NULL allowed |  |

**Indexes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** |
|  | PK\_­Asp­Net­User­Claims | Id | True |
|  | IX\_­Asp­Net­User­Claims\_­User­Id | User­Id |  |

**Foreign Keys**

|  |  |  |
| --- | --- | --- |
| **Name** | **Delete** | **Columns** |
| FK\_­Asp­Net­User­Claims\_­Asp­Net­Users\_­User­Id | Cascade | User­Id->[[dbo].[Asp­Net­Users].[Id]](#gcvxlQQF5rMm3Wm5XuoFLYhZ6Ho=) |

|  |
| --- |
| **[dbo].[Asp­Net­Role­Claims]** |

**Columns**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Name** | **Data Type** | **Max Length (Bytes)** | **Nullability** | **Identity** |
|  | Id | int | 4 | NOT NULL | 1 - 1 |
|  | Role­Id | nvarchar(450) | 900 | NOT NULL |  |
|  | Claim­Type | nvarchar(max) | max | NULL allowed |  |
|  | Claim­Value | nvarchar(max) | max | NULL allowed |  |

**Indexes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Name** | **Key Columns** | **Unique** |
|  | PK\_­Asp­Net­Role­Claims | Id | True |
|  | IX\_­Asp­Net­Role­Claims\_­Role­Id | Role­Id |  |

**Foreign Keys**

|  |  |  |
| --- | --- | --- |
| **Name** | **Delete** | **Columns** |
| FK\_­Asp­Net­Role­Claims\_­Asp­Net­Roles\_­Role­Id | Cascade | Role­Id->[[dbo].[Asp­Net­Roles].[Id]](#AGV9rr9uYQr2m2G5vvJJ6d1Q0qo=) |

**Розробка та реалізація клієнтських компонентів системи**

Система містить два клієнтський копоненти: браузерний клієнт та андроід-клієнт. Перед ними стоять схожі завдання, тому спочатку ми окреслимо загальні моменти в концепції обох компонентів, а потом вкажемо ті аспекти, які відрізняються та опишемо їх реалізацію.

Мета клієнтських компонентів — надавати користувачам графічний інтерфейс доступу до центрального компонента системи і можливість керування ним в залежності від ролі конкретного користувача. При цьому браузерний клієнт надає функціональність всіх трьох передбачений ролей («Адміністратор», «Викладач», «Студент»), а мобільний клієнт надає функціональність лише для студента. Це обумовлено тим, що функціонал адміністратора та викладача включає в себе введення та редагування великої кількості даних, тому було вирішено, що невеликий екран мобільного пристрою та сенсорний ввід зроблять ергономіку цього функціоналу незадовільною

Основними задачами клієнтський додатків для ролі студента є надання достуту до списку тестів, обирати тест для проходження, проходити тест, зберігати результати тестів в базі даних та переглядати результати пройдених тестів.

Задачами для ролі викладача є створення предметів, курсів та тестів, редагування предметів, курсів та тестів, перегляд результатів студентів.

Задачами для ролі адміністратора є створення, редагування та видалення користувачів системи.

Окремо слід виділити задачу реєстрації студента в системі — її може здійснювати незареєстрований користувач.

Первинною мовою інтерфейсу при розробці обрано українську, однак розробка має вестися так, щоб надати доступну можливість для локалізації інтерфейсу будь-якою обраною іншою мовою. З цією метою для вказання тесту екранних повідомлень використовується техніка використання ресурсів, вбудована в середовище .NET. В окремому проекті Visual Studio, щоб дати можливість для використання в різних частинах продукту, створено файл ресурсів формату .resx, та автоматично створений на оновлюємий IDE Visual Studio служобвий клас, який містить методи-геттери для окремих ресурсів. У моделі та клієнтських додатках замість стрічкових літералів використовуються або властивості службового класу, які повертають ресурс згідно поточній локалізації, або реалізацію інтерфейсу IStringLocalizer, завданням якого є також визначення та повернення відповідного ресурсу. В процесі розробки проекту створюється лише файл ресурсів українською мовою, але при потребі додати нову локалізацію достатньо буде лише додати до рішення відповідний файл ресурсів, ніяких змін до коду вносити буде не треба.

**Задачі клієнтський додатків**

**Задачі ролі студента**

Перегляд списку курсів, вибір курсу, проходження курсу, перегляд результатів

Частота — часто

Час на виконання завдання — короткий

Хто використовує — студент

Особливості користувача:

* Вік молодий
* Освіта: середня, неповна вища
* Базовий комп’ютерний досвід: середній або просунутий користувач
* Досвід предметної області: наявний
* Робоче середовище: в класі або поза класом (браузерний клієнт), поза класом (мобільний клієнт)

Можливі виключні ситуації:

* Відсутність зв’язку з центральним компонентом
* Відсутність користувача в базі/невірні користувацькі дані

**Задачі ролі викладача**

Перегляд списку студентів, перегляд списку тестів, додавання/редагування тестів, перегляд результатів студентів

Частота:

Додавання/редагування тестів — рідко

Перегляд списку студентів, перегляд результатів студентів — часто

Час на виконання завдання — короткий

Хто використовує — викладач

Особливості користувача:

* Вік середній або літній
* Освіта вища
* Базовий комп’ютерний досвід: можливо неосвічений користувач
* Досвід предметної області: просунутий
* Робоче середовище: клас, робоче місце

Можливі виключні ситуації:

* Відсутність зв’язку з центральним компонентом
* Відсутність користувача в базі/невірні користувацькі дані

**Задачі ролі адміністратора**

Перегляд списку користувачів, додавання викладачів та адміністраторів

Частота: рікдко

Час на виконання завдання — короткий

Хто використовує — викладач

Особливості користувача:

* Базовий комп’ютерний досвід: просунутий
* Досвід предметної області: просунутий
* Робоче середовище: клас, робоче місце

Можливі виключні ситуації:

* Відсутність зв’язку з центральним компонентом
* Відсутність користувача в базі/невірні користувацькі дані



