**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**Навчально-науковий інститут інноваційних освітніх технологій**

**Кафедра інженерії програмного забезпечення**

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

О.О. Писарчук

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_2019 р.

**ДИПЛОМНИЙ ПРОЕКТ**

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ**

**БАКАЛАВРА**

**Тема:** “ Система моніторингу відвідувань занять студентами ННІНО”

**Виконавець:** Крамар Максим Вікторович

**Керівник:** к.т.н. доц. Радішевський Микола Федорович

**Нормоконтролер:**  к.ф-м.н доц. Оленін Михайло Вікторович

Київ 2019 НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

**Навчально-науковий інститут** інноваційних освітніх технологій

**Кафедра** інженерії програмного забезпечення

**Освітнього ступеня**  бакалавр

**Напрям** 6.050103 Програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач випускової кафедри

Писарчук О.О.

"\_\_\_" \_\_\_\_\_\_\_ 2018 р

ЗАВДАННЯ

на виконання дипломного проекту студента

Крамар Максима Вікторовича

1. Тема проекту: «Система моніторингу відвідувань занять студентами ННІНО»   
затверджена наказом ректора від 21.12.2018 р. № 3084/ст

2. Термін виконання проекту: з 21.12.2018 р. до 28.02.2019 р.

3. Вихідні данні до проекту : програмний продукт розробити за допомогою системи візуального проектування Visual Studio 2017.

4. Зміст пояснювальної записки:

1. Аналіз існуючих засобів моніторингу відвідування студентами занять.
2. Вимоги до програмного засобу .
3. Структура засобу моніторингу відвідувань занять студентами ННІНО.
4. Прототип програмного засобу.

5. Перелік обов'язкових слайдів презентації:

1. Тема, виконавець, керівник.

2. Існуючи методики, аналіз недоліків, постановка завдання.

3. Вимоги до програмного засобу.

4. Структура засобу, діаграма класів

5. Інтерфейс програмного засобу

6. Демонстрація роботи прототипу засобу

6. Календарний план-графік

|  |  |  |  |
| --- | --- | --- | --- |
| № пор | Завдання | Термін виконання | Відмітка про виконання |
| 1. | Ознайомлення з постановкою задачі та вивчення літератури. Складання графіку роботи | 23.12.18 – 25.12.18 |  |
| 2. | Написання 1 розділу, представлення керівнику | 25.12.18 - 30.12.18 |  |
| 3. | Написання 2 розділу, представлення керівнику | 30.05.18 - 15.01.19 |  |
| 4. | Написання 3 розділу, представлення керівнику | 15.01.19 - 20.01.19 |  |
| 5. | Написання 4 розділу, представлення керівнику | 20.01.19 - 25.01.19 |  |
| 6. | Загальне редагування та друк пояснювальної записки, графічного матеріалу | 25.01.19– 04.02.19 |  |
| 7. | Проходження нормо-контролю, перепліт пояснювальної записки. | 05.02.19 – 06.02.19 |  |
| 8. | Розробка тексту доповіді. Оформлення графічного матеріалу для презентації | 07.02.19 – 08.02.19 |  |
| 9. | Отримання відгуку керівника, рецензії. | 07.02.19 – 07.02.19 |  |
| 10. | Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 1 папка, 1 конверт) | 13.02.19 – 14.02.19 |  |

7. Дата видачі завдання 4.11.2018

Керівник:  доцент каф. ІПЗ Радішевський М. Ф.

Завдання прийняв до виконання:  Крамар М.В.

Дата

**РЕФЕРАТ**

Пояснювальна записка до дипломної роботи «“ Система моніторингу відвідувань занять студентами ННІНО»: 40 с., 10 рис., 1 табл., 10 інформаційних джерел.

МОНІТОРИНГ, ВІДСЛІДКОВУВАННЯ ВІДВІДУВАННЯ ЗАНЯТЬ, МЕТОДИСТ, СТУДЕНТ, УСПІШНІСТЬ, АВТОМАТИЗАЦІЯ, ЕЛЕКТРОННИЙ ЖУРНАЛ.

**Об’єкт розробки** – система моніторингу відвідування занять.

**Мета роботи** – покращення ефективності моніторингу відвідувань занять через своєчасне реагування на виникаючі проблеми пропуску занять.

**ABSTRACT**

Explanatory note to the thesis " The software simulator operators recovery skills of air traffic control after a lengthy interruptions ": 40 p. , 10 Fig. , 1 table. , 10 information sources.

MONITORING, RESPONSE TO VISIT, STUDENT, SUCCESS, AUTOMATION, ELECTRONIC JOURNAL.

**Property development** - monitoring system for attending classes.

**Purpose** - Improving the monitoring of attendance at the lessons through timely responses to emerging occupational problems.

