МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ВАРХОЛЯК Олександр Степанович

Мобільна програма для обліку успішності в академічній групі/ Mobile software for accounting the excellence in academic group

напрям підготовки: 6.050103 - Програмна інженерія
фахове спрямування - Програмне забезпечення систем

Бакалаврська дипломна робота

Виконав студент групи ПЗС-41
О. С. Вархоляк
_____

Науковий керівник:
к.т.н., доцент МАРЦЕНЮК Є.О.
_____

Бакалаврську дипломну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2016

РЕЗЮМЕ

Дипломна робота містить 71 сторінки, 10 таблиць, 28 рисунків, список використаних джерел із 16 найменувань.

Метою дипломної роботи є розробка мобільного програмного продукту для обліку успішності в академічній груп.

Об’єктом дослідження є процес внесення інформації про проведені заняття та оцінювання знань студентів.

Предметом дослідження є програмний продукт обліку успішності в академічній груп.

Методи розробки базуються на технології Java.

Одержані результати полягають в розробці мобільного програмного забезпечення обліку успішності в академічній груп, яке може використовуватися на мобільних операційних системах.

Ключові слова: журнал, академічна група, тема заняття, оцінка.

RESUME

Thesis contains 71 pages, 10 tables, 28 figures, list of references with 16 titles.

The aim of the thesis is to develop mobile software to account for success in academic groups.

The object of research is the process of entering information about the lesson and assessment of student learning.

The subject of the study is the software of the success in academic groups.

The methods of making technology based on Java.

The results are in developing mobile software of the success of academic groups that can be used on mobile operating systems.

Keywords: magazine, academic group, topic sessions, evaluation.

ЗМІСТ

ВСТУП

Актуальність теми

Конкурентоспроможність будь-якого вищого навчального закладу залежить від його здатності і уміння готувати кваліфікованих фахівців, якість яких не тільки задовольняє вимогам споживачів і всіх зацікавлених сторін, але і перевершує їх очікування.

При перевірці знань, умінь і навичок велике значення має їх об'єктивна та незаангажована оцінка до якої висуваються наступні вимоги: оцінка повинна бути об'єктивною і справедливою, ясною і зрозумілою– для студента, оцінка повинна виконувати стимулюючу функцію та орієнтувати на– навчання, оцінка повинна бути всебічною та враховувати усі види навчальної– діяльності

Мета і задачі розробки

Метою розробки є створення мобільної програми для обліку успішності в академічній групі. Програма повинна відповідати наступним критеріям:

- бути мобільною ;
- мати зручну форму для введення необхідних даних;
- мати здатність програми імпортувати і експортувати дані;
- наявність різних додаткових інструментів (нагадувалки);
- якісна система захисту даних і синхронізація з усіма пристроями.

В процесі розробки необхідно вирішити наступні задачі:

зробити огляд існуючих програм для обліку успішності в академічній групі;

визначити необхідні функції для реалізації програми;

провести тестування програмного засобу;

Методи, засоби та технології розробки

У даному випадку було вибрано методи аналізу та синтезу. Аналіз — це метод пізнання, який дає змогу поділити предмет на частини. Синтез, навпаки, є наслідком з'єднання окремих частин чи рис предмета в єдине ціле.

Аналіз та синтез взаємопов'язані, вони являють собою єдність протилежностей. Залежно від рівня пізнання об'єкта та глибини проникнення в його сутність застосовуються аналіз і синтез різного роду.

Практичне значення одержаних результатів

Практичне значення розробки полягає у створенні мобільної програми для обліку успішності в академічній групі.

РОЗДІЛ І
РОЗДІЛ АНАЛІЗУ ТА СПЕЦИФІКАЦІЇ ВИМОГ СИСТЕМИ «ОБЛІКУ УСПІШНОСТІ В АКАДЕМІЧНІЙ ГРУПІ»

1.1. Коротка характеристика об’єкту управління

Тематика розробки програмного забезпечення для контролю успішності студентів не є новою але актуальною. Як зазначено у [1] «Конкурентоспроможність будь-якого вищого навчального закладу залежить від його здатності і уміння готувати кваліфікованих фахівців, якість яких не тільки задовольняє вимогам споживачів і всіх зацікавлених сторін, але і перевершує їх очікування. Якість одержаних знань характеризує ефективність спільної навчальної роботи професорсько-викладацького складу і студентів. Об’єктивне уявлення про якість знань студентів можна одержати тільки при систематичному контролі навчальних досягнень студентів. Результати навчальної роботи та якість знань студентів, виражаються в оцінках. Оцінка - це визначення ступеня засвоєння студентами знань, умінь і навичок відповідно до вимог програм та керівних документів, по яких здійснюється навчання. При перевірці знань, умінь і навичок велике значення має їх об'єктивна та незаангажована оцінка до якої висуваються наступні вимоги: оцінка повинна бути об'єктивною і справедливою, ясною і зрозумілою– для студента, оцінка повинна виконувати стимулюючу функцію та орієнтувати на– навчання, оцінка повинна бути всебічною та враховувати усі види навчальної– діяльності. Сам процес оцінювання знань студентів з навчальних дисциплін зазвичай здійснюється на основі результатів поточного модульного контрою і підсумкового контролю знань [2]. Поточний контроль здійснюється під час проведення занять і має на меті перевірку рівня підготовленості студента до виконання конкретної роботи. Поточний контроль реалізується у формі опитування, захисту лабораторних робіт,

виступів на семінарських та практичних заняттях тощо. Модульний контроль є складовою поточного контролю і здійснюється в формі виконання студентом модульного контрольного завдання (контрольної роботи, тесту, колоквіуму тощо). Форма проведення поточного контролю під час навчальних занять та модульного контролю визначаються викладачами, які читають відповідні дисципліни. Кількісним вираженням процесу оцінювання є відмітка. Відмітка - це результат процесу оцінювання, умовно - формальне (знакове), кількісне вираження оцінки навчальних досягнень у цифрах, буквах або іншим чином. Відмітка - це своєрідний орієнтир, що відображає соціальні вимоги до змісту освіти, до рівня оволодіння їм студентів, дієвий регулятив їх навчальної діяльності.»

1.2. Огляд і аналіз існуючих аналогів, що реалізують функції системи обліку успішності в академічній групі

1.2.1. Програмний модуль "ПС-Журнал успішності-Web"

За посиланням [3] розробник «Політек-Софт» пропонує модуль, який є новою частиною програми "ПС-Студент-Web" і призначений для розширення функціональних можливостей інформаційної системи, що функціонує на основі використання пакету програм "Деканат".

Даний Модуль під назвою "Електронний журнал успішності" дивись рисунок 1.1 (http://www.kneu.kiev.ua/ua/publication/content/619.htm) успішно функціонує в університеті у складі автоматизованої системи управління закладом з лютого 2010 р.

Автори зазначають, що модуль "ПС-Журнал успішності-Web" може виконувати наступні функції:

- забезпечення можливості реєстрації поточної успішності та відвідувань занять студентами в реальному часі;

- організація доступу до таких даних всім студентам університету;
- генерація множини звітних документів та узагальнюючих показників для здійснення всебічного аналізу даних щодо поточної успішності студентів і, відповідно, прийняття управлінських рішень;
- інтегрування даних щодо поточної успішності з метою автоматичного формування у базі даних підсумкових семестрових показників успішності та їх друку у вигляді заліково-екзаменаційних відомостей.

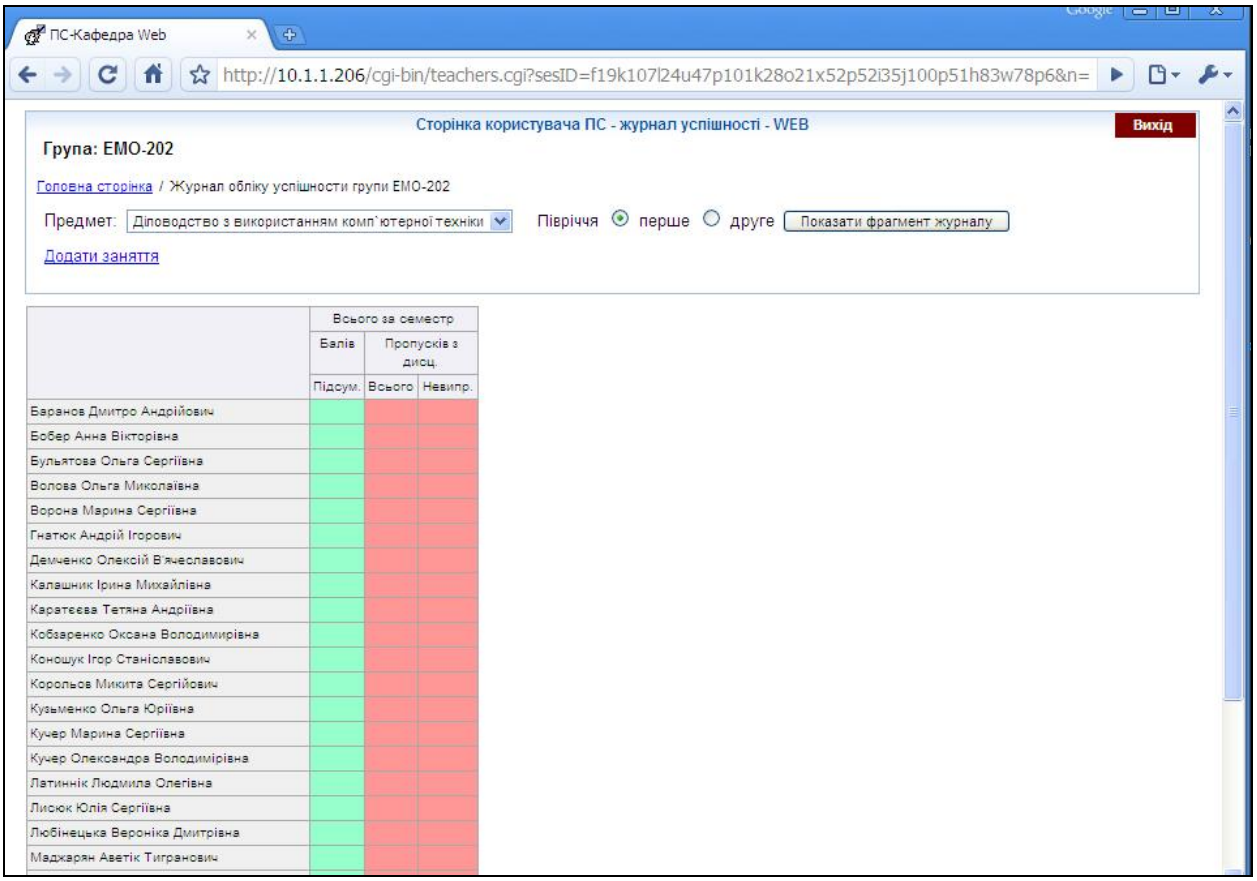

Рисунок 1.1- Початкова (пуста) сторінка журналу і академічної групи з дисципліни

Автори програмного модуля зазначають наступні функції та властивості:

- забезпечення можливість реєстрації викладачів як користувачів Модуля безпосередньо самими викладачами шляхом пошуку себе у списку зареєстрованих викладачів відповідних кафедр та встановлення логіну і паролю. Викладач, що зареєструвався, без спеціального підтвердження не має права доступу до режиму реєстрації даних щодо поточної успішності студентів;
- з метою захисту інформації передбачено збереження у базі даних логінів та паролів викладачів у зашифрованому вигляді;
- реалізується спеціальний режим роботи Модуля, у якому відповідальним особам з кафедр надано доступ до облікових записів викладачів з метою включення або відключення права доступу до режиму реєстрації даних щодо поточної успішності студентів та, у разі необхідності, редагування відповідних логінів та паролів;
- реалізується також спеціальний режим роботи програми "ПС-Студент-Web", у якому секретарям деканатів надано доступ до режиму формування (редагування) даних щодо закріплення занять за викладачами;
- за введеним викладачем логіном та паролем визначається особа викладача, перевіряється його право доступу до режиму реєстрації даних щодо поточної успішності студентів, і викладачу надається перелік академічних груп з зазначенням відповідних курсів, напрямів (спеціальностей), факультетів, у яких він проводить заняття. Після вибору певної групи викладач отримує доступ до режиму реєстрації (редагування) даних щодо поточної успішності студентів обраної групи з тієї дисципліни (дисциплін), яку він викладає;
- в режимі реєстрації (редагування) даних щодо поточної успішності студентів групи викладач може:

- зареєструвати факт проведення заняття з певної дисципліни та певного виду занять у певний вказаний ним день обраного півріччя;
- змінити атрибути заняття (дату та вид заняття);
- видалити зареєстроване заняття як таке, що введене помилково (при цьому всі дані щодо такого заняття зберігаються у базі даних з особливим статусом);
- зареєструвати (змінити) бали, що отримав студент на занятті;
- зареєструвати (змінити) дані щодо відсутності певних студентів на занятті з фіксацією причини відсутності;
- переглядати у студентів групи бали підсумкового контролю за семестр, які формуються Модулем автоматично;
- переглядати у студентів групи загальну кількість пропущених годин та кількість годин, пропущених без поважної причини, на заняттях з даної дисципліни з початку семестру;
- створювати збірні групи з метою побудови власного інтерфейсу. Використання збірних груп суттєво спрощує процес реєстрації даних;

- види можливих занять наперед встановлюються у спеціальній таблиці-довіднику;
- всі зміни, що вносяться у базу даних безпосередньо викладачами та секретарями деканатів, які стосуються даних щодо поточної успішності студентів, автоматично реєструються у спеціальній таблиці. При цьому фіксується ім`я відповідної таблиці, ім`я поля, значення даних перед редагуванням, ідентифікатор користувача, дата зміни. У разі необхідності, може здійснюватись аналіз історії змін;
- секретар деканату після перегляду даних щодо поточної успішності студентів групи, що фіксуються викладачами, може викликати режим автоматичного інтегрування таких даних у бали підсумкового