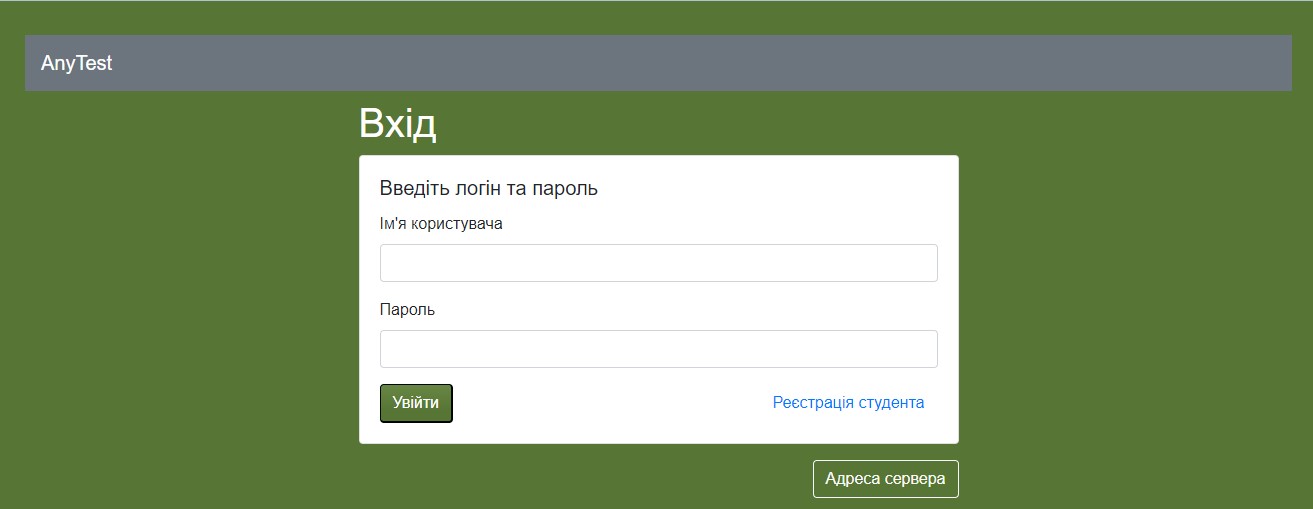
Мапа сайту



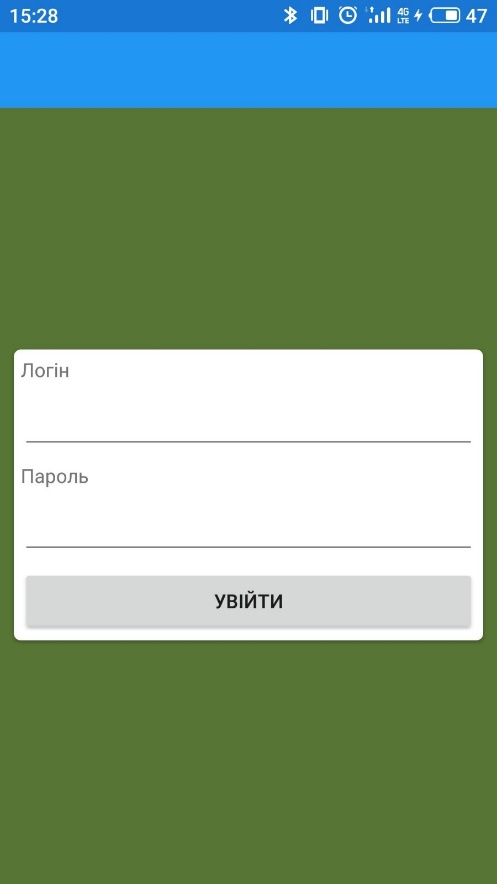
Схема мобільного додатку

Веб-додаток в верхній частині містить навігаційну панель із посиланнями на основні частини додатку. Кількість та вміст посилань залежать від типу користувача. Мобільний додаток студента складається з трьох сторінок-вкладок, на яких розміщені предмети, які вивчає студент, курси, які він проходить та тести, які йому задані.