**ЗМІСТ**

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ………………………………….…6

ВСТУП 7

~~РОЗДІЛ 1. АНАЛІЗ ВПЛИВУ ЛЮДИНИ НА БЕЗПЕКУ ПОЛЬОТІВ 8~~

~~1.1. Людський фактор як причина авіакатастроф 8~~

~~1.1.1 Аспекти людського фактора, що впливають безпеку~~

~~польотів при УПР 10~~

~~1.1.2. Авіаційні події з вини авіадиспетчерів 11~~

~~1.1.3. Помилки авіадиспетчерів 12~~

~~1.2. Управління повітряним рухом 14~~

~~1.3. Організація роботи та навчання операторів УПР 18~~

~~Висновки 19~~

~~РОЗДІЛ 2. ІСНУЮЧІ СИСТЕМИ КОНТРОЛЮ І ВІДНОВЛЕННЯ НАВИЧОК ДИСПЕТЧЕРІВ ПОВІТРЯНОГО РУХУ 20~~

~~2.1. Роль тренажерів для диспетчерів управління повітряним рухом 20~~

~~2.2. Призначення тренажерів для диспетчерів управління повітряним~~

~~рухом 22~~

~~Висновки 24~~

~~РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ТРЕНАЖЕРУ 25~~

~~3.1. Схема роботи програмного тренажеру 25~~

~~3.2. Проектування функціональних можливостей тренажеру 28~~

~~3.3. Проектування програмного тренажеру 32~~

~~3.4. Обробка результатів тестування 34~~

~~Висновки 36~~

~~. . . .~~

~~ВИСНОВКИ 37~~

~~СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38~~

~~ДОДАТОК А. Текст програми 40~~

**ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ**

ПЗ – програмне забезпечення

ННІНО – навчально науковий інститут неперервної оcвіти

ОС – операційна система

БД – база даних

SOLID – single responsibility, open-closed, Liskov substitution, interface segregation та dependency inversion ([Принцип єдиного обов'язку](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D1%94%D0%B4%D0%B8%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%BE%D0%B2%27%D1%8F%D0%B7%D0%BA%D1%83), [Принцип відкритості/закритості](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B2%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%BE%D1%81%D1%82%D1%96/%D0%B7%D0%B0%D0%BA%D1%80%D0%B8%D1%82%D0%BE%D1%81%D1%82%D1%96), [Принцип підстановки Лісков](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%BF%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8_%D0%9B%D1%96%D1%81%D0%BA%D0%BE%D0%B2), [Принцип розділення інтерфейсу](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D1%80%D0%BE%D0%B7%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D1%83), [Принцип інверсії залежностей](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D1%96%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%97_%D0%B7%D0%B0%D0%BB%D0%B5%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9))

DI – Dependency injection (впровадження залежностей)

API - Application Programming Interface (Прикладний програмний інтерфейс)

**ВСТУП**

Відвідування студентами занять завжди являлась однією із найважливіших питань навчального процесу. Часті пропуски лекцій студентами несе за собою великі проблеми, як для самого студента так і для університету та в кінці і суспільству. Знижується успішність студента і як наслідок знижується його конкурентна спроможність на ринку праці, так як студент не засвоює в повній мірі інформацію з лекцій. В свою чергу це і не раціональна трата часу викладачів. На даний момент моніторинг відвідування занять є актуальною темою це обумовлено тим, що відвідування занять є соціальним явищем яке тісно пов’язане з рішенням соціальних проблем як навчального закладу так і самого студента. Вирішення питань пропуску занять безпосередньо впливає на якість освіти, авторитет навчального закладу та можливість студента в майбутньому бути професіоналом у своїй галузі. Відвідування занять являється одним із обов’язків студентів, недобросовісне їх виконання зв’язано з рядом проблем, в першу чергу, самих студентів, навчального закладу та суспільства.

За останні роки кількість студентів які пропускають заняття тільки збільшилась, традиційні методи моніторингу на теперішній момент не являються досить ефективним інструментом у вирішенні цих питань, так як не дають в повній мірі оцінити масштаби проблеми, також проблемою є вчасно відслідкувати на ранніх етапах пропуски заняття студентами, та причини цих пропусків.

**РОЗДІЛ 1.**

АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ МОНІТОРИНГУ ВІДВІДУВАННЯ СТУДЕНТАМИ ЗАНЯТЬ

На даний момент широко використовується традиційна модель моніторингу відвідування занять

**1.1 Аналіз існуючих систем моніторингу відвідування занять студентами**

На даний момент широко використовується традиційна модель моніторингу відвідування занять. Зазвичай для того щоб відслідковувати пропуски занять студентами, в спеціальному журналі їх помічають як таких, що не були на занятті. Такий журнал є один для певної групи студентів, містить список студентів, лекції які зчитаються даній групі, та конкретні заняття де вказуються дата проведення та тема (рис 1.2). Для того, щоб відмітити відсутніх студентів, викладач, іноді староста, відмічає в журналі відсутніх на лекції студентів, після чого викладач перевіряє коректність відмічених полів про пропуск заняття, записує тему і дату лекції. Після закінчення лекцій на даний день, журнал заносять до особи яка моніторить кількість студентів які пропустили заняття, зазвичай це методист. Методист може перенести дані з журналу у власну таблицю, де зібрані дані по всіх групах та студентах. В кінці місяця, іноді раніше, підбиваються підсумки по відвідуванню занять студентами, що є не зовсім ефективним способом.

З розвитком інформаційних технологій проблема використання паперових журналів має бути переглянутою, підхід із застосуванням традиційних методів є застарілим, не гнучким, та потребує більше часу на прийняття певних рішень.

Кафедра ННІІОТ

НАУ 19 11 07 000 ПЗ

*Листів*

*Літ.*

*Розроб.*

Крамар М.В.

*Лист*

8

8

Система моніторингу відвідувань занять студентами ННІНО

*Керівник.*

Радішевський М.Ф.

ПІ-501бз - 6.050103

*Н. Контр.*

Оленін М.В.