модульного контролю (ПМК). Після здійснення інтегрування доступ до змін у даних щодо поточної успішності у відповідному семестрі для викладачів блокується. Для секретаря зберігається режим ручного редагування внесеного автоматично балу ПМК;

- Модуль дозволяє демонструвати показники поточної успішності студентам університету. При цьому авторизація студента здійснюється за його прізвищем та номером залікової книжки. Студент може переглядати лише власні показники успішності;
- у разі встановлення Модуля на сервер Замовника у базі даних додатково створюється бібліотека SQL-запитів (далі "Бібліотека") для моніторингу та всебічного оперативного аналізу поточної успішності студентів, що реєструється силами викладачів навчального закладу (журнали успішності), а також для контролю ходу процесу ведення журналів. Бібліотека зберігається у базі даних, доступна для використання "активним" користувачам програми "ПС-Студент-Web". Більшість запитів Бібліотеки є параметризованими та здійснюються через спеціальний модуль, що входить до складу програми – "Деканат-SQL". Результати виконання запиту можуть автоматично завантажуватись у Internet Explorer, MS Excel або у файли формату *.dbf;
- забезпечується автоматична генерація відповідної множини звітних документів, що віддзеркалюють узагальнюючі показники поточної успішності студентів на рівнях Заклад, Факультет, Курс, Група, Студент;
- дані щодо поточної успішності студентів протягом семестру доступні користувачам інформаційної системи для перегляду та аналізу шляхом фільтрації в режимі "Відібрати дані";
- в режимі "Відібрати дані" користувачі інформаційної системи отримують можливість створення необмеженої кількості шаблонів табличних звітів з встановленням переліку полів, їх слідування, способу сортування, шрифтового та текстового оформлення. Такі шаблони зберігаються у базі

даних та дозволяють генерувати прості табличні звіти про результати відбору даних щодо поточної успішності за довільним множинним критерієм, які задає користувач;

- реєстрація поточної успішності студентів у Замовника здійснюється з урахуванням положень Порядку оцінювання знань студентів Замовника;
- передбачається збереження даних щодо поточної успішності студентів, що реєструється у базі даних протягом навчального року. Після закінчення навчального року ці дані після здійснення їх інтеграції у семестрові показники успішності видаляються;
- Модуль функціонує на основі використання Web-інтерфейсу та Internet-технологій (є cgi-сценарієм), що виключає необхідність проведення спеціальних робіт по його встановленню на комп'ютери користувачів вищого навчального закладу;
- Модуль розрахований на роботу (на сервері) в операційній системі Windows Server 2000/2003/2008 з використанням Web-сервера та сервера баз даних Firebird;
- Модуль може використовувати https-протокол при умові використання Замовником Web-сервера, що такий протокол підтримує, та наявності сертифікату сервера SSL. Використання даного протоколу дозволяє передавати дані між користувачами та сервером у зашифрованому вигляді.

1.3. Специфікація вимог до системи «Обліку успішності в академічній групі»

Метою створення специфікації вимог до системи є визначення функцій системи та її не функціональних вимог. Першим кроком створення специфікації є опис глосарію проекту, який відображений у таблиці 1.1.

Таблиця 1.1

Глосарій проекту

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
Журнал обліку успішності	Документ в який записують інформацію про слухачів, вид, дату проведення заняття та присутність студентів. Журнал також є основою моніторингу в умовах кредитно-модульної системи
Викладач	Користувач системи, який має права визначати характеристики заняття виставляти оцінки, отримувати звіти
Студент (слухач)	Особа, що відвідує заняття та отримує оцінку за демонстрацію знань.
Академічна група	Перелік слухачів, які відвідують певні заняття та отримують оцінки
Режим реєстрації	Режим роботи системи під час якого викладач створюю доні що до проведеного заняття (вид, тема, час, наявність слухачів та їх оцінки)
Бали оцінювання	Чисельний або інший символьний еквівалент за допомогою якого викладач оцінює знання студентів
Вид заняття	Визначається типом взаємодії викладача та студентів, може мати значення (лекція, практичне заняття, семінар, лабораторна робота)
Показники поточної успішності	Інтегрований показник балів по відношенню до максимального його значенн
Звітний документ	Документ отриманий з даних занесених в

	систему оцінювання знань
Порядок оцінювання знань	Певні критерії та алгоритм виначення оцінки знань студентів.
2. Користувачі системи	
Викладач	
Студент	
3. Вхідні та вихідні документи	
Перелік академічних груп	Перелік академічних груп, які будуть занесені в систему
Список академічної групи	Прізвища студентів записані в базу даних і прив’язаних до академічних груп
Список дисциплін	Перелік дисциплін, які викладаються в відповідних академічних групах
Журнал відвідування	Документ де занесена інформація присутності студента та тому, або іншому занятті
Відомість оцінок	Документ, що пов’язує список академічної групи з оцінками за певною дисципліною
Витяг успішності студента	Перелік оцінок з в вибраних дисциплін для певного студента

Опрацювавши джерела, що наведені вище та опитавши потенціальних користувачів системи приймаємо наступну специфіка вимог до системи обліку успішності в академічній групі. Вимоги будемо специфікувати за допомогою діаграми варіантів використання.

Користувачами системи будуть: викладач, студент.

На рисунку 1.2 зображено загальну діаграму варіантів використання проектованого програмного забезпечення з точки зору користувача в ролі «Викладач».

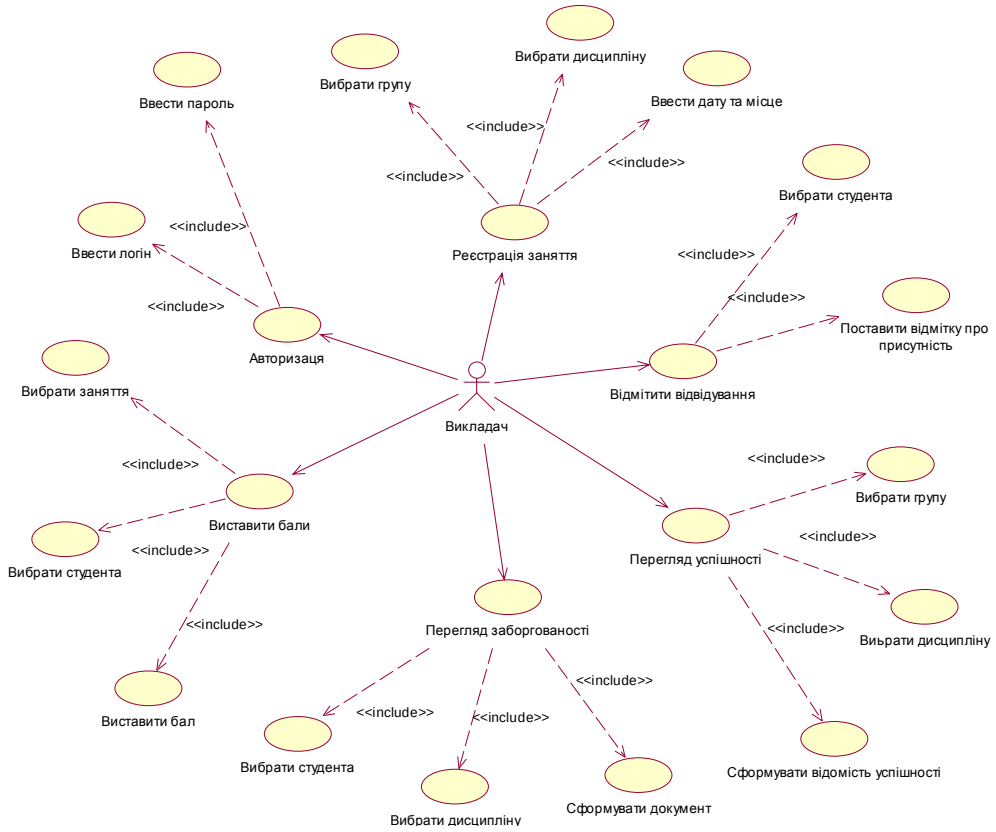

Рисунок 1.2- Загальна діаграма варіантів використання для користувача в ролі «Викладач»

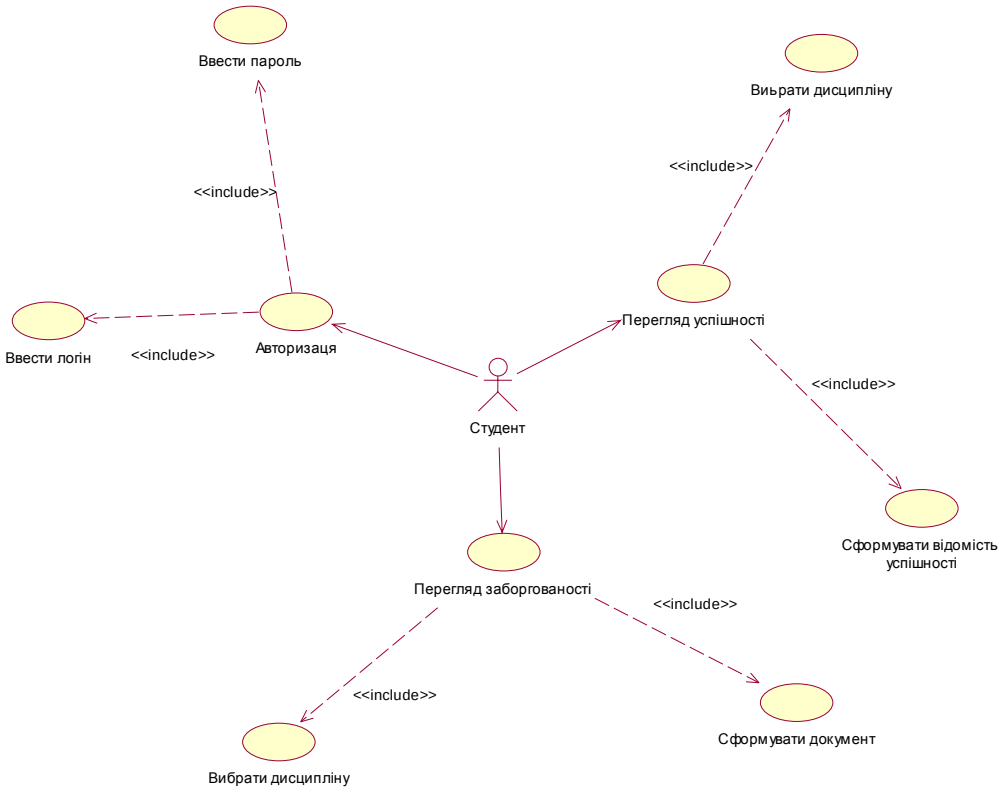

Рисунок 1.3 - Загальна діаграма варіантів використання для користувача в ролі «Студент»

Для користувача «Студент» функції є значно обмежені і від має можливості тільки переглядати свої заборгованості та формувати відомість про свої заборгованості за певними дисциплінами дивись рисунок 1.3.

Далі розроблено табличний опис варіантів використання, які реалізують основну функцію.

Таблиця 1.2

Варіант використання «Авторизація»

Контекст використання	Вхід в систему
Дійові особи	Користувач системи
Передумови	Початок роботи з системою
Триггер	Запуск програми на виконання
Сценарій	Ввести логін Ввести пароль Натиснути кнопку «Ok»
Постумова	Отримання доступу до елементів системи Стартове вікно «Панель керування»

Таблиця 1.3

Варіант використання «Реєстрація заняття»

Контекст використання	Проведення заняття
Дійові особи	Користувач системи «Викладач»
Передумови	Вибрати пункт меню «Реєстрація заняття»
Триггер	Реєстрація заняття
Сценарій	Вибрати академічну групу Вибрати дисципліну Ввести дату та місце проведення
Постумова	Відмітити присутність студентів

Таблиця 1.4

Варіант використання «Відмітити відвідування»

Контекст використання	Внести інформацію про присутність студентів на конкретному занятті
Дійові особи	Користувач системи «Викладач»
Передумови	Виконати реєстрацію заняття
Триггер	Реєстрація заняття та встановлення присутніх на занятті
Сценарій	Отримати перелік студентів академічної групи Поставити відмітки про присутність (відсутність) Зберегти дині про перекличку
Постумова	Система залишається в тому самому стані для внесення інших відміток (оцінка, коментар)

Таблиця 1.5

Варіант використання «Виставити бали»

Контекст використання	Необхідність записати оцінку
Дійові особи	Користувач системи «Викладач»
Передумови	Виконати варіант використання «Реєстрація заняття»
Триггер	Реєстрація заняття та встановлення присутніх на занятті
Сценарій	Вибрати студента Записат в комірці на проти студента та під відповідним заняттям оцінку
Постумова	Система залишається в тому самому стані для внесення інших відміток (оцінка, коментар)

Таблиця 1.6

Варіант використання «Перегляд успішності»

Контекст використання	Отримати інформацію про успішність
Дійові особи	Користувач системи «Викладач»
Передумови	Заповнені дані про академічні групи та виставлені оцінки
Триггер	Необхідність отримати інформацію про успішність студентів
Сценарій	Вибрати групу студентів Вибрати дисципліну Сформувати відомість успішності
Постумова	Повернення у вікно «Панель керування»

Таблиця 1.7

Варіант використання «Перегляд заборгованості»

Контекст використання	Отримати інформацію про заборгованість
Дійові особи	Користувач системи «Викладач»
Передумови	Заповнені дані про академічні групи та відмітки про присутність
Триггер	Необхідність отримати інформацію про заборгованість студентів
Сценарій	Вибрати студента Вибрати дисципліну Сформувати відомість заборгованості
Постумова	Повернення у вікно «Панель керування»

Далі виконаємо розкадровку варіантів використання, представлення варіанті використання по кадрах (екранних формах), що використовуються для реалізації функцій.

Розкадровка варіанту використання «Авторизація» показана на рисунку 1.4.