При вході в додаток неавторизованого користувача відображається сторінка входу



Сторінка авторизації веб-додатку

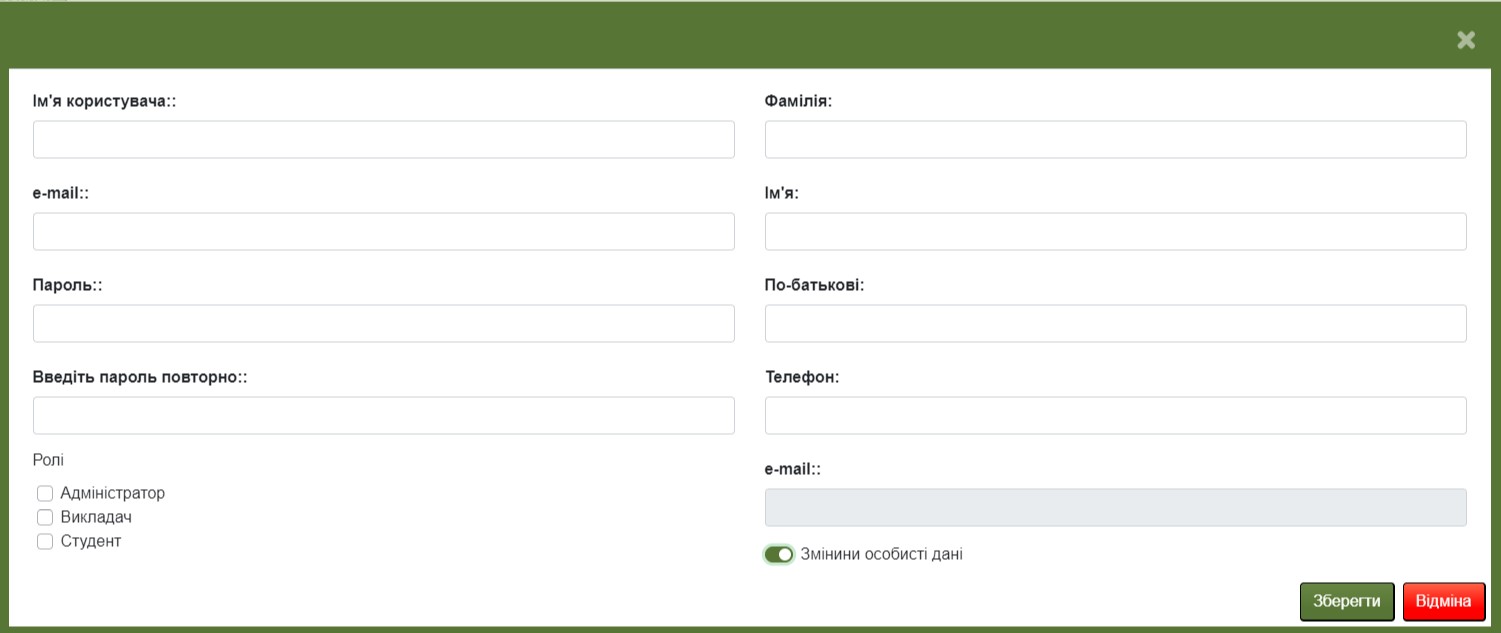


Сторінка авторизації мобільного додатку

Сторінка авторизації містить такі компоненти: поля для вводу логіна та пароля, та кнопку входу. На сторінці авторизації веб-додатку також міститься посилання для переходу на реєстрацію студента в системі.

Конструкція сторінки авторизації

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування елементу | Тип елементу | Дія користувача | Відклик системи |
| Поле вводу логіну та паролю | Текстове поле вводу | Ввід з клавіатури | — |
| Увійти | Кнопка | Одинарний клік лівою кнопкою миші (веб-додаток), одинарне натискання (мобільний додаток) | Перевірка введених логіну та паролю. При вірному вводі вхід в додаток, при невірному відображення повідомлення про помилку |
| Реєстрація студенту | Гіперпосилання | Одинарний клік лівою кнопкою миші | Перехід на сторінку реєстрації |



Форма реєстрації користувача

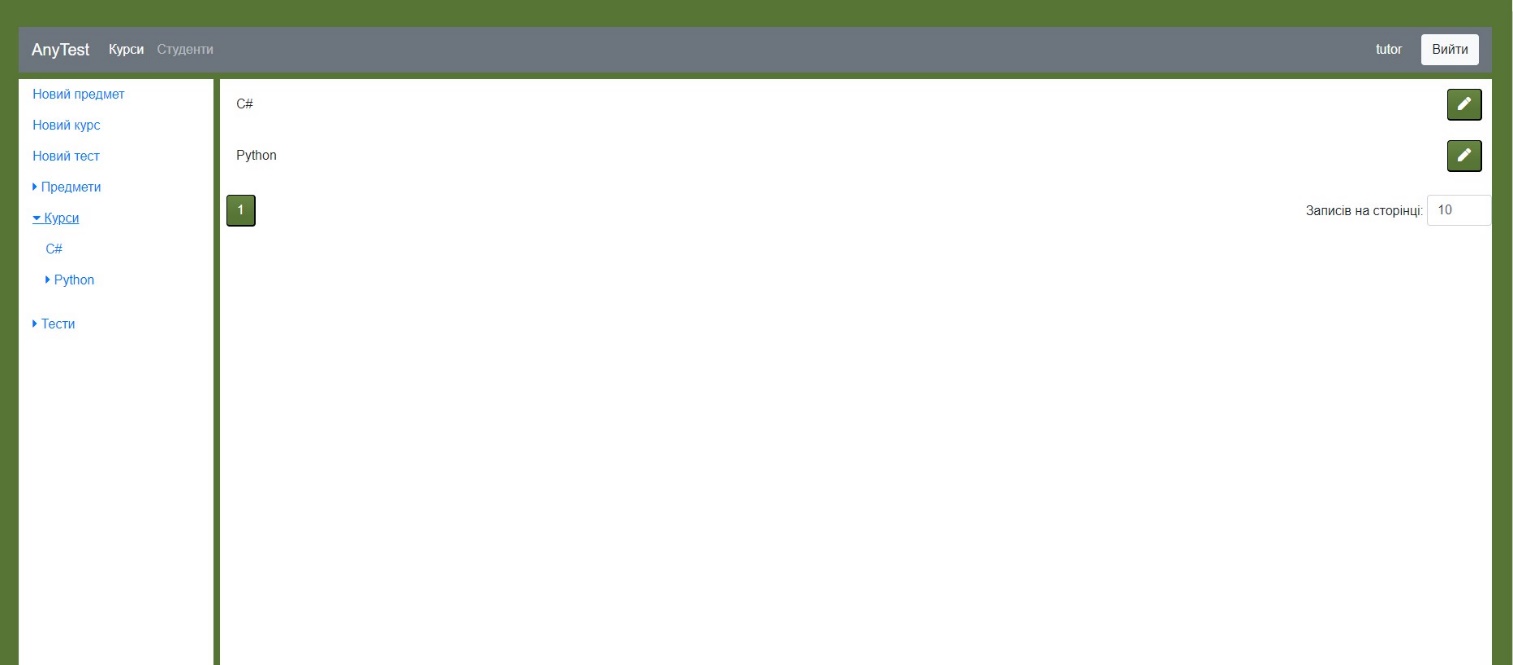
На формі реєстрації користувача зліва знаходяться поля для ввода даних для входа в систему: логіна та паролі, та електронної адреси користувача. Також, якщо реєстрацію користувача здійснює адміністратор, він може обрати ролі, які будуть призначені користувачу в системі авторизації. Неавторизований користувач може зареєструватися в системі лише в якості студента, поля ролей не відібражаються.

Справа на формі знаходяться поля для вводу особистих даних: імені та телефону. Поле електронної пошти відключене, бо електронна пошта прив’язана до авторизаційного аккаунту та є зв’язком між таблицями користувачів в базі авторизації та студентів та викладачів в основній базі даних

Дана форма призначена для реєстрації користувача в системі. Незареєстрований користувач може зареєструватися лише в якості студента. Реєстрація користувача з ролями «адміністратор» та «викладач» можуть здійснювати лише користувачі, які входять до ролі «адміністратор». Обов’язковими при реєстрації є поля логіну, паролю, підтвердження паролю та електронної пошти. Якщо перемикач «Змінити особисті дані» відключено, то запис буде створено лише в базі даних авторизації і користувачу буде запропоновано ввести свої дані самостійно при першому вході в систему. При встановленні перемикача «змінити особисті дані» в положення «включено», при реєстрації також буде створено запис в таблиці «Особи» із особистими даними користувача, а також запис в таблицях «Викладачі» або «Студенти» в залежності від обраних ролей. При вводі особистих даних обов’язковими є поля «Ім’я» та «Фамілія». Електронна пошта вводиться при реєстрації користувача в системі авторизації.