Моніторинг відвідування студентами занять з використанням журналів маю певні недоліки.

-Не враховується причина відсутності студента на занятті.

-Можливість втручання в існуючі записи недобросовісними студентами.

-Велика кількість часу витрачається на підрахунок пропущених занять.

-Неможливість аналізу причин прогулу занять студентами.

Технічний прогрес не стоїть на місці і весь світ переходить з паперових носіїв даних на цифрові, так як останні являються більш зручними. Засобами моніторингу якими користуються вже багато років не можуть в повній мірі забезпечити всі аспекти виявлення проблем які виникають при пропуску занять студентами та пов’язані з цим наслідки.

Розроблювана система покликана покращити та автоматизувати процеси моніторингу відвідування занять, створити такі умови які б допомогли поліпшити ситуацію яка склалась за останні роки, коли студенти пропускають велику кількість лекцій, або їх відмічають колеги, коли насправді студент, по суті, не на парі. Дана система дозволить усунути ці проблеми. Відмічати відсутніх студентів повинен тільки авторизований користувач з певними правами.

Якість освіти – один характеристика системи освіти, яка відображає ступінь реальних освітніх процесів які відповідають нормам освіти, соціальним і особистим очікуванням. Також це - це комплекс характеристик освітнього процесу, що визначають послідовне та практично ефективне формування компетентності та професійної свідомості. Тут можна виділити три групи характеристик: якість потенціалу досягнення мети освіти, якість процесу формування професіоналізму та якість результату освіти.

Моніторингом системи якості освіти називається спеціальна система яка дозволяє збирати інформацію про стан освіти, прогнозувати наслідки певних проблем які можуть виникнути, на основі наданих даних.

Однією із основних причин падіння успішності студента являється часті пропуски занять. Для підвищення рівня освіти студентів потрібно мати сучасну систему моніторингу відвідування занять, яка в режимі реального часу відслідковує кількість студентів які не присутні на парах, загальну кількість пропусків по кожному із них. Потрібно мати механізми впливу на студентів, контактні данні для зв’язку з студентом, або його батьками.

Існуючі методи моніторингу відвідувань, являються морально застарілими в наш час журнали старого формату (рис 1.1) повинні відійти в минуле.



Рис. 1.1 Журнал навчальних занять

Паперові журнали громіздкі, не довговічні, їх можна втратити, зіпсувати. Для того щоб відмітити студентів як відсутніх на парі потрібно, фізично принести журнал на лекцію, Проблеми доступу до даних в один момент часу різними користувачами до журналу на даний момент теж є актуальною проблемою, наразі одночасно не можна вносити корективи чи вибирати дані. Система моніторингу студентів ННІНО дозволить одночасно користуватись журналами з різних пристроїв в один момент часу. Що позитивно відобразиться на продуктивності навчального закладу.

Для методистів, дуже важливо мати оперативну, об’єктивну інформацію про дійсний стан навчального процесу. За необхідності це дозволяє своєчасно здійснити методичну підтримку і ввести необхідні корективи. Таку інформацію надають регулярно здійснені моніторингові дослідження, які є хорошим інструментом аналізу різноманітних сторін навчального процесу. Вони дозволяють одержати реальну картину результативності нововведень, напрямів плану розвитку навчального закладу, від мети дослідження можна виділити найрізноманітніші напрями моніторингу. Однак найважливішим з них було і залишається визначення результативності навчального процесу.

При систематичному моніторингу за відвідуванням і своєчасним прийнятті рішень, що до проблем які виникають з відвідуванням, дозволяє дотримуватись стабільних результатів відвідування лекцій, що в свою чергу веде до підвищення рівня знань студентів, дозволяє зберегти студенів, не виключати їх з університету. Студенти з високим рівнем знань предметної області стають конкурентно спроможними на ринку праці, що підвищує рівень авторитету університету, що дає великі перспективи розвитку навчального закладу, його колективу, та студентів які там навчаються.



Рис 1.2 Журнал в розгорненому вигляді

Журнали якими користуються в даний час не дають тих можливостей які може дають їх електронний аналог. Паперові журнали сильно обмежені простором, на 1-2 сторінок об’єктивно не вміститься вся інформація яка, теоретично, може знадобитися для прийняття рішень пов’язаних з підвищенням рівня освіти. Існуючі журнали не можуть вмістити інформацію про те чому певний студент був відсутній, не має прикріпленої довідки, у випадку коли студент хворіє, або не можна вказати в певному полі, якщо у студента була поважна причина. Розроблювана система допоможе вдосконалити журнали якими користуються вже багато років, нова система матиме змогу не просто вказати що студент відсутній, а й вказати причину, вибрати одну із (хворий, має поважну причину, був відсутній без поважної причини) а також написати коментар для уточнення інформації про відсутність студента (за бажанням).



Рис 1.3 Інструкція з ведення журналу, властивості якими він володіє

Електронний журнал є набагато кращим за свого попередника і тим, що дозволяє в повній мірі вказувати всю інформацію по предмету, чи конкретному уроку, не обмежуючись полями журналу, також дає змогу вносити нові поля, які б у звичайному журналі не могли б вміститися.

Змога розширятись є великим плюсом розроблюваної системи, вона може без великих зусиль створювати нові записи, нові групи, зберігати довгий час інформацію про відвідування занять тими групами, які закінчили своє навчання у минулому, можна порівнювати інформації про відвідування занять студентами які навчались в минулих роках з тими результатами які існують на даний момент та створювати статистику, щоб прослідкувати, на що варто звернути увагу, або що змінити. На відміну від паперових аналогів для збереження журналів не потрібно багато місця, певного приміщення, це є плюсом переходу на електронну систему.