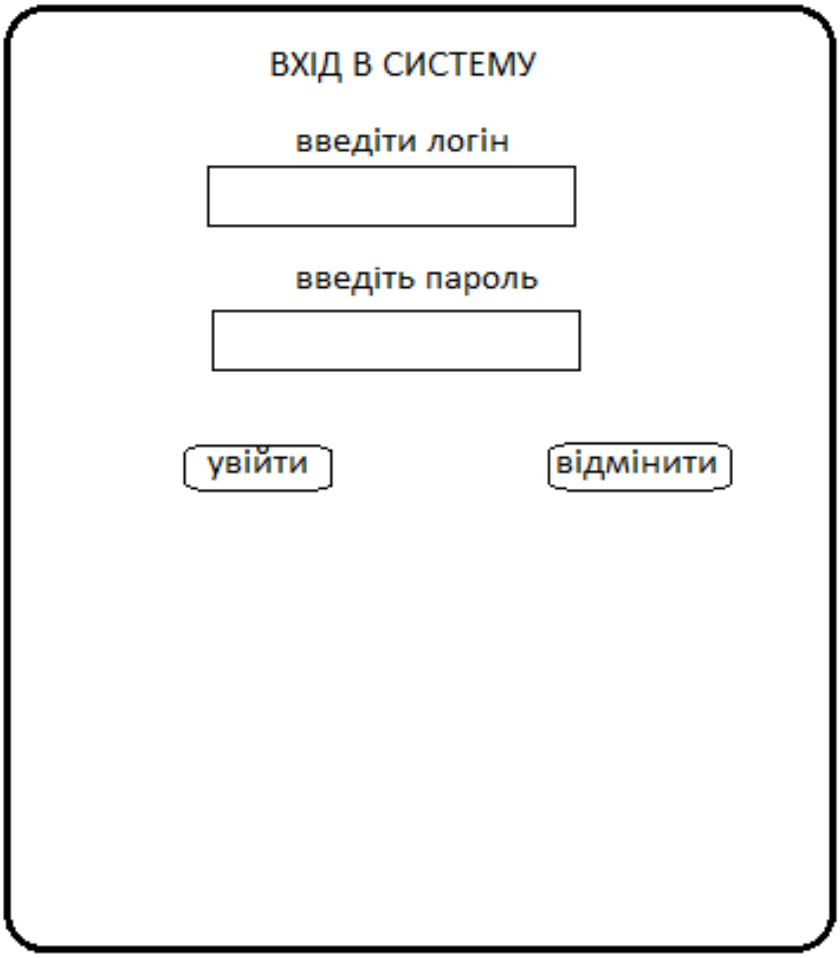

Рисунок 1.4- Розкадровка варіанту використання «Авторизація»

Розкадровка варіанту використання «Реєстрація заняття» показана на рисунку 1.5.

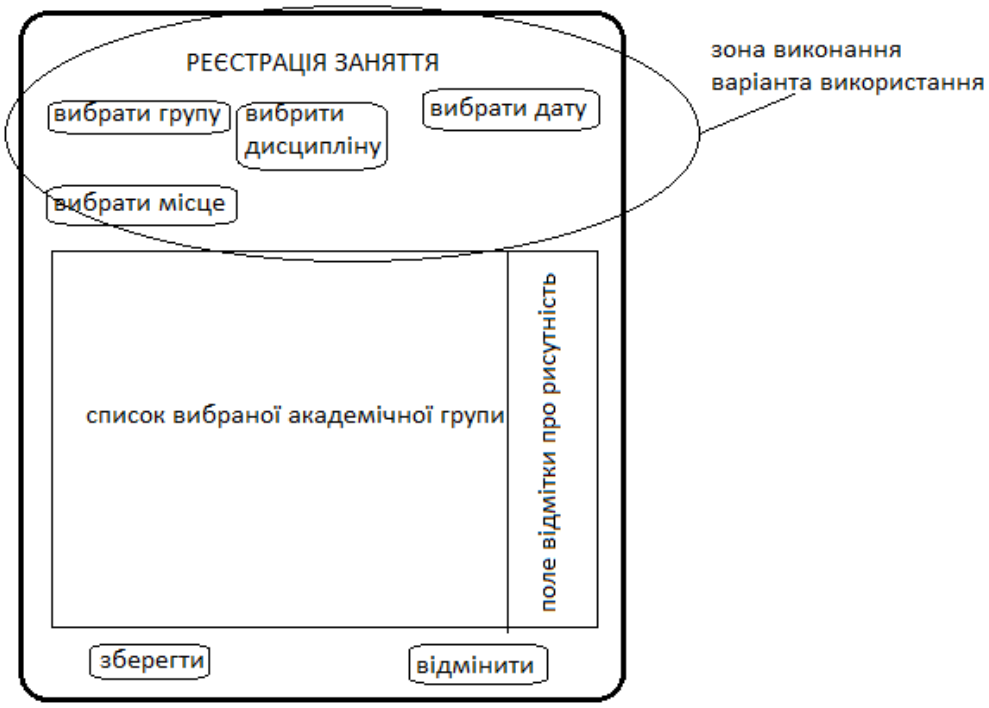

Рисунок 1.5.- Розкадровка варіанту використання «Реєстрація заняття»

Розкадровка варіанту використання «Відмітити відвідування» показана на рисунку 1.6.

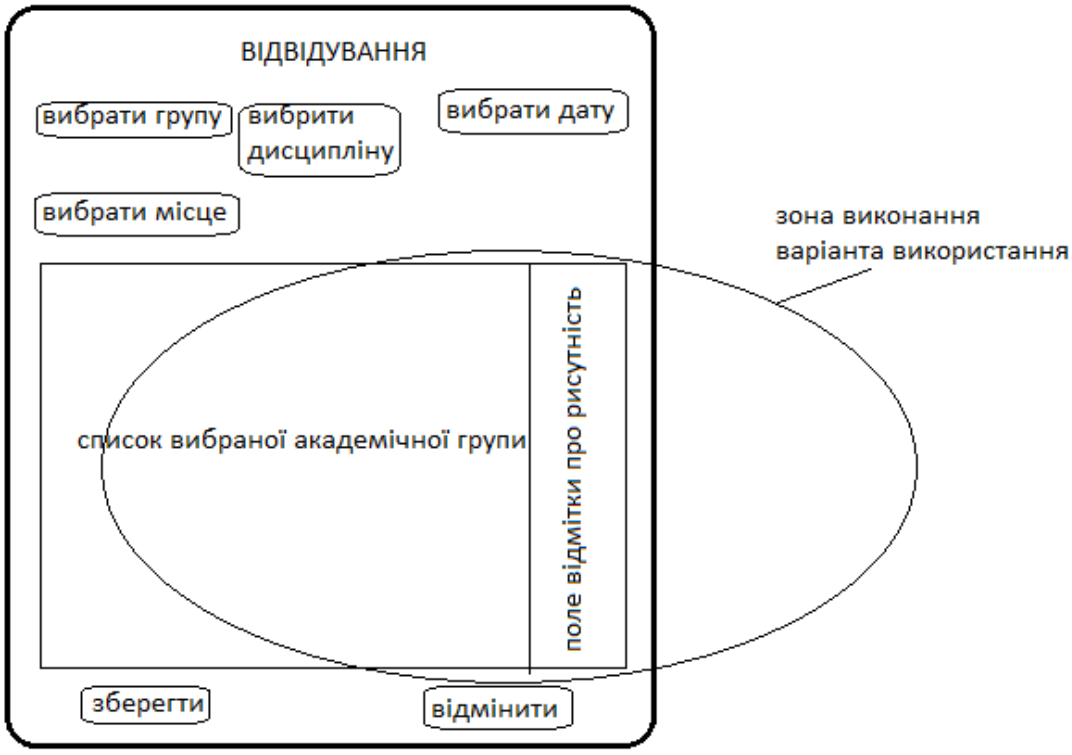

Рисунок 1.6- Розкадровка варіанту використання «Відмітити відвідування»

Розкадровка варіанту використання ««Виставити бали»» показана на рисунку 1.7.

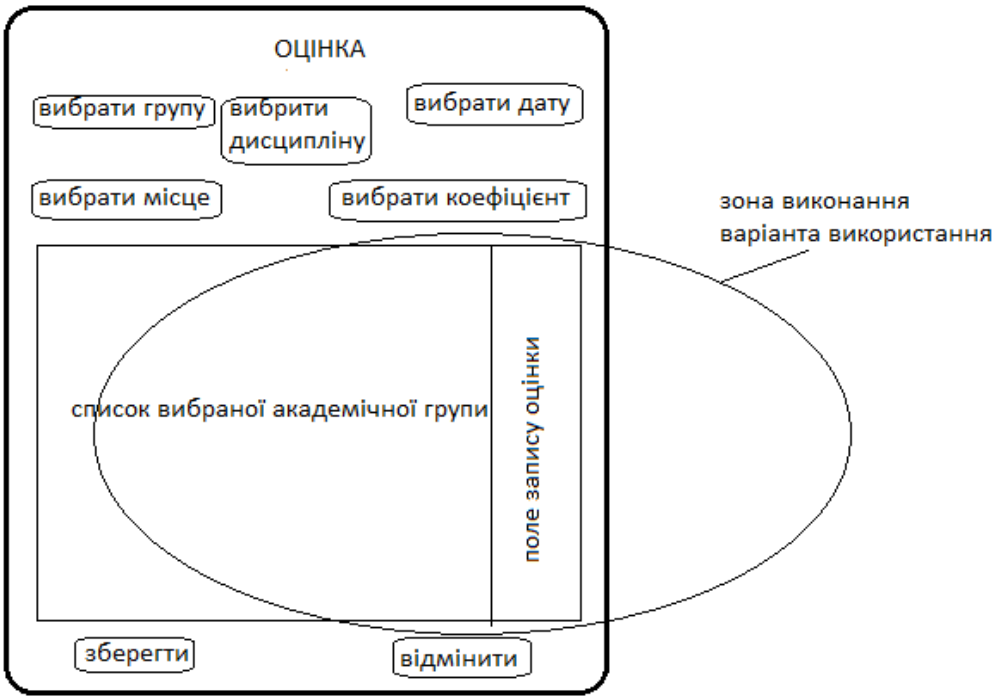

Рисунок 1.7.- Розкадровка варіанту використання «Виставити бали»

Розкадровка варіанту використання «Перегляд успішності» показана на рисунку 1.8.

Рисунок 1.8- Розкадровка варіанту використання «Перегляд успішності»

Розкадровка варіанту використання «Перегляд заборгованості» показана на рисунку 1.9.

Рисунок 1.9- Розкадровка варіанту використання «Перегляд заборгованості»

В результаті опису отримуємо специфікацію функціональних та нефункціональних вимог відповідно таблиця 1.8 та таблиця 1.9.

Таблиця 1.8

Специфікацію функціональних вимог

Іденти-фікатор вимог	Назва вимоги	Атрибути вимоги		
		Пріоритет	Складність	Контакт
1	Авторизація	Обов’язкова	Середня	
2	Реєстрація заняття	Обов’язкова	Середня	
3	Відмітити відвідування	Обов’язкова	Висока	
4	Виставити бали	Обов’язкова	Середня	

5	Перегляд успішності	Обов’язкова	Середня	
6	Перегляд заборгованості	Рекомендована	Середня	

Таблиця 1.9

Специфікацію нефункціональних вимог

Іденти-фікатор вимог	Назва вимоги	Атрибути вимоги		
		Пріоритет	Складність	Контакт
Застосовність				
1	Авторизація	Обов’язкова	5	
2	Реєстрація заняття	Обов’язкова	20	
3	Відмітити відвідування	Обов’язкова	30	
4	Виставити бали	Обов’язкова	15	
5	Перегляд успішності	Обов’яз кова	15	
6	Перегляд заборгованості	Рекомендована	20	
Надійність				
1	Авторизація	Обов’язкова	99	
2	Реєстрація заняття	Обов’язкова	98	
3	Відмітити відвідування	Обов’язкова	98	
4	Виставити бали	Обов’язкова	98	
5	Перегляд успішності	Обов’язкова	98	
6	Перегляд заборгованості	Рекомендована	95	
Експлуатаційна придатність				
1	Авторизація	Обов’язкова	100	

2	Реєстрація заняття	Обов’язкова	100	
3	Відмітити відвідування	Обов’язкова	100	
4	Виставити бали	Обов’язкова	100	
5	Перегляд успішності	Обов’язкова	100	
6	Перегляд заборгованості	Рекомендована	100	

Висновки до першого розділу

Мобільна програма для обліку успішності в академічній групі має гнучку систему налаштувань, що дозволяє викладачу налаштувати інтерфейс програми. Мобільна програма для обліку успішності в академічній групі дає можливість працювати мобільно, а також публікації в мережі Internet, що надасть можливість обліку оцінок викладачем також з будь-якого комп’ютера підключеного до мережі Internet, та можливість перегляду студентами кожної групи своїх оцінок.

РОЗДІЛ ІІ
ПРОЕКТУВАННЯ СИСТЕМИ «МОБІЛЬНА ПРОГРАМА ДЛЯ ОБЛІКУ УСПІШНОСТІ В АКАДЕМІЧНІЙ ГРУПІ»

2.1. Розроблення архітектури програмної системи

Архітектура - це набір значущих рішень з приводу організації системи програмного забезпечення, набір структурних елементів і їх інтерфейсів, за допомогою яких компонується система, разом з їх поведінкою , обумовленим у взаємодії між цими елементами, компоновка елементів в поступово укрупнюються підсистеми, а також стиль архітектури який направляє цю організацію - елементи та їх інтерфейси, взаємодії і компоновку [4].

Архітектура програми або комп'ютерної системи - це структура або структури системи, які включають елементи програми, видимі ззовні властивості цих елементів і зв'язку між ними [5].