Конструкція форми реєстрації

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування елемента форми | Тим елементу форми | Дія користувача | Відклик системи |
| Поля вводу даних авторизації | Текстові поля вводу | Ввід з клавіатури | — |
| Поля вибору ролей користувача | Чекбокси | Одинарний клік лівою кнопкою миші | — |
| Поля вводу особистих даних | Текстові поля вводу | Ввід з клавіатури | — |
| Перемикач вводу особистих даних | Перемикач | Одинарний клік лівою кнопкою миші | Поля вводу особистих даних переходять із неактивної форми в активну та навпаки |
| Зберегти | Кнопка | Одинарний клік лівою кнопкою миші | Валідація введених даних. При коректності даних — збереження нового користувача |
| Відміна | Кнопка | Одинарний клік лівою кнопкою миші | Відміна реєстрації користувача. Повернення на екран, з якого була викликана форма реєстрації |



Сторінка курсів викладача

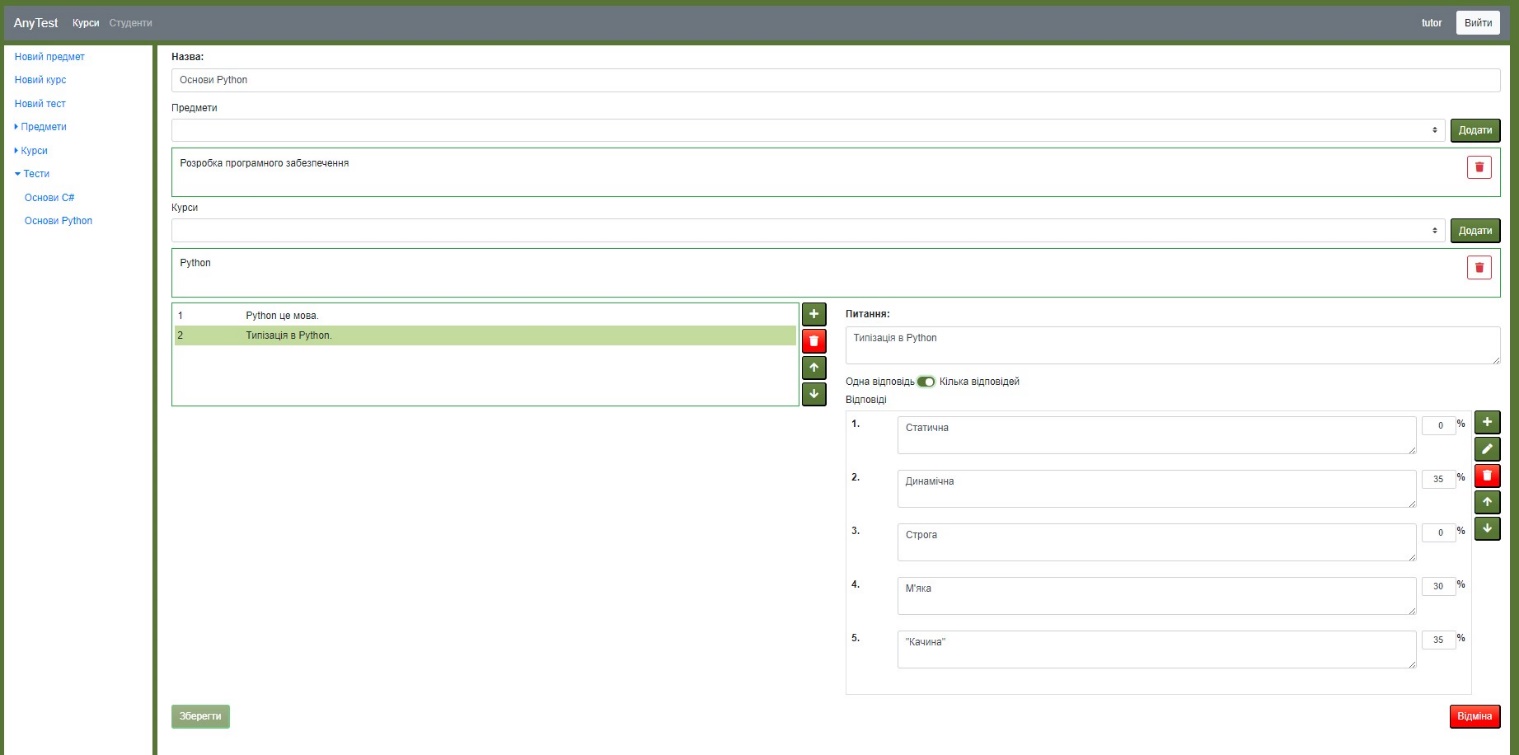
На сторінці курсів викладача викладач може переглядати наявні предмети, курси та тести, створювати нові та редагувати предмети, курси та тести, автором чи власником яких він є.

Зліва на сторінці знаходиться навігаційна панель, яка містить деревовидні списки предметів, курсів та тестів та посилання на сторінки створення нових предметів, тестів та курсів.

Основна панель містить перелік предметів, курсів та тестів або усіх, або таких, які належать до обраної на навігаційному меню категорії (курси з певного предмету, тести, які належать до певного курсу) у вигляді назви та посилання (оформленого як кнопка за допомогою класів бібліотеки bootstrap) на сторінку перегляду або редагування докладних характеристик обраного предмету, тесту чи курсу.