Висновок

Електронна система моніторингу відвідування занять студентами ННІНО є кращою у всьому, ніж журнали які використовують зараз, буть то тим, що системою можуть користуватись декілька людей одночасно, вона не потребує фізичної присутності журналу у кожного хто з ним працює. Така система є надійною, довговічною, захищеною від несанкціонованого втручання зовні. Надає розширені можливості в порівняні з тим що є на даний момент.

**РОЗДІЛ 2.**

ВИМОГИ ДО ПРОГРАМНОГО ЗАСОБУ

Специфікація вимог до програмного забезпечення – це повний опис поведінки системи яку розробляють . Вона включає множину прецедентів що описують всі дії, які користувачі проводять з програмним продуктом, а також функціональними вимогами.

* 1. **Вимоги представлення інформації**

Сформулюємо ряд вимог яким повинен відповідати програмне забезпечення модулю табельного обліку в ході виконання робот:

• створення інтуїтивно зрозумілого інтерфейсу;

• створення зрозумілих та зручних форм робіт з первинною інформацією;

• точність розрахунку та виведення інформації за визначеними запитами;

•автоматичний розрахунок кількості днів фактичної роботи і відпрацьованих годин;

• захист від некоректного введення інформації;

• наявність довідкової системи;

• забезпечувати роздруківку таблиць бази даних або результатів запитів у відформатованому вигляді;

• забезпечувати функції резервного збереження та відновлення даних;

Кафедра ІПЗ

НАУ 19 11 07 000 ПЗ

*Листів*

*Літ.*

*Розроб.*

Крамар М.В.

*Лист*

16

6

Система моніторингу відвідувань занять студентами ННІНО

*Керівник.*

.Радішевський М.Ф.

ПІ-501бз - 6.050103

*Н. Контр.*

Оленін М.В.

**2.2 функціональні вимоги**

Функціональні вимоги описують сервіси, які надаються програмним середовищем, його поведінку в певних ситуаціях, реакцію на ті чи інші вхідні дані і дії які система дозволить виконувати користувачам.

Авторизація користувача в системі. Користувач має мати можливість увійти в систему за допомогою свого логіну та паролю

Реєстрація користувача в системі

Користувач може зареєструватись в системі ввівши свій логін та пароль.

Ролі в програмі

Програма повинна містити декілька ролей для інкапсуляції деякої реалізації, доступу до функціоналу користувачам у яких роль є меншого рівня доступу. Загалом потрібно розробити 4 ролі, адміністратор, викладач, методист, староста. Адміністратор має право на доступ до будь яких елементів програми, в тому числі присвоєння користувачам нових ролей. Викладач повинен мати змогу відмічати присутніх студентів, записувати тему уроку, дату проведення заняття, відмічати причину відсутності студента. Методист має право на перегляд таблиць студентів, моніторинг їх відвідування занять, у разі проблем приймати міри, отримати номер телефону студента чи його батьків. Також методист має право вносити зміни в таблицю студентів, груп, проблемних студентів.

Операції над студентами

Передбачити можливість створення нових студентів, списку студентів, заповнення персональні данні, видаляти, обновляти їх.

Відслідковування успішності

Розробити підсистему яка б автоматично відслідковувала відсутніх студентів та в разі необхідності сповіщала про проблеми з відвідуванням методиста, для прийняття рішень до того як це може стати більшою проблемою.

Відмічання причини відсутності студента

Передбачити можливість вказувати не просто те, що студент являється відсутнім, але і вказати причину по якій студент не з’явився на лекцію.

**Нефункціональні вимоги**

Програмний продукт має бути створений за дотриманням шаблону трьох рівневої архітектури (рис 2.1), де кожен рівень являє собою окремий проект який було б можливо підмінити на інший, у разі вдосконалення чи переходу на нові технології.

Перший рівень являє собою рівень доступу до даних, він має містити тільки логіку роботи з БД, створення нових моделей, їх видалення, редагування, вибірку за певним параметром та вибірку всіх елементів з певної таблиці. В цьому проекті також повинно бути присутні репозиторії, які і наддадуть нам змогу у зручній формі працювати з базою даних. Кожен логічний модуль має описуватись в окремому класі для легшого читання та навігації по коду. Класи зі схожими властивостями мають бути об’єднані в окремі для кожної групи namespaces.

Другий рівень повинен містити в собі логіку обробки даних з рівня представлення та передача їх на рівень доступу до даних. В даному проекті повинні бути присутні класи Data transfer object які дозволять нам проектувати моделі з рівня доступу д даних на моделі бізнес логіки, що дасть певні плюси у питаннях безпеки та відсутності тих класів, які на рівні бізнес логіки не потрібні для автоматичного проектування одних моделей на інші рекомендовано використати одну із програм автоматичного проектування сутностей, наприклад Automapper. Даний рівень також має містити власні класи для обробки помилок який потрібно буде унаслідувати від базового, стандартного класу Exeption та розширити його функціонал необхідними нам властивостями у класі насліднику. Для більшої гнучкості програми потрібно використовувати слабкий зв'язок між класами, для цього потрібно використати інтерфейси а також принцип Dependency injection який якраз і дозволяє будувати застосунок із слабкою зв’язаністю, для легшого тестування і можливості швидкої заміни даних класів на інші. Для впровадження таких залежностей потрібно використати IoC контейнер наприклад Ninject він дасть змогу в зручний спосіб вказати який клас використовувати заміть інтерфейсу, так як для зв’язку класів за типом композиція конструктор класу буде приймати посилання на інтерфейс а не конкретну його реалізацію. В програмі повинні бути описані інтерфейси для кожного класу бізнес логіки який буде в програмі. Обов’язково в застосунку повинні бути сервіси, які і будуть виконувати обробку значень які нам надходять з рівня представлення. Кожен сервіс описує певну логічну одиницю, для написання коду який буде легко підтримувати потрібно дотримуватись принципів SOLID.