Архітектура - це структура організації та пов'язане з нею поводження системи. Архітектуру можна рекурсивно розібрати на частини, які взаємодіють за допомогою інтерфейсів, зв'язку, які з'єднують частини, і умови складання частин. Частини, які взаємодіють через інтерфейси, включають класи, компоненти і підсистеми [6].

Архітектура програмного забезпечення системи або набору систем складається з усіх важливих проектних рішень з приводу структур програми і взаємодій між цими структурами, які складають системи. Проектні рішення забезпечують бажаний набір властивостей, які повинна підтримувати система, щоб бути успішною. Проектні рішення надають концептуальну основу для розробки системи, її підтримки та обслуговування. [7]

Хоча визначення дещо відрізняються, ми можемо бачити чималу ступінь подібності. Наприклад, більшість визначень вказують на те, що

архітектура пов'язана зі структурою і поведінкою, а також тільки зі значущими рішеннями, може відповідати деякому архітектурному стилю, на неї впливають зацікавлені в ній особи і її оточення, вона втілює рішення на основі логічного обґрунтування.

2.1.1. Структура програмного забезпечення

Поняття архітектура в англійській мові часто пов'язується з будівництвом будівель або деяких інших інженерних споруд (engineering structure), наприклад, мостів. Хоча існують і інші характеристики цих елементів, такі як поведінка, відповідність мети і навіть естетика, але саме термін "структура (structure)" найбільш відомий і найчастіше згадується.

Під архітектурою часто розуміють схему, на якій будуть зображені структурні аспекти системи - будь то архітектурні рівні, компоненти або розподілені вузли. Дійсно, структура є найважливішою характеристикою архітектури. Структурні аспекти архітектури проявляються багатьма способами, і в результаті більшість визначень архітектури свідомо залишають невизначеними. Структурний елемент може бути підсистемою, процесом, бібліотекою, базою даних, обчислювальним вузлом, системою в традиційному сенсі, готовим продуктом і так далі.

Багато визначення архітектури визнають також не тільки самі структурні елементи, але і композиції із структурних елементів, їх зв'язку (і будь-які сполучні ланки, необхідні для підтримки цих відносин), інтерфейси і розбиття. І знову, кожен з цих елементів може бути представлений різними способами. Наприклад, сполучна ланка може являти собою сокет, бути синхронним або асинхронним, бути пов'язаним з конкретним протоколом і так далі.

На рисунку 2.1 показано діаграма модулів UML, що містить структурні елементи, які представляють систему «Мобільна програма для обліку успішності в академічній групі».

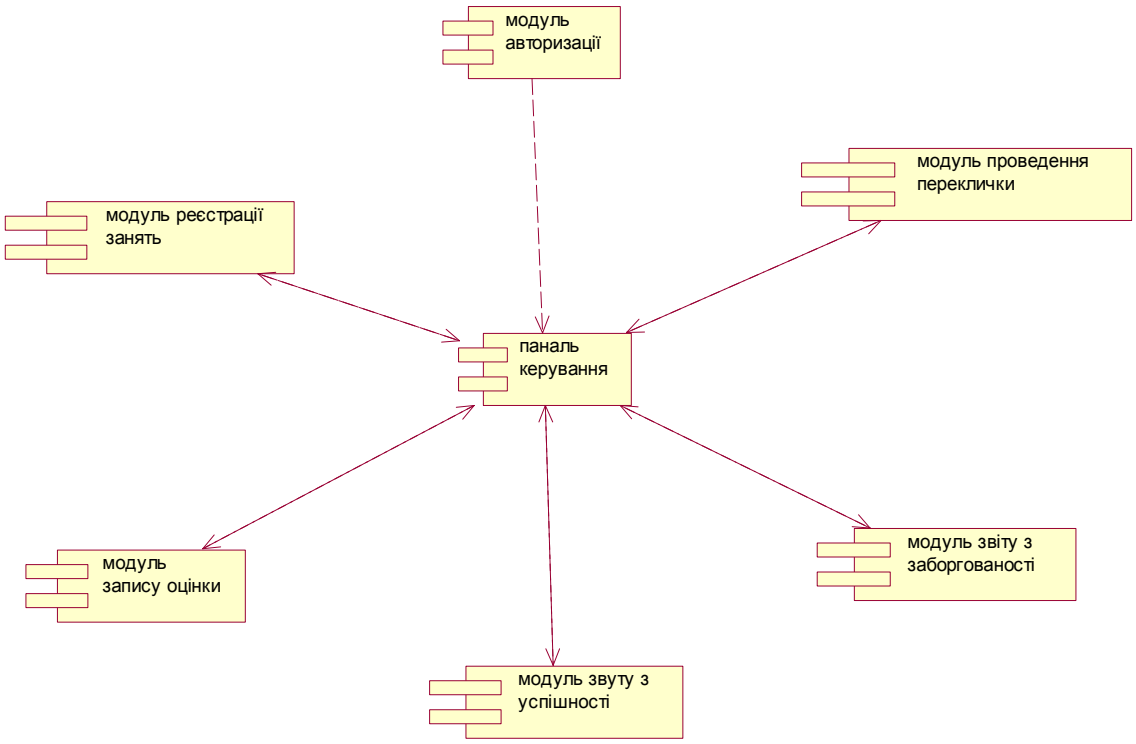

Рисунок-2.1. Архітектура програмної системи для обліку успішності в академічній групі

Виходячи з функції системи, визначених у попередньому розділі, встановлюємо наступні функціональні блоки системи (у вигляді окремих класів), які співпадають з модулями системи:

Авторизація- «Authorization».

Реєстрація занять - «Registration_sessions ».

Проведення переклички - «Holding_roll».

Запис оцінки- «Writing_assessment».

Звіт успішності - «Report_progress».

Звіт заборгованості - «Debt_Report».

Панель керування – «Control_Panel»

Деталізуємо обрані варіантів використання і модулі системи з використанням діаграми класів системи рисунок 2.2.

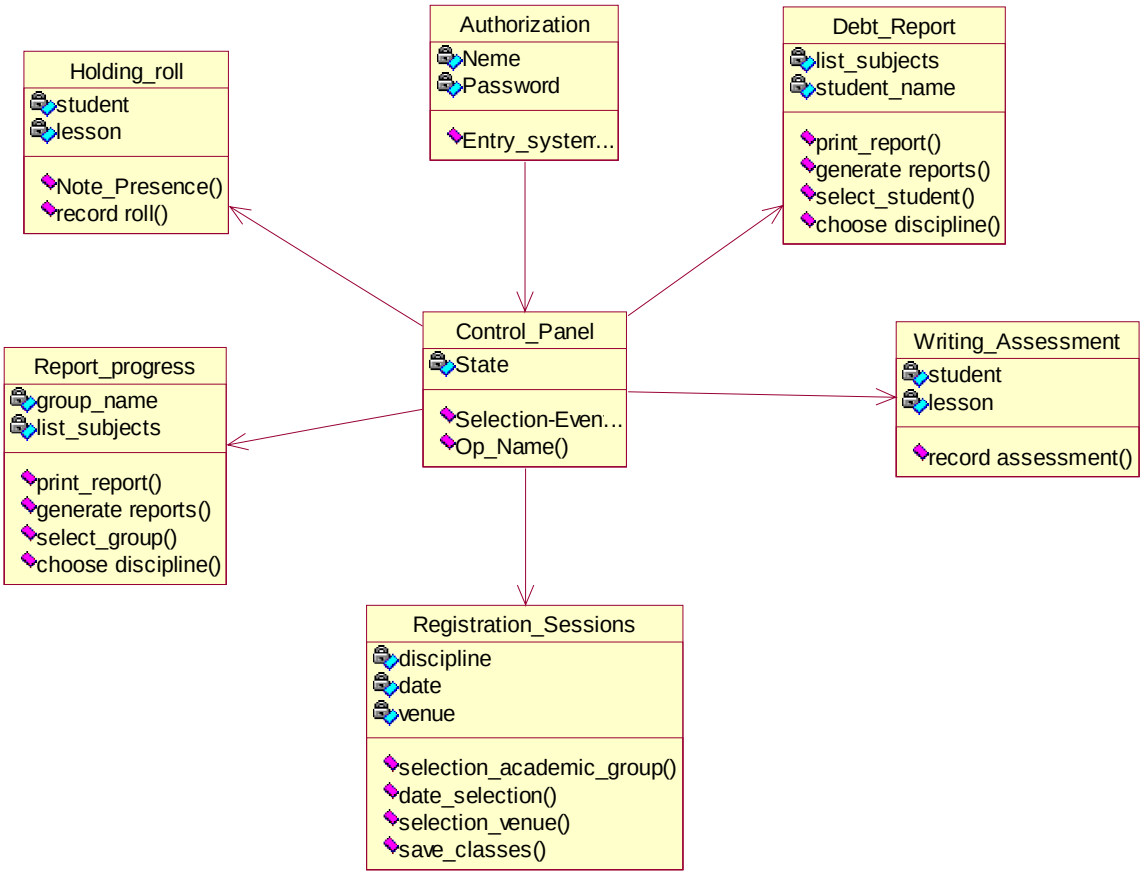

Рисунок-2.2 Діаграма класів програмної системи для обліку успішності в академічній групі

2.2. Проектування структури бази даних

Природно, що проект бази даних треба починати з аналізу предметної області та виявлення вимог до неї окремих користувачів (співробітників організації, для яких створюється база даних) в нашому випадку це викладач та студент.

Об'єднуючи окремі уявлення про вміст бази даних, отримані в результаті опитування користувачів, і свої уявлення про дані, які можуть

знадобитися в майбутніх додатках, спочатку створюємо узагальнене неформальний опис створюваної бази даних. Це опис, виконуємо з використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих усім людям, які працюють над проектуванням бази даних, називають інфологічною моделлю даних.

Всі архітектури представляються нотацією UML і при потребі, доповнюються текстовими описами.

В ході проектування архітектури БД, було обрано для її відтворення реляційну модель даних та виділено наступні відношення, що відображені на наступних рисунках рисунки 2.3-2.9.

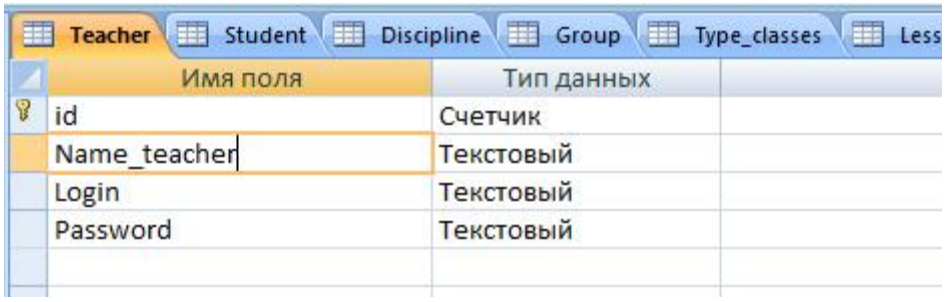

Рисунок 2.3- Таблиця бази даних «Teacher»

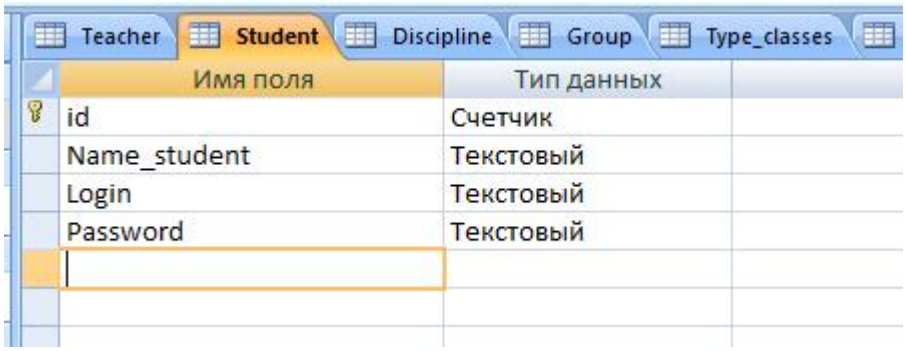

Рисунок 2.4- Таблиця бази даних «Student»

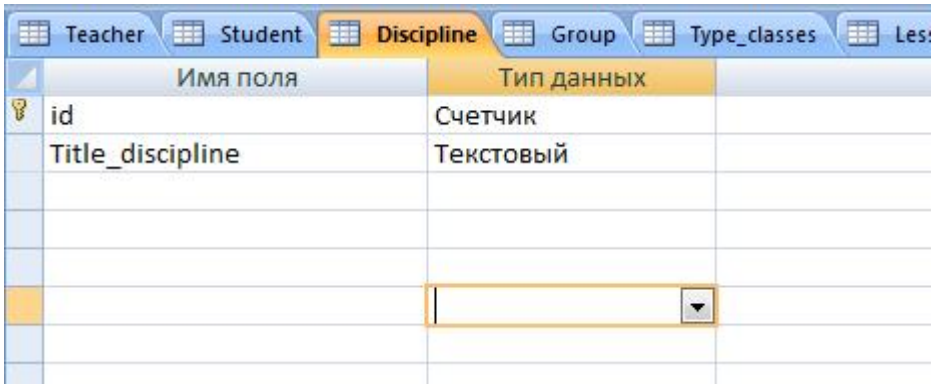

Рисунок 2.5- Таблиця бази даних «Discipline»