Конструкція сторінки курсів викладача

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування елемента форми | Тим елементу форми | Дія користувача | Відклик системи |
| Бічна панель навігації | Гіперпосилання | Одинарний клік лівою кнопкою миші | Перехід на сторінку відповідного елементу.  При наявності підкатегорій — відкриття дерева підкатегорій |
| Список курсів, розбитий на сторінки | Таблиця | Одинарний клік лівою кнопкою миші | Перехід на сторінку відповідного курсу |
| Список сторінок | Посилання | Перехід на відповідну сторінку списку курсів | Відображення відповідної сторінки списку курсів |
| Поле розміру сторінки | Числове поле вводу | Вводу за допомогою клавіатури або одинарного кліку лівою кнопкою миші | Зміна кількості курсів, які відображаються на сторінці |

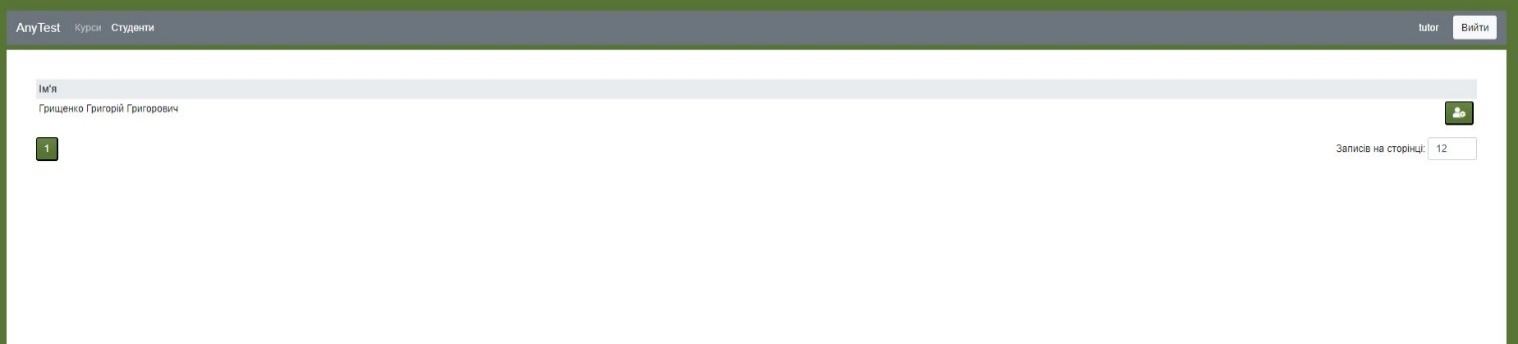


Екран редагування тесту

Екран редагування тесту містить поле для вводу назви тесту, випадаючі списки для обрання предметів та курсів, до яких має відноситися тест, перелік питань тесту із можливістю додавати, видаляти питання та переміщати їх вгору та вниз у порядку питань на форму редагування питання, яка містить поле для питання, перемикач режимів «одна відповідь/багато відповідей» та список відповідей із можливістью додавати, видаляти відповіді, відмічати відсоток вірності відповідей та змінювати їх порядок

Конструкція сторінки редагування тесту.

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування елемента форми | Тип елементу форми | Дія користувача | Відклик системи |
| Поле назви тесту | Поле вводу | Ввід з клавіатури | — |
| Список предметів | Випадаючий список | Одинарний клік лівою кнопкою миші | — |
| Кнопка додати предмет | Кнопка | Одинарний клік лівою кнопкою миші | Предмет додається до списку предметів |
| Кнопка «видалити предмет» | Кнопка | Одинарний клік лівою кнопкою миші | Предмет видаляється зі списку предметів |
| Список курсів | Випадаючий список | Одинарний клік лівою кнопкою миші | — |
| Кнопка додати курс | Кнопка | Одинарний клік лівою кнопкою миші | Курс додається до списку курсів |
| Кнопка «видалити курс» | Кнопка | Одинарний клік лівою кнопкою миші | Курс видаляється зі списку курсів |
| Список курсів | Таблиця | — | — |
| Кнопки керування питаннями | Кнопки | Одинарний клік лівою кнопкою миші | Зміна списку курсів |
| Назва питання | Тестове поле вводу | Ввід с клавіатури | — |
| Перемикач «одна/кілька відповідей» | Перемикач | Одинарний клік лівою кнопкою миші | Змінюється тип вказання відповідей |
| Таблиця відповідей | Таблиця, текстове поле вводу, чекбокс, числове поле вводу | Одинарний клік лівою кнопкою миші, ввід з клавіатури | — |
| Кнопки керування відповідями | Кнопки | Одинарний клік лівою кнопкою миші | Зміна списку відповідей |
| «Зберегти» | Кнопка | Одинарний клік лівою кнопкою миші | Збереження тесту до бази даних |
| «Відміна» | Кнопка | Одинарний клік лівою кнопкою миші | Відміна створення/редагування тесту |



Сторінка студентів викладача

Сторінка містить розбитий на сторінки список студентів із можливістю перейти до перегляду та редагування студенту

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування елемента форми | Тип елементу форми | Дія користувача | Відклик системи |
| Список студентів | Список | Одинарний клік лівою кнопкою миші | Перехід на сторінку даних студента |

Як браузерний так і клієнтський додатки побудовані за концепцією MVVM (Model-View-ViewModel). Концепція MVVM передбачає розділення додатку на три рівні: рівень моделі, рівень представлення і рівень моделі представлення, яка здійснює зв’язок між моделлю та представленням за допомогою механізму прив’язки елементів моделі до елементів представлення.

Обидві обрані технології (Blazor для браузерного додатку та Xamarin.Forms для мобільного додатку) підтримують концепцію MVVM та механізми прив’язки.