Рівень представлення повинен надавати методи взаємодії користувача з нашою програмою. Даний проект має в собі представлення у вигляді HTML сторінки (чи її аналогів) для виведення інформації з нашої бази даних на екран. Повинна мати елементи керування нашими моделями, інтерфейсом взаємодії з кодом. Логічна частина даного проекту знаходиться в контролерах. Контролери мають містити в собі тільки базові функції валідації даних які передаються в застосунок користувачем, контролери не мають містити зайвий код, якщо виникає необхідність внести розширену логіку в контролер, то краще задуматись над тим, що перенести цю логіку в сервіси бізнес рівня. На рівні представлення також повинні бути присутніми ViewModels які будуть схожими на моделі Data transfer objects тільки з поправкою на те, що саме буде виводитись на сторінку представлення, моделі які використовуються в даному проекті і передаються на рівень бізнес логіки повинні бути перетворені в Data transfer objects, для цього можна скористатись фреймворком який автоматично спроектує схожі моделі класу одна на іншу для відправки обєктів між проектами.

На рівні представлені повинні бути присутніми крім звичайних контролерів, WEB API контролери які є більш гнучкою версією звичайних контролерів, WEB API контролери дають змогу звертатись до нашого застосунку не тільки з сайту який звичайно відображає інформацію з БД і надає можливість роботи з наданою інформацією. API контролери можна використовувати з будь яких пристроїв які мають підключення до інтернету, головна ідея цих контролерів є те, що вони можуть при звернені до них за певною адресою ( зазвичай, ім’я веб сайту, Апі, і тип запиту(GET, POST, PUT, DELETE)). API дає можливість легкого розширення застосунку на інші пристрої, або сайти, через надання відкритих контролерів за посиланням.

**2.2 Розробка інтерфейсу для програмного забезпечення**

Інтерфейс – засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відображення інформації, максимально пристосованих для зручності користувача; у графічних системах інтерфейс реалізовується багато віконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштовуваннями як самих вікон, так і окремих їхніх елементів

Надійність системи

Система має стабільно працювати без збоїв цілодобово.

Застосунок повинен підтримувати можливість працювати з різними пристроями.

Система повинна швидко реагувати на дії користувача, операції мають виконувати в продовж, не більше ніж, 250 Мілесекунд.

Написання програмного коду з використанням програмної мови C#

Використання трьох рівневої архітектури, розділення рівня доступу до даних, бізнес логіки та представлення.

Використати в якості бази даних СУБД SQL server

Для проектування сутностей програмного коду на базу даних використати ORM Entity Framework

Застосунок повинен бути побудований на клієнт серверній архітектурі.

**2.3 Вимоги до побудови бази даних**.

БД має включати в себе такі сутності:

* Студент - описує властивості кожного студента, персональні дані.
* Викладач – описує персональні дані, та інформацію про посаду, лекції які він проводить.
* Група – містить набір студентів, назву. Група об’єднує студентів в один підрозділ.
* Адреса – містить інформацію про місце проживання студентів та викладачів.
* Лекція – описує лекцію, її назву, викладач, що її проводить.
* Конкретна лекція – містить відомості про певну лекцію, дату проведення, викладача, та опис теми заняття.
* Проблемний студент – таблиця яка вказує на те, що студенти які в ній містяться мають серйозні проблеми з відвідуванням занять.
* Присутній студент – таблиця яка описує кількість пропущених занять певного студента, причину відсутності, додаткові відомості про відсутність.

**РОЗДІЛ 3.**

СТРУКТУРА ЗАСОБУ МОНІТОРИНГУ ВІДВІДУВАНЬ ЗАНЯТЬ СТУДЕНТАМИ ННІНО



**Рисунок 2.1 схема трьох рівневої архітектури**

Кафедра ІПЗ

НАУ 19 11 07 000 ПЗ

*Листів*

*Літ.*

*Розроб.*

Крамар М.В.

*Лист*

22

21

Система моніторингу відвідувань занять студентами ННІНО

*Керівник.*

.Радішевський М.Ф.

ПІ-501бз - 6.050103

*Н. Контр.*

Оленін М.В.

**РОЗДІЛ 4.**

ПРОТОТИП ПРОГРАМНОГО ЗАСОБУ

прототип. Це опис результату, скриншоти роботи програми і т д.