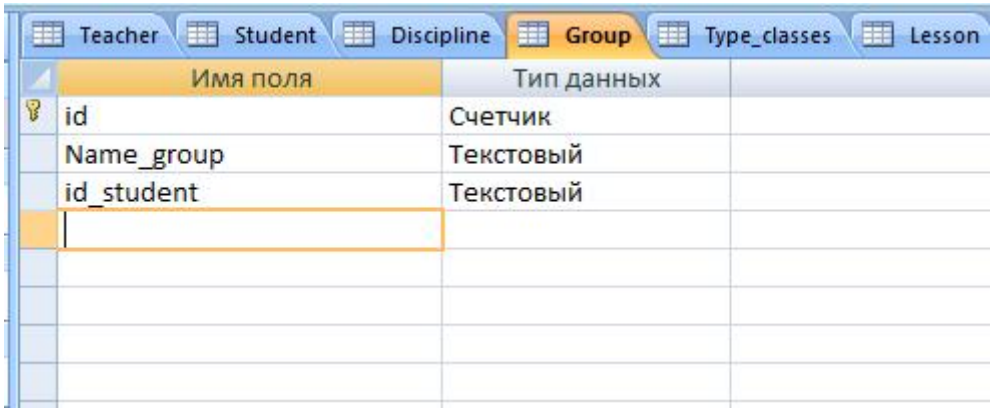

Рисунок 2.6- Таблиця бази даних «Group»

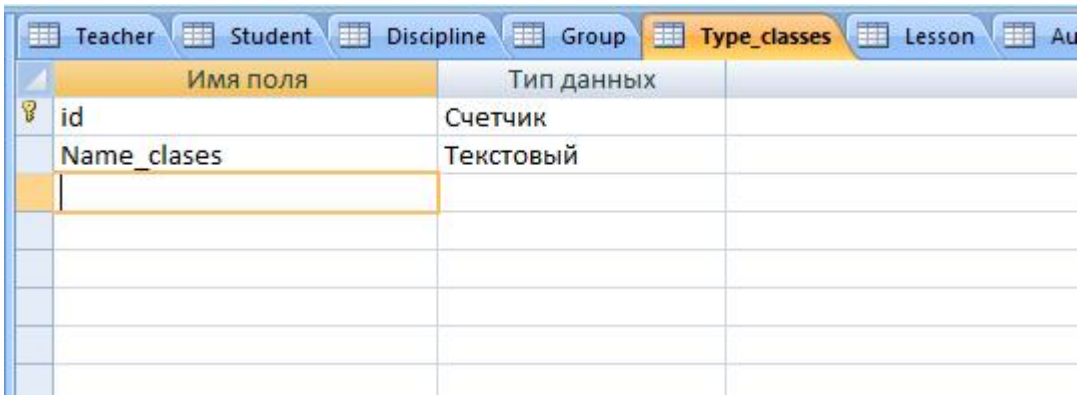

Рисунок 2.7- Таблиця бази даних «Type_classes»

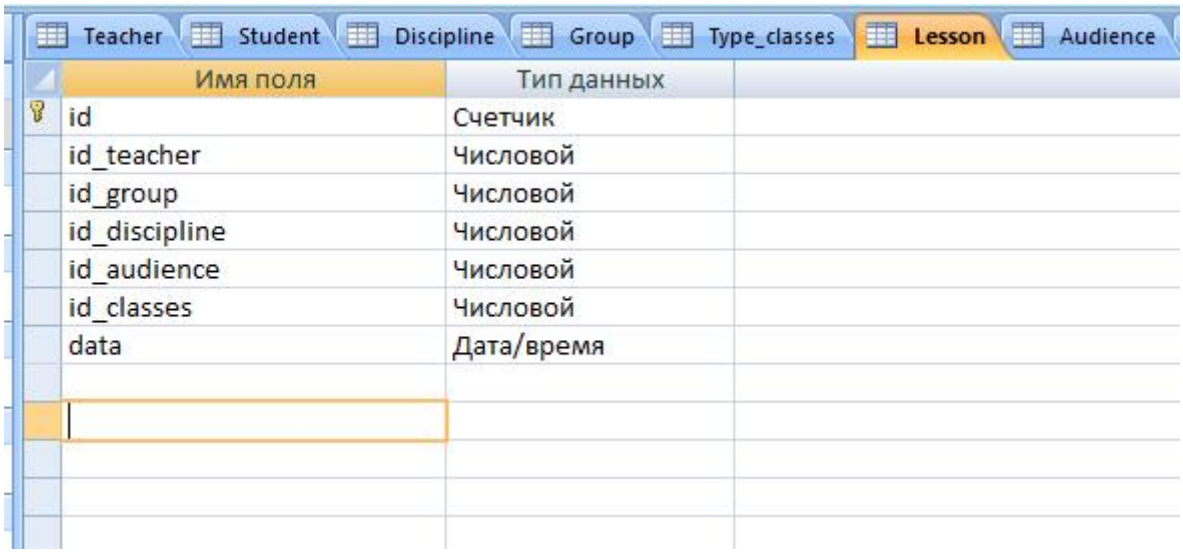

Рисунок 2.8- Таблиця бази даних «Lesson»

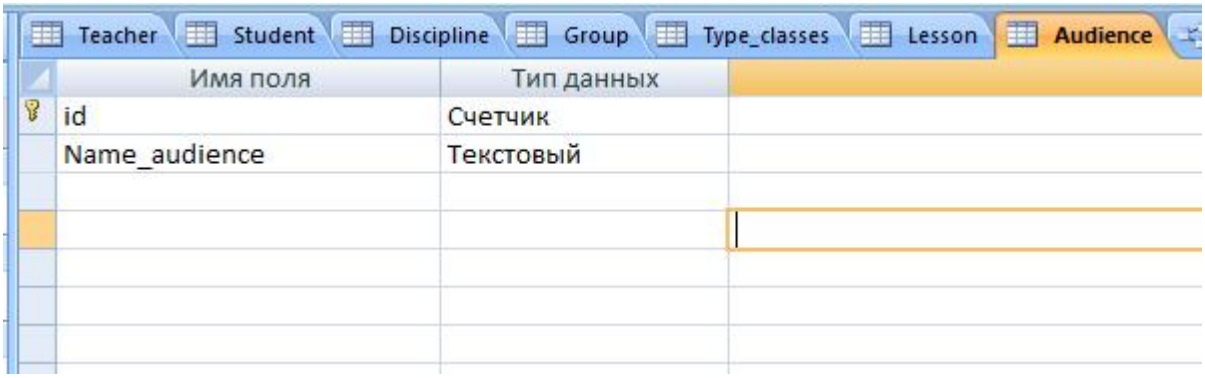

Рисунок 2.9- Таблиця бази даних «Audience»

Діаграма реляційної моделі даних зображена на рисунку 2.10.

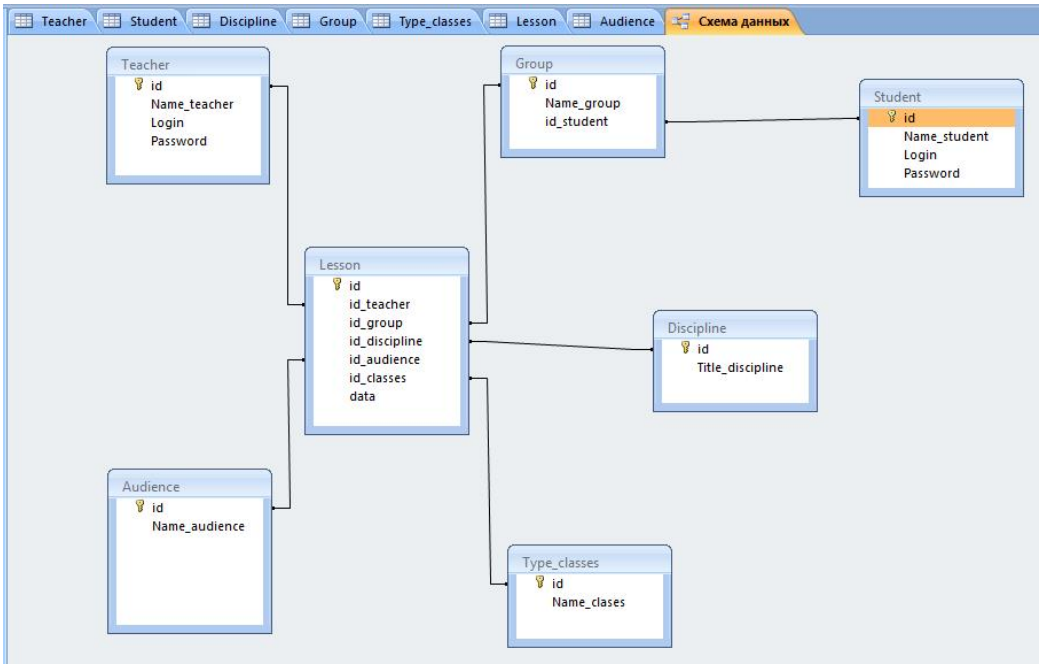

Рисунок 2.10- ER діаграма бази даних

Висновки до другого розділу

В другому розділі запроектовано мобільну програму для обліку успішності в академічній групі, також запроектовано до неї база даних. Проектування здійснювалося з використанням об’єктно-орієнтованого підходу до проектування, а також використовувалася UML діаграми для нотацій.

РОЗДІЛ III
РОЗДІЛ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОБІЛЬНОЇ СИСТЕМИ «МОБІЛЬНА ПРОГРАМА ДЛЯ ОБЛІКУ УСПІШНОСТІ В АКАДЕМІЧНІЙ ГРУПІ»

3.1. Програмна реалізація проекту

3.1.1. Вибір засобу створення системи

Найдоступнішим мобільним пристроєм є пристрій, що базується на операційній системі «Android», тому засобом створення програмного забезпечення буде Android Studio.

Android Studio - це інтегроване середовище розробки (Integrated Development Environment IDE) для роботи з платформою Android , анонсована 16 травня 2013 року на конференції Google I / O .

IDE перебувала у вільному доступі починаючи з версії 0.1, опублікованій в травні 2013, а потім перейшла в стадію бета-тестування, починаючи з версії 0.8, яка була випущена в червні 2014 року. Перша стабільна версія 1.0 була випущена в грудні 2014 року, тоді ж припинилася підтримка плагіна Android Development Tools (ADT) для Eclipse .

Android Studio, заснована на програмному забезпеченні IntelliJ IDEA від компанії JetBrains , офіційне засіб розробки Android додатків. Дане середовище розробки доступна для Windows , OS X і Linux.

Нові функції з'являються з кожною новою версією Android Studio. На даний момент доступні наступні функції:

- Розширений редактор макетів: WYSIWYG , здатність працювати з UI компонентами за допомогою Drag-and-Drop , функція попереднього макета на декількох конфігураціях екрану.
- Збірка програм, заснована на Gradle .
- Різні види збірок і генерація кількох .apk файлів

- рефакторинг коду
- Статичний аналізатор коду (Lint), що дозволяє знаходити проблеми продуктивності, несумісності версій і інше.
- Вбудований ProGuard і утиліта для підписки додатків.
- Шаблони основних макетів і компонентів Android.
- Підтримка розробки додатків для Android Wear і Android TV [6] .
- Вбудована підтримка Google Cloud Platform, яка включає в себе інтеграцію з сервісами Google Cloud Messaging і App Engine.
- Android Studio 2.1 підтримує Android N Preview SDK, а це означає, що розробники зможуть почати роботу зі створення програми для нової програмної платформи.
- Нова версія Android Studio 2.1 здатна працювати з оновленим компілятором Jack, а також отримала покращену підтримку Java 8 і вдосконалену функцію Instant Run [7] .

Системні вимоги:

Windows- версія OS Microsoft Windows 10/8/7 / Vista / 2003 (32 або 64-bit).

OS X- Mac® OS X® 10.8.5 або вище, до 10.9 (Mavericks).

Linux- GNOME або KDE.

Оперативна пам'ять- 2 ГБ (мінімум), 4 ГБ (рекомендується).

Вільне місце на диску- 400 МБ.

Вільне місце для Android SDK- 1 ГБ (мінімум).

Версія JDK- Java Development Kit 7

Роздільна здатність екрану - 1280 x 800 (мінімум).

На рисунку 3.1 наведено зовнішній вигляд середовища розробки Android Studio.

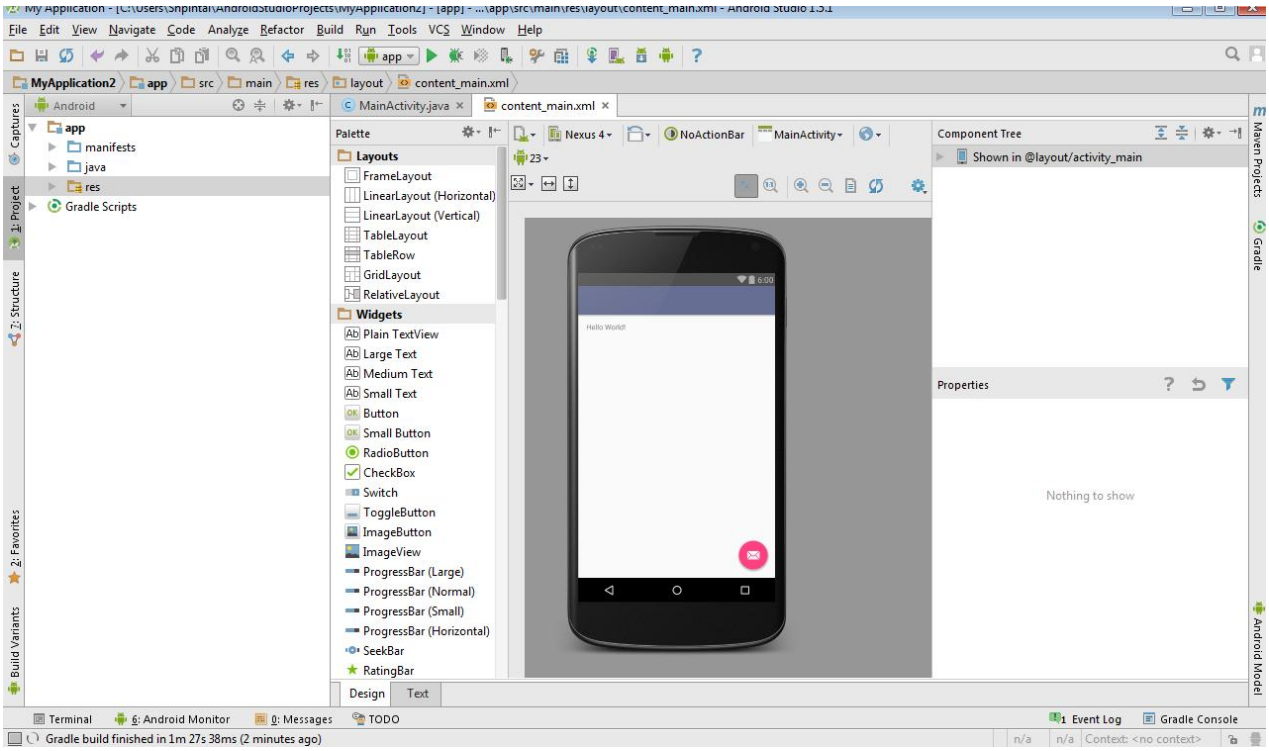

Рисунок 3.1- Зовнішній вигляд середовища розробки Android Studio

Для створення нового проекту необхідно виконати наступні дії:

Запускаємо Android Studio і вибираємо File | New | New Project. З'явиться діалогове вікно майстра .