Для обох додатків створюється основна модель стану додатку, яка зберігає глобальний стан додатку. В браузерному додатку це клас StateContainerViewModel:

**{**

**private** HttpClient \_httpClient**;**

**public** StateContainerViewModel**(**HttpClient httpClient**)** **=>** \_httpClient **=** httpClient**;**

**public** Person Person **{** **get;** **private** **set;** **}** **=** **new** Person**();**

**public** List**<**UserInfo**>** Users **{** **get;** **private** **set;** **}** **=** **new** List**<**UserInfo**>();**

**public** List**<**Student**>** Students **{** **get;** **private** **set;** **}** **=** **new** List**<**Student**>();**

**public** Dictionary**<**string**,** List**<**TestsTreeModel**>>** TestsTreeList **{** **get;** **private** **set;** **}** **=** **new** Dictionary**<**string**,** List**<**TestsTreeModel**>>();**

**public** **event** EventHandler TestsUpdated**;**

**public** IEnumerable**<**Subject**>** Subjects **=>**

TestsTreeList**.**ContainsKey**(**"subjects"**)** **?** TestsTreeList**[**"subjects"**].**Select**(**tm **=>** **new** Subject **{** Id **=** tm**.**Id**,** Name **=** tm**.**Name **})** **:** **new** List**<**Subject**>();**

**public** IEnumerable**<**Course**>** Courses **=>**

TestsTreeList**.**ContainsKey**(**"courses"**)** **?** TestsTreeList**[**"courses"**].**Select**(**tm **=>** **new** Course **{** Id **=** tm**.**Id**,** Name **=** tm**.**Name **})** **:** **new** List**<**Course**>();**

**public** IEnumerable**<**Test**>** Tests **=>**

TestsTreeList**.**ContainsKey**(**"tests"**)** **?** TestsTreeList**[**"tests"**].**Select**(**tm **=>** **new** Test **{** Id **=** tm**.**Id**,** Name **=** tm**.**Name **})** **:** **new** List**<**Test**>();**

**public** Person LoadingStub **=** **new** Person

**{**

FirstName **=** "Loading..."**,**

FamilyName **=** "Loading..."**,**

Patronimic **=** "Loading..."**,**

Phone **=** "Loading..."**,**

Email **=** "Loading..."

**};**

**public** void ResetPerson**()** **=>** Person **=** **new** Person**();**

**public** **async** Task GetPersonByAuthorizedUser**()**

**{**

Person **=** LoadingStub**;**

Person person **=** **new** Person**();**

**try**

**{**

person **=** **await** \_httpClient**.**GetJsonAsync**<**Person**>(**"Accounts/Person"**);**

**}**

**finally**

**{**

Person **=** person**;**

**}**

**}**

**public** Person GetCopyOfPerson**()**

**{**

**return** **new** Person

**{**

Id **=** Person**.**Id**,**

FirstName **=** Person**.**FirstName**,**

FamilyName **=** Person**.**FamilyName**,**

Patronimic **=** Person**.**Patronimic**,**

Phone **=** Person**.**Phone**,**

Email **=** Person**.**Email

**};**

**}**

**public** **async** Task SavePerson**(**Person person**,** IEnumerable**<**string**>** roles **=** **null)**

**{**

**if(**person**.**Id **==** 0**)**

**{**

Person **=** **await** \_httpClient**.**PostJsonAsync**<**Person**>(**"people"**,** person**);**

**if(**roles**?.**Contains**(**"Tutor"**)** **??** **false)** **await** SaveTutor**(new** Tutor**());**

**if(**roles**?.**Contains**(**"Student"**)** **??** **false)** **await** SaveStudent**(new** Student**());**

**}**

**else** Person **=** **await** \_httpClient**.**PutJsonAsync**<**Person**>(**$"people/{person.Id}"**,** person**);**

**}**

**public** **async** Task GetUsers**()**

**{**

Users **=** **await** \_httpClient**.**GetJsonAsync**<**List**<**UserInfo**>>(**"accounts/users"**);**

**}**

**public** **async** Task SaveTutor**(**Tutor tutor**)**

**{**

**if** **(**tutor**.**Id **==** 0**)**

**{**

tutor**.**Id **=** Person**.**Id**;**

Person**.**Tutor **=** **await** \_httpClient**.**PostJsonAsync**<**Tutor**>(**"tutors"**,** tutor**);**

**}**

**else** Person**.**Tutor **=** **await** \_httpClient**.**PutJsonAsync**<**Tutor**>(**"tutors"**,** tutor**);**

**}**

**public** **async** Task SaveStudent**(**Student student**)**

**{**

**if** **(**student**.**Id **==** 0**)**

**{**

student**.**Id **=** Person**.**Id**;**

Person**.**Student **=** **await** \_httpClient**.**PostJsonAsync**<**Student**>(**"students"**,** student**);**

**}**

**else** Person**.**Student **=** **await** \_httpClient**.**PutJsonAsync**<**Student**>(**"students"**,** student**);**

**}**

**public** **async** Task GetTestsList**()**

**{**

TestsTreeList **=** **await** \_httpClient**.**GetJsonAsync**<**Dictionary**<**string**,** List**<**TestsTreeModel**>>>(**"tests/list"**);**

TestsUpdated**?.**Invoke**(this,** EventArgs**.**Empty**);**

**}**

**public** **async** Task SaveSubject**(**Subject subject**)**

**{**

**if(**subject**.**Id **==** 0**)**

**{**

**await** \_httpClient**.**PostJsonAsync**(**"subjects"**,** subject**);**

**}**

**else**

**{**

**await** \_httpClient**.**PutJsonAsync**(**$"subjects/{subject.Id}"**,** subject**);**

**}**

**await** GetTestsList**();**

**}**

**public** **async** Task SaveCourse**(**Course course**)**

**{**

**if(**course**.**Id **==** 0**)**

**{**

**await** \_httpClient**.**PostJsonAsync**(**"courses"**,** course**);**

**}**

**else**

**{**

**await** \_httpClient**.**PutJsonAsync**(**$"courses/{course.Id}"**,** course**);**

**}**

**await** GetTestsList**();**

**}**

**public** **async** Task SaveTest**(**Test test**)**

**{**

**if(**test**.**Id **==** 0**)**

**{**

**if(**Person**.**Id **==** 0**)** **await** GetPersonByAuthorizedUser**();**

test**.**AuthorId **=** Person**.**Id**;**

**await** \_httpClient**.**PostJsonAsync**(**"tests"**,** test**);**

**}**

**else**

**{**

**await** \_httpClient**.**PutJsonAsync**(**$"tests/{test.Id}"**,** test**);**

**}**

**await** GetTestsList**();**

**}**

**public** **async** Task GetStudents**()**

**{**

Students **=** **await** \_httpClient**.**GetJsonAsync**<**List**<**Student**>>(**"students"**);**

**}**

**}**

Екземпляр цього класу вводиться до браузерного додатку за допомогою ін’єкції залежності як сінглєтон.