1. ~~У більшості випадків авіаційних подій (АП) - 71% виникнення і розвиток аварійної ситуації на повітряних судах було обумовлено людським фактором. Основні причини – недостатня підготовка до польотів у даних умовах, недотримання екіпажами встановлених параметрів руху при заході на посадку, порушення вимог керівних документів, відсутність належної взаємодії в екіпажі і порушення встановлених правил експлуатації.~~ 
   * 1. **~~Аспекти людського фактора, що впливають на безпеку польотів при УПР.~~**

~~< Заповнювати сторінку до кінця > отступи до рамки: зліва – 20 мм, зверху, знизу справ – 5 мм,~~

Кафедра ІПЗ

НАУ 19 11 07 000 ПЗ

*Листів*

*Літ.*

*Розроб.*

Крамар М.В.

*Лист*

24

13

Система моніторингу відвідувань занять студентами ННІНО

*Керівник.*

.Радішевський М.Ф.

ПІ-501бз - 6.050103

*Н. Контр.*

Оленін М.В.

**4.1**  **Класи сутності, моделі Code First Entity Framework 6**



**Рисунок 4.1 сутності які проектуються на базу даних**

public class EFContext : DbContext

   {

       public EFContext(string connectionString)

             : base(connectionString)

       { }

       public EFContext()

           : base("DefaultConnection")

       {

       }

       public DbSet<Address> Addresses { get; set; }

       public DbSet<ConcreteLesson> ConcreteLessons { get; set; }

       public DbSet<Group> Groups { get; set; }

       public DbSet<Lesson> Lessons { get; set; }

       public DbSet<PresetStudent> PresetStudents { get; set; }

       public DbSet<ProblemStudent> ProblemStudents { get; set; }

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

       public DbSet<Teacher> Teachers { get; set; }

   }

**Рисунок 4.2 клас контекст через який ми працюємо з нашою БД**

 **Рис. 4.3 клас репозиторій та його інтерфейс**

public class EFGenericRepository<TEntity> : IGenericRepository<TEntity>

where TEntity : class

    {

        EFContext \_context;

        DbSet<TEntity> DBEntity;

        public EFGenericRepository(EFContext context)

        {

            \_context = context;

            DBEntity = context.Set<TEntity>();

        }

        public async Task<IEnumerable<TEntity>> GetAllAsync()

        {

            var result = await DBEntity.ToListAsync();

            return result;

        }

        public async Task<TEntity> GetAsync(int id)

        {

            var result = await DBEntity.FindAsync(id);

            return result;

        }

        public IEnumerable<TEntity> GetAll()

        {

            var result = DBEntity.ToList();

            return result;

        }

        public TEntity Get(int id)

        {

            var result = DBEntity.Find(id);

            return result;

        }

        public void Create(TEntity item)

        {

            DBEntity.Add(item);

        }

        public void Update(TEntity item)

        {

            \_context.Entry(item).State = EntityState.Modified;

        }

        public void Delete(int? id)

        {

            var item = DBEntity.Find(id.Value);

            if (item != null)

{

                 DBEntity.Remove(item);

}

        }

        public IEnumerable<TEntity> Find(Func<TEntity, bool> predicate)

        {

            return \_context.Set<TEntity>().Where(predicate).ToList();

        }

    }

Вище приведено клас з застосунку. Шаблонний репозиторій, дає можливість створювати на основі цього класу, імплементації репозиторіїв параметризовані конкретним типом. В даному коді описано основні операції які ми можемо проводити над моделями, створювати, вибирати, видаляти, оновлювати, шукати. Шаблонний репозиторій допомагає зменшити кількість коду який повторюється в програмі, за рахунок використання техніки підстановки конкретного типу.

**4.2**  **Патерн Unit of work**

Патерн Unit of work (рис. 3.4) слідкує за всіма діями застосунку, які можуть змінити базу даних в рамках одного бізнес процесу. Коли бізнес дія завершується, Unit of work виявляє та зберігає внесені зміни в БД. Даний патерн гарантує нам, що коли ми будемо створювати різні репозиторії для інкапсулювання роботи з джерелом даних, всі вони будуть використовувати один і той самий контекст даних.

public interface IUnitOFWork

   {

       IGenericRepository<Student> StudentUOW { get; }

...

void Save();

  }

public class EFUnitOfWork : IUnitOFWork

   {

       private EFContext DB\_Context;

       private EFGenericRepository<Student> StudentRepository;

...

       public EFUnitOfWork(string connectionString)

       {

           DB\_Context = new EFContext(connectionString);

       }

       public IGenericRepository<Student> StudentUOW

       {

           get

           {

               if (StudentRepository == null)

                   StudentRepository = new EFGenericRepository<Student>(DB\_Context);

               return StudentRepository;

           }

       }

...

Вище приведений уривок програмного коду який описує, на прикладі, властивості студента, логіку роботи патерна, створення репозиторія для студентів якщо його не існує, або надання доступу до цього репозиторію, якщо він був раніше створений.



**Рис. 4.4 клас патерну Unit Of Work**

Таблиці бази даних які надають можливість реєстрації та авторизації користувача в системі (рис.3.5) дані таблиці є згенеровані з фреймворком для авторизації користувачів в системі яка використовує ASP, база даних має таблиці ролей, по яким надають права доступу користувачів до певних контролерів (користувачі з різними ролями мають різний доступ до функціоналу застосунку). Таблиця Users містить дані про користувача, його email, імя, пароль. Таблиця ClientProfile має розширені відомості про користувача. \_MigrationHistory – таблиця яка мітить інформацію про всі зміни в базі даних які вносили, будь то додані нові властивості, їх видалення, та зміни.