Поле Application name: - зрозуміле ім'я для програми, яка буде відображатися в заголовку програми. За замовчуванням вже є My Application .

Поле Company Domain: служить для вказівки вашого сайту. За замовчуванням там може з'явитися ваше ім'я як користувача комп'ютера. Якщо сайт у вас є, то можете ввести його адресу, або придумайте яку-небудь назву. Введене ім'я запам'ятовується і буде автоматично підставлятися в наступних нових проектах.

Поле Package name: формує спеціальний Java-пакет на основі вашого імені з попереднього поля. У Java використовується перевернутий варіант для найменування пакетів, тому спочатку йде домен, а потім вже назва сайту. Пакет служить для унікальної ідентифікації вашої програми, коли ви

будете його поширювати. Кнопка Edit дозволяє відредагувати підготовлений варіант.

Третє поле Project location: дозволяє вибрати місце на диску для створюваного проекту рисунок 3.2.

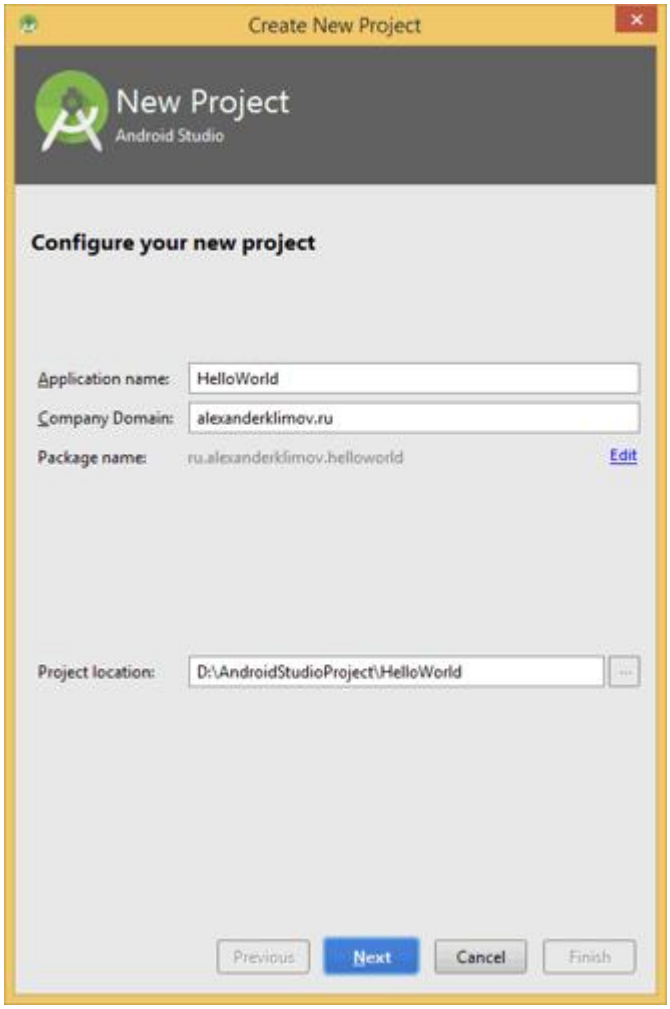

Рисунок 3.2- Діалогове вікно майстра розробки Android Studio

Натискаємо на кнопку Next і переходимо до наступного вікна рисунок 3.3. Тут ми вибираємо типи пристроїв, під які будемо розробляти свій додаток. Вибираємо для смартфонів і планшетів, тобто залишаємо прапорець у першого пункту.

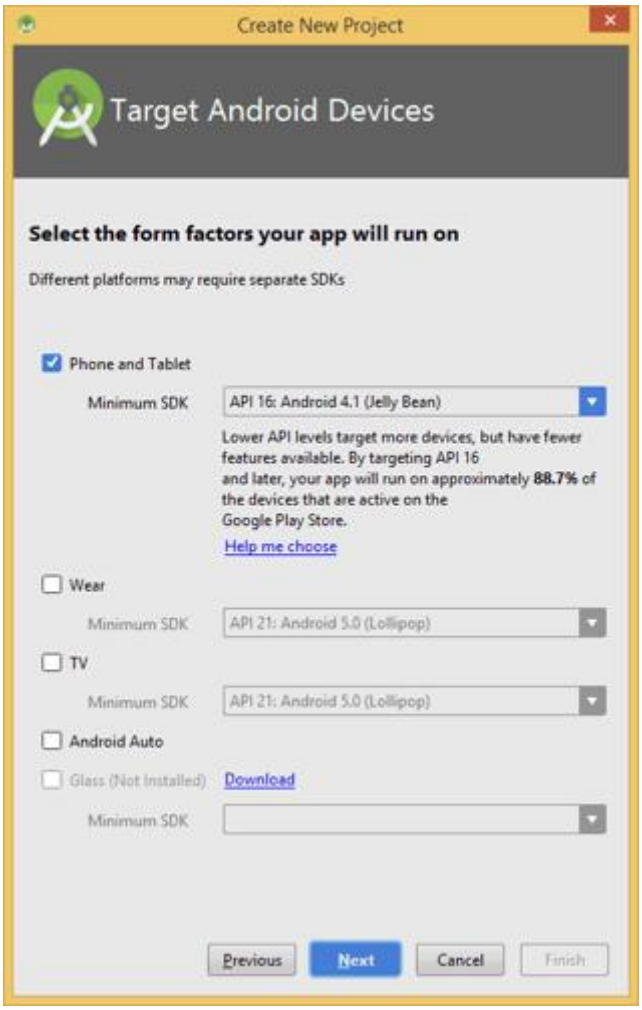

Рисунок 3.3- Діалогове вікно вибору типу пристроїв Android Studio

Крім вибору типу пристроїв, вибраємо мінімальну версію системи, під якою буде працювати додаток. Вибераємо свій варіант. На даний момент Гугл підтримує версії, починаючи з API 7, випускаючи спеціальні бібліотеки сумісності для старих пристроїв.

Далі знову натискаємо кнопку Next . У вікні, що появилося рисунок 3.4 слід вибрати зовнішній вигляд екрану програми.

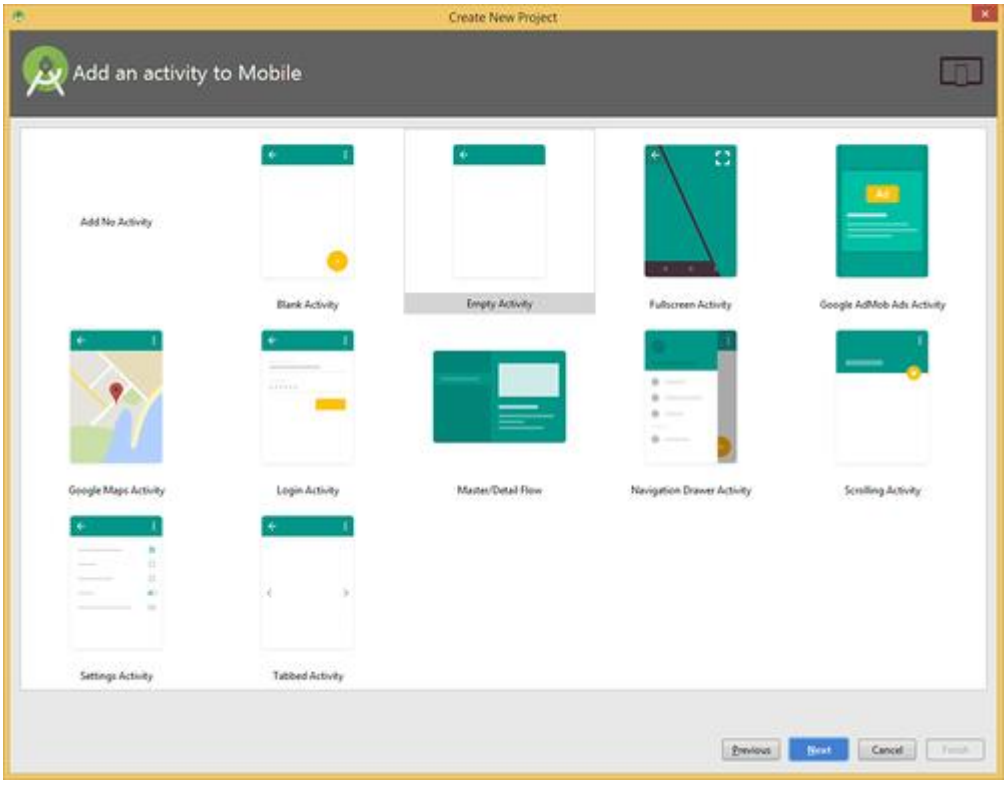

Рисунок 3.4- Діалогове вікно вибору зовнішнього вигляду екрану програми Android Studio

Запропоновані шаблони дозволяють заощадити час на написання стандартного коду для типових ситуацій.

Пару років назад був тільки один шаблон. Список шаблонів постійно поповнюється і тепер їх більше. Перерахую частину з них.

- Blank Activity
- Empty Activity
- Fullscreen Activity
- Google Maps Activity
- Google AdMod Ads Activity
- Login Activity
- Master / Detail Flow
- Navigation Drawer Activity (новинка)
- Scrolling Activity (новинка)

Шаблон Empty Activity призначений для звичайних телефонів. На зображенні над назвою видо приблизний вигляд програми з використанням даної заготовки.

Шаблон Master / Detail Flow призначений для планшетів з реалізацією двохпанельний режиму.

Шаблон Fullscreen Activity можна використовувати для ігор, коли потрібно додатковий простір без зайвих деталей.

Інші шаблони потрібні для створення додатків з гуглокартамі або сервісами Google Play.

Отже, вибраємо варіант Empty Activity і переходимо до наступного вікна рисунок 3.5.

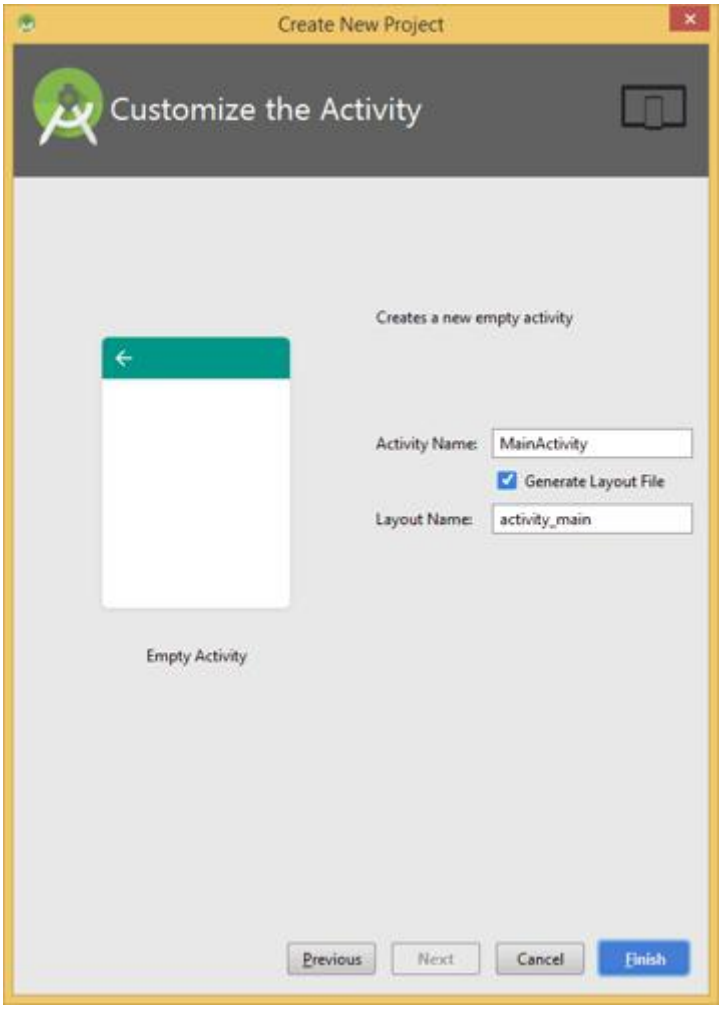

Рисунок 3.5- Діалогове вікно завершення створення проекту

Натискаємо кнопку Finish і дивимося, що далі буде.

А далі студія формує проект і створює необхідну структуру з різних файлів і папок.

3.1.2. Вибір мови програмування

В якості мови програмування для Android використовується Java. Для створення призначеного для користувача інтерфейсу використовується XML.

Java - об'єктно-орієнтована мова програмування , розроблена компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-який віртуальної Java-машині незалежно від комп'ютерної архітектури. Дата офіційного випуску - 23 травня 1995 року.