Через те, що технологія Xamarin.Forms не має нативного контейнету залежностей, в якості контейнера стану в мобільному додатку використовується статичний клас AppState^

**public** static class AppState

**{**

**private** static **readonly** string tokenKey **=** "authToken"**;**

**public** static HttpClient HttpClient **=** **new** HttpClient**(**

**new** HttpClientHandler

**{**

ServerCertificateCustomValidationCallback **=** **(**sender**,** cert**,** chain**,** sslPolicyErrors**)** **=>** **true**

**},** **false**

**)** **{** BaseAddress **=** **new** Uri**(**CrossSettings**.**Current**.**GetValueOrDefault**(**"baseAddress"**,** "https://192.168.0.115:44358/api/"**))** **};**

**public** static ObservableCollection**<**TestsTreeModel**>** Subjects **=** **new** ObservableCollection**<**TestsTreeModel**>();**

**public** static ObservableCollection**<**TestsTreeModel**>** Courses **=** **new** ObservableCollection**<**TestsTreeModel**>();**

**public** static ObservableCollection**<**TestsTreeModel**>** Tests **=** **new** ObservableCollection**<**TestsTreeModel**>();**

**public** static bool IsLoggedIn **=>**

App**.**Current**.**Properties**.**ContainsKey**(**tokenKey**)**

**&&** App**.**Current**.**Properties**[**tokenKey**]** **is** string token

**&&** **!**string**.**IsNullOrWhiteSpace**(**token**)**

**&&** **!**ApiAuthenticaionStateProvider**.**TokenExpired**(**token**);**

**public** static **async** Task**<**LoginResult**>** Login**(**ClientAuthentication**.**LoginModel model**)**

**{**

var loginAsJson **=** JsonSerializer**.**Serialize**(**model**);**

var response **=** **await** HttpClient**.**PostAsync**(**"Login"**,** **new** StringContent**(**loginAsJson**,** Encoding**.**UTF8**,** "application/json"**));**

var loginResult **=** JsonSerializer**.**Deserialize**<**LoginResult**>(await** response**.**Content**.**ReadAsStringAsync**(),** **new** JsonSerializerOptions **{** PropertyNameCaseInsensitive **=** **true});**

**if(**response**.**IsSuccessStatusCode**)**

**{**

App**.**Current**.**Properties**[**tokenKey**]** **=** loginResult**.**Token**;**

HttpClient**.**DefaultRequestHeaders**.**Authorization **=** **new** AuthenticationHeaderValue**(**"bearer"**,** loginResult**.**Token**);**

**}**

**return** loginResult**;**

**}**

**public** static void Logout**()** **=>** App**.**Current**.**Properties**.**Remove**(**tokenKey**);**

**private** static **async** Task GetTestsList**()**

**{**

**try**

**{**

var tests **=** **await** HttpClient**.**GetJsonAsync**<**Dictionary**<**string**,** List**<**TestsTreeModel**>>>(**"tests/list"**);**

**if** **(**tests**.**ContainsKey**(**"subjects"**))** tests**[**"subjects"**].**ForEach**(**s **=>** Subjects**.**Add**(**s **as** TestsTreeModel**));**

**if** **(**tests**.**ContainsKey**(**"courses"**))** tests**[**"courses"**].**ForEach**(**c **=>** Courses**.**Add**(**c **as** TestsTreeModel**));**

**if** **(**tests**.**ContainsKey**(**"tests"**))** tests**[**"tests"**].**ForEach**(**t **=>** Tests**.**Add**(**t **as** TestsTreeModel**));**

**}**

**catch(**Exception ex**)**

**{**

Console**.**WriteLine**(**ex**.**Message**);**

**}**

**}**

**public** static **async** void LoadTestsList**()** **=>** **await** GetTestsList**();**

**}**

Обидва класи містять властивості, в яких зберігаються об’єкти стану додатку, які споживаються незалежно різними компонентами додатку та методи, які використовуються для оновлення цих властивостей.

**Тестування системи**

Для тестування використовується ручне функціональне тестування за допомогою тестових сценаріїв та автоматизоване модульне тестування.

**Функціональне тестування**

План тестування

|  |  |  |  |
| --- | --- | --- | --- |
| № | Варіант використання | Тест | Результати |
| 1 |  | Авторизація адміністратора: ім’я користувача admin, пароль Admin\_1 | Тест виконано |
| 2 | Вхід в програму | Авторизація неіснуючого користувача: ім’я 1, пароль 1 | Тест не виконано |
| 3 | Авторизація студента: ім’я student, пароль Student\_1 | Тест виконано |
| 4 | Пройти тест | Виконати тест: тест Основи C# | Тест виконано |
| 5 | Ознайомитися з результатами | Завершити виконання тесту студентом | Тест виконано |
| 6 | Виконати авторизацію викладача і ознайомитися із загальними результатами тестування знать |  |
| 7 | Додати предмет | Додати предмет. Назва предмету: математика | Тест виконано |
| 8 | Видалити предмет | Видалити предмет: назва предмету математика | Тест виконано |
| 9 | Додати курс | Додати курс: назва тесту Python | Тест виконано |
| 10 | Видалити курс | Видалити курс: назва тесту Python | Тест виконано |
| 11 | Додати тест | Додати тест: назва тесту Класи в C# | Тест виконано |
| 12 | Видалити тест | Видалити тест: назва тесту Класи в C# | Тест виконано |
| 13 | Додати питання | Додати питання до існуючого тесту. Питання: типізація в C#. Відповіді відсутні | Тест не виконано |
| 14 |  | Додати питання до існуючого тесту. Питання: типізація в C#. Відповіді Строга, м’яка, статична, динамічна, «Качина». Правільні відповіді: не вказані | Тест не виконаний |
| 15 |  | Додати питання до існуючого тесту. Питання: типізація в C#. Відповіді Строга, м’яка, статична, динамічна, «Качина». Правільні відповіді: 50%, 0%, 50%, 0%, 0% | Тест виконано |
| 16 | Видалити питання | Видалити існуюче питання | Тест виконано |
| 17 | Реєстрація студента | Реєстрація нового студента: Ім’я: Василенко, Ім’я: Василь, по-батькові: Васильович, логін testStudent, пароль: Student\_2, підтрверждення паролью: Student\_2 | Тест виконано |

На підставі результатів тестування можна зробити висновок, що основні функції продукту, що розробляються виконуються коректно.