**Рисунок 4.5 база даних для авторизації**

Основна База даних застосунку (рис.3.2) містить 8 таблиць які є повязані між собою. Таблиця Addresses містить відомості про місце проживання студентів та викладачів.

**Lessons** містить поля які вказують назву предмету та викладача який веде лекції.

**ConcreteLessons** – таблиця яка відображає відомості проконкретну лекцію викладача за певним предметом. Вона містить в собі, опис лекції, дату проведення, іентифікатор викладача та предмету до якого відноситься дана лекція.

**Group** – описує групу студентів, містить назву групи та посилається на студентів через зв'язок один до багатьох.

**Teacher** – таблиця яка має в собі опис властивостей якими володіє вчитель його імя, прізвище,ідентифікатор предмету, електрону пошту, посаду викладача, посилання на таблицю адреси.

**Students** – таблиця яка містить в собі опис студента, його ім’я, прізвище, дату народження, телефонний номер студента, телефонний номер батьків, електронну пошту та посилання на адресу та групу студена.

**PresentStudent** – таблиця яка допомагає нам зберігати дані про пропуски занять студентом, за цією таблицею ми можемо побачити коли студент не був на лекції, причину чому він пропустив заняття, будь то через хворобу, з поважної причини чи без причини, при необхідності також передбачене поле для внесення додаткових відомостей, чому студент не був на лекції. Через поле яке вказує скільки разів студент пропустив заняття, методист може отримати вибірку студенті в яких дане поле містить значення кількості пропущених занять більше ніж допустиме, наприклад якщо студент пропустив більше 7 лекцій він може потрапити в таблицю.

**ProblemStudents** – зберігаються посилання на всіх студентів в яких є проблеми з відвідуванням занять.



**Рисунок 4.6 схема бази даних**

**4.4 Інтерфейси програми**

Програма використовує інтерфейси для більш гнучкої взаємодії класів які реалізують ці інтерфейси, так як ми можемо з легкістю підміняти реалізацією цих інтерфейсів на інші класи, що дає змогу ефективно, швидко та зручно, при необхідності використовувати конкретну реалізацію кожного інтерфесу.

Інтерфейси написані таким чином, що кожен індивідуальний інтерфес виконує певну, покладену на нього задачу. IGenericService – являється контрактом для реалізації, в ньому об’єднані операції роботи з шаблонними сутностями, в собі він містить CRUD операції. Враховуючи, що даний інтерфейс являється шаблонним, то ми можемо використовувати його з різними типами даних, які являються класами. Основною перевагою написання інтерфейсів являється можливість в подальшому використання впровадження залежностей (Dependency injection) що є рекомендованим для програм які можуть в подальшому розширюватись. За допомогою цього механізму полегшується процес тестування програми. Кожен інтерфейс повинен описувати конкретну задачу і не містити лишніх, не притаманної для цієї задачі функціональної частини, так як це порушуватиме один із принципів SOLID а саме розділення інтерфейсів, не дотримання цих рекомендацій веде до погіршення розуміння коду, збільшення непотрібного коду в класах які реалізують цей інтерфейс, але не потребують тих додаткових функцій які в ньому описані.



**Рисунок 4.7 інтерфейси сервісів**

**ITeacherService**(рис. 3.7) містить в собі ті методи які належать до безпосередньо тих обовязків які повинен виконувати тільки вчитель, в подальшому клас який реалізує цей інтерфейс може і доповнити його своєю реалізацією.  
 **ILessonService –** мітить опис метоів які стосуються суто лекції. Додання нової лекції, викладачем, вказання дати проведення леекції, та теми лекції.

**IMethodistService** – інтерфейс який описує методи, що застосовуються методистом, для вибірки студентів, вчителів, пошуку студентів які мають поблеми з відвідуванням занять.

**IUserService –** інтерфейс який надає нам змогу реалізації реєстрації нового користувача в системі.

**IGenericService<T>** - інтерфейс який на відміну від інших в цій групі являється шаблонним, що дає нам змогу використовувати його з різними типами. Особливістю цього інтерфейсу являється те що він надає можливість для створення, редагування, видалення, отримання даних з БД різних сутностей, які повинні параметризувати місце заповнення типу <T>.

**4.5 Класи сервіси**

Основною метою сервісів є реалізація функціональної частини програми, в них описуються логіка програми, обробка запитів, передання результатів на рівень доступу до даних, обробки помилок, якщо такі виникатимуть.



**Рисунок 4.8 класи сервіси**

class MethodistService : IMethodistService, IGenericService<StudentDTO>

    {

        IUnitOFWork UnitOfWork { get; set; }

        IUnitOFWork UnitOfWork { get; set; }

        public MethodistService(IUnitOFWork \_UnitOfWork)

        {

            UnitOfWork = \_UnitOfWork;

        }

        public IEnumerable<StudentDTO> GetItems()

        {

            IEnumerable<Student> students = UnitOfWork.StudentUOW.GetAll();

            var mapper = new MapperConfiguration(cfg => cfg.CreateMap<Student, StudentDTO>()).CreateMapper();

            return mapper.Map<IEnumerable<Student>, List<StudentDTO>>(students);

        }

        public void CreateItem(StudentDTO instanceDTO)

        {

            if (instanceDTO == null)

{

                throw new ArgumentNullException();

}

            var mapper = new MapperConfiguration(cfg => cfg.CreateMap<StudentDTO,

Student>()).CreateMapper();

            var student = mapper.Map<StudentDTO, Student>(instanceDTO);

            UnitOfWork.StudentUOW.Create(student);

            UnitOfWork.Save();

        }

**MethodistService –** сервіс призначений для роботи методиста, інкапсулює в собі методи для роботи з студентами, перевірки успішності, виведення інформації про пропущені заняття студентами, а також реагування на систематичні пропуски студентами.