XML (EXtensible Markup Language - розширювана мова розмітки). Рекомендований Консорціумом Всесвітньої павутини (W3C). Специфікація XML описує XML-документи і частково описує поведінку XML-процесорів (програм, які читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся як мова з простим формальним синтаксисом, зручний для створення і обробки документів програмами і одночасно зручний для читання і створення документів людиною, з підкресленням націленості на використання в Інтернеті. Мова називається розширюваною, оскільки нею не фіксується розмітку, яка використовується в документах: розробник може створити розмітку відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови. Поєднання простого формального синтаксису, зручності для людини, розширюваності, а також базування на кодуваннях Юнікод для подання змісту документів призвело до широкого використання як власне XML, так і множини похідних

спеціалізованих мов на базі XML в найрізноманітніших програмних засобах.

3.1.3. Організація інтерфейсу з користувачем

Відповідно до запроектованих діалогових форм, у першому розділі (рисунки 1.4-1.9) реалізовуємо відповідні форми у вибраному середовищі програмування.

На рисунку 3.6 діалогове вікно входу в систему

Рисунок 3.6- Діалогове вікно для реалізації варіанту використання «Авторизація»

На рисунку 3.7 діалогове вікно реєстрації заняття

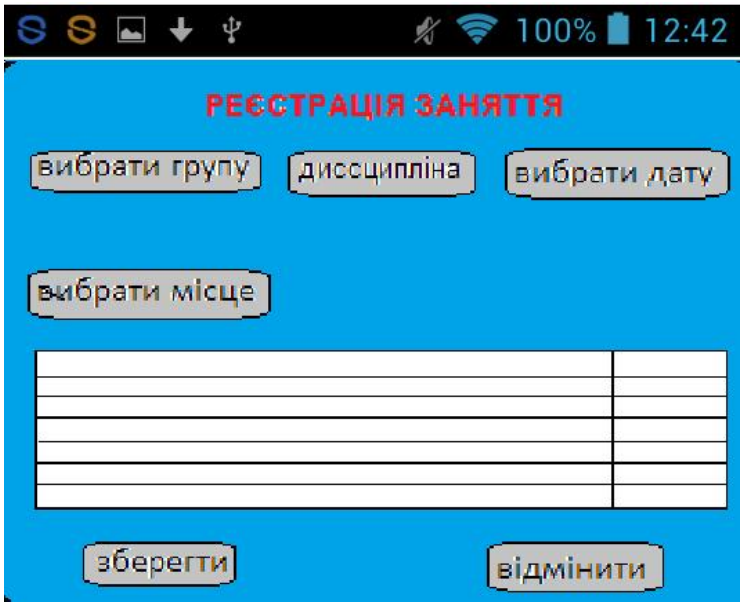

Рисунок 3.7- Діалогове вікно для реалізації варіанту використання «Реєстрація занять»

На рисунку 3.8 діалогове вікно для реалізації варіанта використання «Виставити бали»

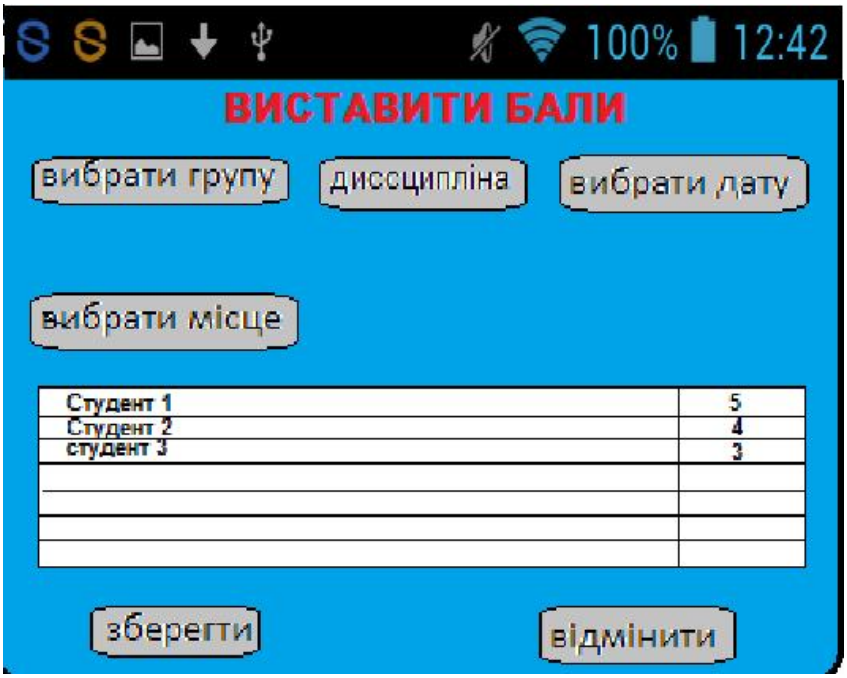

Рисунок 3.8- Діалогове вікно для реалізації варіанту використання «Виставити бали»

На рисунку 3.9 показано діалогове вікно для реалізації варіанту «Відмітити відвідування»

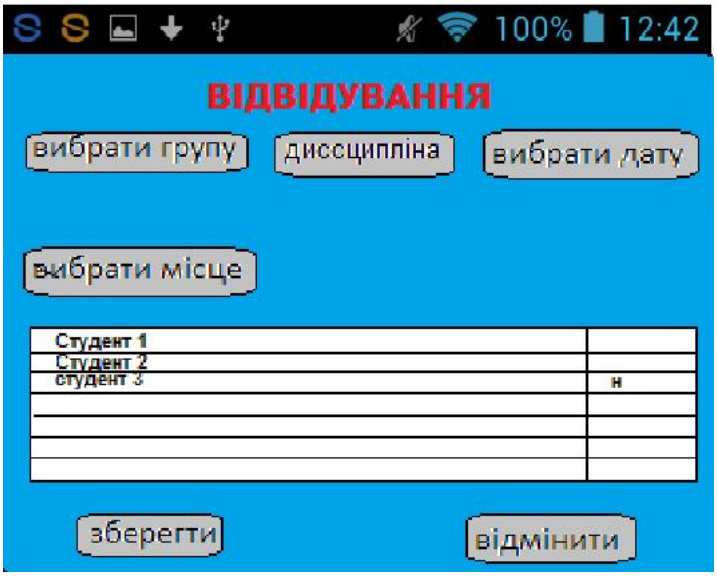

Рисунок 3.9- Діалогове вікно для реалізації варіанту «Відмітити відвідування»

3.2. . Програмна реалізація бази даних

Реалізація архітектури баз даних

Визначивши та описавши архітектуру баз даних, її було реалізовано у середовищі керування базами даних MS SQL Server 2008, дивись рисунок 3.9.

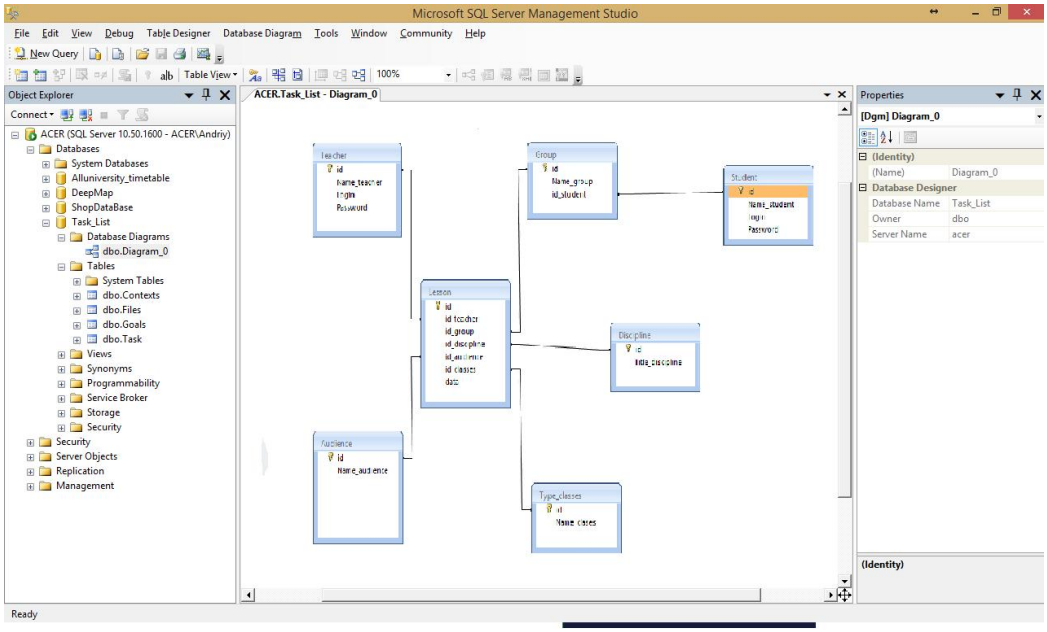

Рисунок 3.9- Реалізація бази даних

Висновки до третього розділу

В третьому розділі описано процес кодування (створення програмного коду) Мобільної програми для обліку успішності в академічній групі з використанням мови програмування Java. В цьому розділі також показані діалогові вікна для взаємодії користувача з системою.

РОЗДІЛ IV
РОЗДІЛ ТЕСТУВАННЯ ТА ДОСЛІДНОЇ ЕКСПЛУАТАЦІЇ МОБІЛЬНОЇ ПРОГРАМИ ДЛЯ ОБЛІКУ УСПІШНОСТІ В АКАДЕМІЧНІЙ ГРУПІ

4.1. Тестування

Тестування модулів (або блоків) являє собою процес тестування окремих підпрограм або процедур програми. Тут мається на увазі, що, перш ніж починати тестування програми в цілому, слід протестувати окремі невеликі модулі, що утворюють цю програму. Такий підхід мотивується трьома причинами.

По-перше, з'являється можливість управляти комбінаторикою тестування, оскільки спочатку увага концентрується на невеликих модулях програми.

По-друге, полегшується завдання налагодження програми, тобто виявлення місця помилки і виправлення тексту програми.

По-третє, допускається паралелізм, що дозволяє одночасно тестувати кілька модулів. Мета тестування модулів - порівняння функцій, що реалізуються модулем, зі специфікаціями його функцій або інтерфейсу. Тестування модулів в основному орієнтовано на принцип «білого ящика». Це пояснюється перш за все тим, що принцип «білого ящика» важче реалізувати при переході в подальшому до тестування більших одиниць, наприклад, програм в цілому.

Крім того, подальші етапи тестування орієнтовані на виявлення помилок різного типу, тобто помилок, не обов'язково пов'язаних з логікою програми, а виникають, наприклад, через невідповідність програми вимогам користувача. Є великий вибір можливих підходів, які можуть бути використані для злиття модулів у більші одиниці.

Покрокове тестування.

Реалізація процесу тестування модулів спирається на два ключових положення: побудова ефективного набору тестів і вибір способу, за допомогою якого модулі комбінуються при побудові з них робочої програми. Друге положення є важливим, так як воно задає форму написання тестів модуля, типи засобів, використовуваних при тестуванні, порядок кодування і тестування модулів, вартість генерації тестів і вартість налагодження. Розглянемо два підходи до комбінування модулів: покрокове і монолітне тестування.

Виникає питання: «Що краще - по виконати окремо тестування кожного модуля, а потім, комбінуючи їх, сформувати робочу програму або ж кожен модуль для тестування підключати до набору раніше протестованих модулів?».

Перший підхід зазвичай називають монолітним методом, або методом «великого удару», при тестуванні і складанні програми;

Другий підхід відомий як покроковий метод тестування або збірки.

Метод покрокового тестування передбачає, що модулі тестуються не ізольовано один від одного, а підключаються по черзі для виконання тесту до набору вже раніше протестованих модулів. Покроковий процес триває до тих пір, поки до набору протестованих модулів не буде підключений останній модуль.

Детального розбору обох методів ми робити не будемо, наведемо лише деякі загальні висновки.

1. Монолітне тестування вимагає великих витрат праці. При покроковому ж тестуванні «знизу-вгору» витрати праці скорочуються.

2. Витрата машинного часу при монолітному тестуванні менше.

3. Використання монолітного методу надає великі можливості для паралельної організації роботи на початковій фазі тестування (тестування всіх модулів одночасно). Це положення може мати важливе значення при виконанні великих проектів, в яких багато модулів і багато виконавців,

оскільки чисельність персоналу, який бере участь в проекті, максимальна на початковій фазі.

4. При покроковому тестуванні раніше виявляються помилки в інтерфейсах між модулями, оскільки раніше починається збірка програми. На противагу цьому при монолітному тестуванні модулі «не бачать один одного» до останньої фази процесу тестування.

5. Налагодження програм при покроковому тестуванні легше. Якщо є помилки в міжмодульних інтерфейси, а зазвичай так і буває, то при монолітному тестуванні вони можуть бути виявлені лише тоді, коли зібрана вся програма. У цей момент локалізувати помилку досить важко, оскільки вона може перебувати в будь-якому місці програми. Навпаки, при покроковому тестуванні помилки такого типу в основному пов'язані з тим модулем, який підключається останнім.

6. Результати покрокового тестування більш досконалі. На закінчення відзначимо, що п. 1, 4, 5, 6 демонструють переваги покрокового тестування, а п. 2 і 3 - його недоліки. Оскільки для сучасного етапу розвитку обчислювальної техніки характерні тенденції до зменшення вартості апаратури і збільшення вартості праці, наслідки помилок в математичному забезпеченні досить серйозні, а вартість усунення помилки тим менше, ніж раніше вона виявлена; переваги, зазначені в п. 1, 4, 5, 6, виступають на перший план. У той же час збиток, що наноситься недоліками (п. 2 і 3), невеликий. Все це дозволяє нам зробити висновок, що покрокове тестування найбільш прийнятний.

Переконавшись у перевагах покрокового тестування перед монолітним, досліджуємо дві можливі стратегії тестування: спадний і висхідний. Перш за все з'ясуємо в термінологію.