**Модульне тестування**

Модульне або unit-тестування дозволяє дослідити працездатність окремих модулів системи. Воно проводиться на всіх етапах розробки, при створенні нових функцій, додаванні та зміні функцій вже існуючої системи.

В якості рішення для модульного тестування був обраний MS Test. Для створення імітаційних об’єктів залежності використаний фремвокр Moq.

Розглянемо модульне тестування на прикладі тестування класу StudentsController. Для його тестування створений наступний тестовий клас:

[TestClass]

public class StudentsControllerTests

{

private Mock<IStudentsRepository> repository = new Mock<IStudentsRepository>();

private Mock<IPersonRepository> people = new Mock<IPersonRepository>();

private StudentsController controller;

public StudentsControllerTests()

{

controller = new StudentsController(repository.Object, people.Object);

}

[TestMethod]

public void GetReturnsListOfStudents()

{

//Arrange

repository.Setup(r => r.Get()).ReturnsAsync(new List<Student>() as IEnumerable<Student>);

//Act

var result = controller.Get().Result;

//Assert

Assert.IsInstanceOfType(result, typeof(IEnumerable<Student>));

}

[TestMethod]

public void GetIdReturnsStudent()

{

//Arrange

repository.Setup(r => r.Get(It.IsAny<object[]>())).ReturnsAsync((object[] id) =>

{

return new Student

{

Id = (long)id[0]

};

});

long studentId = 1;

//Act

var result = controller.Get(studentId).Result;

//Assert

Assert.IsInstanceOfType(result, typeof(Student));

Assert.AreEqual(studentId, result?.Id);

}

[TestMethod]

public void PostReturnsStudent()

{

//Arrange

repository.Setup(r => r.Post(It.IsAny<Student>())).ReturnsAsync((Student s) => { return s; });

//Act

var result = (controller.Post(new Student()).Result as OkObjectResult).Value;

//Assert

Assert.IsInstanceOfType(result, typeof(Student));

}

[TestMethod]

public void PutReturnsCaseHistory()

{

//Arrange

var context = new Mock<HttpContext>();

controller.ControllerContext = new ControllerContext { HttpContext = context.Object };

var identity = new Mock<ClaimsIdentity>();

context.SetupGet(c => c.User.Identity).Returns(identity.Object);

string email = "test@test.com";

identity.Setup(i => i.FindFirst(ClaimTypes.Email)).Returns(new Claim("email", email));

people.Setup(p => p.Get(It.IsAny<long>())).ReturnsAsync(new Person { Email = email });

repository.Setup(r => r.Exists(It.IsAny<object[]>())).ReturnsAsync(true);

repository.Setup(r => r.Put(It.IsAny<Student>(), It.IsAny<object[]>())).ReturnsAsync((Student s, object[] key) => { return s; });

//Act

var result = (controller.Put(1, new Student()).Result as OkObjectResult).Value;

//Assert

Assert.IsInstanceOfType(result, typeof(Student));

}

[TestMethod]

public void DeleteReturnsStudent()

{

//Arrange

var context = new Mock<HttpContext>();

controller.ControllerContext = new ControllerContext { HttpContext = context.Object };

var identity = new Mock<ClaimsIdentity>();

context.SetupGet(c => c.User.Identity).Returns(identity.Object);

string email = "test@test.com";

identity.Setup(i => i.FindFirst(ClaimTypes.Email)).Returns(new Claim("email", email));

people.Setup(p => p.Get(It.IsAny<long>())).ReturnsAsync(new Person { Email = email });

repository.Setup(r => r.Exists(It.IsAny<object[]>())).ReturnsAsync(true);

repository.Setup(r => r.Delete(It.IsAny<object[]>())).ReturnsAsync(new Student());

//Act

var result = (controller.Delete(1).Result as OkObjectResult).Value;

//Assert

Assert.IsInstanceOfType(result, typeof(Student));

}

[TestMethod]

public void GetStudensPageReturnsListOfStudents()

{

//Arrange

repository.Setup(r => r.GetStudentPage(It.IsAny<int>(), It.IsAny<int>())).ReturnsAsync(new List<Student>() as IEnumerable<Student>);

//Act

var result = controller.GetStudentsPage(0, 15).Result;

//Assert

Assert.IsInstanceOfType(result, typeof(IEnumerable<Student>));

}

[TestMethod]

public void AddStudentToCourseReturnsStudenttCourse()

{

//Arrange

repository.Setup(r => r.AddToCourse(It.IsAny<long>(), It.IsAny<long>())).ReturnsAsync(new StudentCourse());

repository.Setup(r => r.Exists(It.IsAny<object[]>())).ReturnsAsync(true);

//Act

var result = (controller.AddStudentToCourse(1, 1).Result as OkObjectResult)?.Value;

//Assert

Assert.IsInstanceOfType(result, typeof(StudentCourse));

}

[TestMethod]

public void RemoveStudentFromCourseReturnsStdentCourse()

{

//Arrange

repository.Setup(r => r.RemoveFromCourse(It.IsAny<long>(), It.IsAny<long>())).ReturnsAsync(new StudentCourse());

repository.Setup(r => r.Exists(It.IsAny<object[]>())).ReturnsAsync(true);

//Act

var result = (controller.RemoveStudentFromCourse(1, 1).Result as OkObjectResult)?.Value;

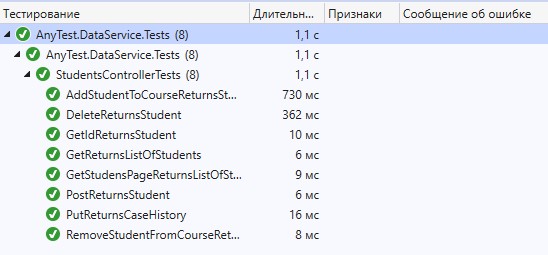
//Assert

Assert.IsInstanceOfType(result, typeof(StudentCourse));

}

}

При модульному тестуванні класу StudentsController отримані наступні результати:



**Діаграма розгортання**

****