**TeacherService** – сервіс який інкапсулює в собі методи та властивості з якими працюватиме викладач, даний клас містить методи повязані з навчальним процесом, отримання списку групи, списку студентів, можливість відмітити відсутніх студентів на парі, вказати причину відсуності.

**StudentService –** даний сервіз призначений для створення, оновлення, видалення, отримання інформації про студента, або отримання списку студентів.

**UserService –** представляє набір методів для реєстрації та авторизації користувача в системі.

**ServiceCreator** – клас який має в собі метод, який створює новий клас UserService приймаючи строку підключення до БД, та повертає екземпляр класу приведеного до інтерфейсу IUserService.

**3.9 Контроллери**

Контроллери обробляють запити від користувача приймаючи певні данні в Json, XML форматі та передають їх на рівень бізнес логіки.



**Рисунок 4.9 класи контроллерів**

[Authorize(Roles = "Methodist")]

   [RoutePrefix("api/Methodist")]

   public class MethodistController : ApiController

   {

       IStudentService StudentService;

       public MethodistController(IStudentService serv)

       {

           StudentService = serv;

       }

       [HttpGet]

       public IHttpActionResult GetListOfStudents()

       {

           IEnumerable<StudentDTO> studentDTO = StudentService.GetItems();

           if (studentDTO == null)

               throw new HttpResponseException(System.Net.HttpStatusCode.NotFound);

           return Ok(studentDTO);

       }

       [AllowAnonymous]

       [HttpGet]

       [Route("details/{id:int}")]

       public IHttpActionResult GetStudentById(int? id)

       {

           if (id == null)

               return NotFound();

           StudentDTO studentDTO = StudentService.GetItem(id);

           if (studentDTO == null)

               throw new HttpResponseException(System.Net.HttpStatusCode.NotFound);

           var mapper = new MapperConfiguration(cfg => cfg.CreateMap<StudentDTO, StudentViewModel>()).CreateMapper();

           StudentViewModel studentView = mapper.Map<StudentDTO, StudentViewModel>(studentDTO);

           //  StudentService.CreateItem(await StudentService.GetItemAsync(id));

           return Ok(studentView);

       }

       //[HttpPost("upload")]

       [HttpPost]

       [Route("details/{id:int}")]

       public IHttpActionResult CreateNewStudent([FromBody]StudentViewModel

studentView)

       {

           if (!ModelState.IsValid)

               throw new HttpResponseException(System.Net.HttpStatusCode.BadRequest);

           var studentDTO = new StudentDTO

           {

               // studentDTO.Id = studentView.Id;

               Name = studentView.Name,

               Surname = studentView.Surname

           };

           StudentService.CreateItem(studentDTO);

           return Ok();

       }

       // PUT: api/Student/5

       [HttpPut]

       public IHttpActionResult UpdateStudentById(int? id, [FromBody]StudentViewModel studentView)

       {

           if (!ModelState.IsValid)

               throw new HttpResponseException(System.Net.HttpStatusCode.BadRequest);

           if (id == null)

               return BadRequest();

           StudentDTO item = StudentService.GetItem(id); //get student by given id

                                                         // item.Id = studentView.Id;

           item.Name = studentView.Name;

           item.Surname = studentView.Surname;

           StudentService.EditItem(item);

           return Ok();

       }

**ЗРАЗОК ОФОРМЛЕННЯ НАСТУПНИХ АРКУШІВ РОЗДІЛУ ДИПЛОМНОЇ РОБОТИ** ( для бакалаврів)

(Текст наступних аркушів розділу)

**ВИСНОВКИ**

Дипломна робота описує автоматизовану систему моніторингу студентів ННІНО. Розглянуто методи моніторингу які використовувались в минулому, які використовують зараз, вказано на недоліки існуючих методі. Описано важливість відслідковування відвідування занять студентами, вказано як часте відвідування занять поліпшить ситуацію з не тільки для самих студентів, а й для навчального закладу та суспільства в цілому.

В дипломному проекті описане проектування системи моніторингу відвідування занять. Вдалось реалізувати програмний засіб який дозволить поліпшити ситуацію з недобросовісним відвідуванням занять студентами. Має можливість використовувати методи які на ранніх етапах допоможуть виявити студентів які часто пропускають лекції, та прийняти міри по усуненню даних проблем.

Даний програмний продукт сприятиме поліпшенню ситуації з присутністю студентів на лекціях, отримання детальної інформації про причини відсутності, тих чи інших студентів. Допоможе скоротити час на виявлення проблеми та її вирішення, що сприятливо відобразиться на розвитку як студента так і навчального закладу.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Розробка інформаційної системи керуванням навчальним процесом кафедри. М.М. МАЛЬКО, О.Ю. КРОПАЧЕК, О.Т. МИРОШНИК 2007

2. В. Е. Кабикин «Диагностика оперативного мышления», К., «Наукова думка», 1977. - 110с.

3. Джефрі Ріхтер CLR via C# 2013. - 810 c.

4. ASP.NET MVC Адам Фрімен. 2015. - 745 c.