По-перше, терміни «спадний тестування», «спадна розробка», «спадний проектування» часто використовуються як синоніми. Дійсно, терміни «спадний тестування» і «спадна розробка» є синонімами (в тому

сенсі, що вони мають на увазі певну стратегію при тестуванні і створенні текстів модулів), але спадний проектування - це зовсім інший і незалежний процес. Програма, спроектована низхідним методом, може тестуватися і низхідним, і висхідним методами.

По-друге, висхідна розробка, або тестування, часто ототожнюється з монолітним тестуванням. Це непорозуміння виникає через те, що початок висхідного тестування ідентично монолітному при тестуванні нижніх або термінальних модулів. Але вище ми показали, що висхідне тестування насправді являє собою покрокову стратегію.

Висхідне тестування При висхідному підході програма збирається і тестується «знизу вгору». Тільки модулі самого нижнього рівня («термінальні» модулі, модулі, які не викликають інших модулів) тестуються ізольовано, автономно. Після того як тестування цих модулів завершено, виклик їх повинен бути так само надійний, як виклик вбудованої функції мови або оператор присвоювання. Потім тестуються модулі, безпосередньо викликають вже перевірені. Ці модулі більш високого рівня тестуються не автономно, а разом з уже перевіреними модулями більш низького рівня. Процес повторюється до тих пір, поки не буде досягнута вершина. Тут завершується і тестування модулів, і тестування сполучень програми. При висхідному тестуванні для кожного модуля необхідний драйвер: потрібно подавати тести відповідно до сполучення модуля, що тестується. Одне з можливих рішень - написати для кожного модуля невелику провідну програму. Тестові дані представляються як «вбудовані» безпосередньо в цю програму змінні і структури даних, і вона багаторазово викликає тестовий модуль, з кожним викликом передаючи йому нові тестові дані. Є і краще рішення: скористатися програмою тестування модулів - це інструмент тестування, що дозволяє описувати тести на спеціальній мові і позбавляє від необхідності писати драйвери. Тут відсутні проблеми, пов'язані з неможливістю або складністю створення всіх тестових ситуацій, характерні

для низхідного тестування. Драйвер як засіб тестування застосовується безпосередньо до того модулю, який тестується, де немає проміжних модулів, які слід брати до уваги. Аналізуючи інші проблеми, що виникають при низхідному тестуванні, можна помітити, що при висхідному тестуванні неможливо прийняти нерозумне рішення про суміщення тестування з проектуванням програми, оскільки не можна почати тестування до тих пір, поки не спроектовані модулі нижнього рівня. Не існує також і труднощів з незавершеністю тестування одного модуля при переході до тестування іншого, тому що при висхідному тестуванні з застосуванням декількох версій заглушки немає складнощів з поданням тестових даних.

Спадне тестування. Спадне тестування не є повною протилежністю висхідному, але в першому наближенні може розглядатися як таке. При низхідному підході програма збирається і тестується «зверху вниз». Ізольовано тестується тільки головний модуль. Після того як тестування цього модуля завершено, з ним з'єднуються (наприклад, редактором зв'язків) один за іншим модулі, безпосередньо викликані їм, і тестується отримана комбінація. Процес повторюється до тих пір, поки не будуть зібрані і перевірені всі модулі. При цьому підході виникають два питання:

1. Що робити, коли тестовий модуль викликає модуль нижчого рівня (якого в даний момент ще не існує)?

2. Як подаються тестові дані?

Відповідь на перше питання полягає в тому , що для імітації функцій відсутніх модулів програмуються модулі - заглушки, які моделюють функції відсутніх модулів. Цікавий і друге питання: в якій формі готуються тестові дані і як вони передаються програмі? Якби головний модуль містив всі необхідні операції введення і виведення, відповідь була б проста: тести пишуться у вигляді звичайних для користувачів зовнішніх даних і передаються програмі через виділені їй пристрої введення. Так, проте, трапляється рідко. У добре спроектованої програмі фізичні операції

введення-виведення виконуються на нижніх рівнях структури, оскільки фізичний введення-виведення - абстракція досить низького рівня. Тому для того, щоб вирішити проблему економічно ефективно, модулі додаються не в строго низхідній послідовності (всі модулі одного горизонтального рівня, потім модулі наступного рівня), а таким чином, щоб забезпечити функціонування операцій фізичного введення-виведення якомога швидше. Коли ця мета досягнута, спадний тестування отримує значну перевагу: всі подальші тести готуються в тій же формі, яка розрахована на користувача. Спадний метод має як переваги, так і недоліки в порівнянні з висхідним. Найзначніше перевага - то, що цей метод поєднує тестування модуля, тестування сполучень і частково тестування зовнішніх функцій. З цим же пов'язана інша його перевага: коли модулі введення-виведення вже підключені, тести можна готувати в зручному вигляді. Спадний підхід вигідний також в тому випадку, коли є сумніви щодо здійсненності програми в цілому або, коли в проекті програми можуть виявитися серйозні дефекти. Перевагою спадного підходу дуже часто вважають відсутність необхідності в драйверах; замість драйверів вам просто слід написати «заглушки». Спадний метод тестування має, на жаль, деякі недоліки. Основним з них є те, що модуль рідко тестується досконально відразу після його підключення. Справа в тому, що дуже докладне тестування деяких модулів може зажадати вкрай витончених заглушок. Програміст часто вирішує не витрачати багато часу на їх програмування, а замість цього пише прості заглушки і перевіряє лише частину умов в модулі. Він, звичайно, збирається повернутися і закінчити тестування розглянутого модуля пізніше, коли прибере заглушки. Такий план тестування - безумовно не краще рішення, оскільки про відкладення умовах часто забувають. Другий тонкий недолік спадного підходу полягає в тому, що він може породити віру в можливість почати програмування і тестування верхнього рівня програми до того, як вся програма буде повністю спроектована. Ця ідея на перший

погляд здається економічною, але зазвичай справа йде зовсім навпаки. Більшість опитаних проектувальників визнає, що проектування програми - процес інтерактивний. Рідко перший проект виявляється досконалим. Нормальний стиль проектування структури програми передбачає після закінчення проектування нижніх рівнів повернутися назад і підправити верхній рівень, внісши в нього деякі удосконалення або виправляючи помилки, або іноді навіть викинути проект і почати все спочатку, тому що розробник раптово побачив кращий підхід. Якщо ж головна частина програми вже запрограмована і відтестована, то виникає серйозний опір будь-яким поліпшенням її структури. В кінцевому підсумку за рахунок таких поліпшень зазвичай можна заощадити більше, ніж ті кілька днів або тижнів, які розраховує виграти проектувальник, приступаючи до програмування занадто рано.

4.1.1. Ручне тестування

Ручне тестування призначеного для користувача інтерфейсу проводиться тестувальником-оператором, який керується в своїй роботі описом тестових прикладів у вигляді набору сценаріїв. Кожен сценарій включає перерахування послідовності дій, які повинні виконати оператор, і опис важливих для аналізу результатів тестування у відповідь реакцій системи, відбиваних в призначеному для користувача інтерфейсі.

Форма запису сценарію для проведення ручного тестування - таблиця, в якій в одній колонці описані дії (кроки сценарію), в іншій - очікувана реакція системи, а третя призначена для запису того, чи співпала очікувана реакція системи з реальною і перерахування неспівпадань.

Результати проведення тестування наведені в таблиці 4.1.

Таблиця 4.1.

Результати проведення ручного тестування

№ п/п	Дія	Реакція системи	Результат
1.	Запустіть програму	Появляється стартове вікно програми рисунок 3.6	Вірно
2.	Ввести логін та пароль та натисніть «Увійти»	Перехід до діалогового вікна пошуку реєстрації заняття рисунок 3.7	Вірно
3.	Вибрати параметри заняття та зареєструвати його	У вікні вибираємо: групу, дисципліну, дату, місце. Результатом є зареєстроване заняття рисунок 3.7	Вірно
4.	Виставити оцінки	У формі рисунок 3.8 проти прізвищ студентів проставляємо оцінки	Вірно
5.	Відмітити присутність	У формі рисунок 3.9 проти прізвищ студентів які відсутні проставляємо відмітку «н»	Вірно

4.2. Розгортання програмного продукту

Для розгортання програмного продукту необхідно отримати інсталяційний пакет, зберегти його на постійному носії та запустити. В результаті запуску такого файлу в режимі діалогу на комп’ютер буде скопійовано необхідні для роботи файли у вказані директорії.

Висновки до четвертого розділу

В четвертому розділі проведено тестування Мобільної програми для обліку успішності в академічній групі, тут показано її придатність до використання, а також надано інформацію, що до його розгортання. Також є інструкція користувачу, яка охоплює всі сценарії роботи з засобом.

ВИСНОВКИ

В результаті проведеного вивчення питання стану програмного забезпечення для обліку успішності в академічній групі було поставлена задача створити своє аналогічне програмне забезпечення. Для створення такого програмного забезпечення використовували засіб розробки Android Studio, мова програмування Java, та технологія XML для зберігання даних. Це дало можливість створити програмне забезпечення мобільного типу. Тестування показало коректну роботу додатку. Інструкція до використання програмного забезпечення не є обов’язковою, так як програмний засіб є інтуїтивно зрозумілим та легким у використанні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маляров М.В./ Використання прикладних програм для автоматизації обліку поточних та підсумкових оцінок слухачів/ М.В.Маляров, В.О. Шулік//: http://nuczu.edu.ua/sciencearchive/Conferences/ProblemsOfCivil Protection/%CC%E0%EB%FF%F0%EE%E2%20%20%D8%F3%EB%E8%EA%E0.pdf

2. Швець Є.Я. Організація поточного і підсумкового контролю знань студентів при модульно-рейтинговій технології навчання. / Є. Я. Швець, Д. Є Швець // Гуманітарний вісник ЗДІА. – 2010. – Вип. 42. – С. 227 – 235

3. Програмний модуль "ПС-Журнал успішності-Web" . — Режим доступу до журн. : http://www.politek-soft.kiev.ua/index.php?do= newdevelopments& product=ps-gradebook-web

4. Philippe Kruchten, The Rational Unified Process: An Introduction, Third Edition (Филипп Крачтен, Rational Unified Process: вступ, третє видання, видавництво Addison-Wesley Professional 2003).

5. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition (Лен Басс, Пол Клементс і Рік Кацман , Практична архітектура програмного забезпечення, друге видавництво), видавництво Addison Wesley 2003 год.

6. Object Management Group Inc., Специфікація уніфікованої мови моделювання OMG версія 1.5, Документ номер 03-03-01. березень 2003 г.

7. James McGovern, et al., A Practical Guide to Enterprise Architecture (Джеймс Мак-Говерн і інші, Практичний посібник з архітектури корпорацій). Видавництво Prentice Hall 2004 р

8. Чен М.С. и др.Программирование на JAVA:1001 совет:Наиболее полное руководство по Java и Visual J++:Пер.с англ./Чен М.С.,Грифис С.В.,Изи Э.Ф.-Минск:Попурри,1997.-640с.ил.+ Прил.(1диск).

9. Майкл Эферган Java: справочник.- QUE Corporation, 1997, Издательство "Питер Ком", 1998

10. Ренеган Э.Дж.(мл.)1001 адрес WEB для программистов:Новейший путеводитель программиста по ресурсам World Wide Web:Пер.с англ..-Минск:Попурри,1997.-512с.ил.

11. Сокольский М.В.Все об Intranet и Internet.-М.:Элиот,1998.-254с.ил.

12. Джо Вебер Технология Java в подлиннике.- QUE Corporation, 1996, "BHV-Санкт-Петербург",1997

13. Джейсон Мейнджер Java: Основы программирования.- McGraw-Hill,Inc.,1996, Издательская группа BHV, Киев,1997

14. И.Ю.Баженова Язык программирования Java.- АО "Диалог-МИФИ", 1997

15. Джон Родли Создание Java-апплетов.- The Coriolis Group,Inc.,1996, Издательство НИПФ "ДиаСофт Лтд.",1996 5

16. Майкл Томас, Пратик Пател, Алан Хадсон, Доналд Болл(мл.) Секреты программирования для Internet на Java.- Ventana Press, Ventana Communications Group, U.S.A.,1996, Издательство "Питер Пресс", 1997

Додаток А

Код програми

```
package com.puppycrawl.tools.checkstyle;

import java.io.File;
import java.io.IOException;
import java.util.Locale;
import java.util.regex.Pattern;

import antlr.RecognitionException;
import antlr.TokenStreamException;

import com.puppycrawl.tools.checkstyle.api.CheckstyleException;
import com.puppycrawl.tools.checkstyle.api.DetailAST;
import com.puppycrawl.tools.checkstyle.api.DetailNode;
import com.puppycrawl.tools.checkstyle.api.FileContents;
import com.puppycrawl.tools.checkstyle.api.FileText;
import com.puppycrawl.tools.checkstyle.api.TokenTypes;
import com.puppycrawl.tools.checkstyle.utils.JavadocUtils;
import com.puppycrawl.tools.checkstyle.utils.TokenUtils;

/**
 * Class for printing AST to String.
 * @author Vladislav Lisetskii
 */
public final class AstTreeStringPrinter {
```

```
        /** Newline pattern. */
        private static final Pattern NEWLINE = Pattern.compile("\n");
        /** Return pattern. */
        private static final Pattern RETURN = Pattern.compile("\r");
        /** Tab pattern. */
        private static final Pattern TAB = Pattern.compile("\t");

        /** OS specific line separator. */
        private static final String LINE_SEPARATOR =
System.getProperty("line.separator");

        /** Prevent instances. */
        private AstTreeStringPrinter() {
            // no code
        }

        /**
         * Parse a file and print the parse tree.
         * @param file the file to print.
         * @param withComments true to include comments to AST
         * @return the AST of the file in String form.
         * @throws IOException if the file could not be read.
         * @throws CheckstyleException if the file is not a Java source.
         */
        public static String printFileAst(File file, boolean withComments)
                throws IOException, CheckstyleException {
            return printTree(parseFile(file, withComments));
        }
```

```
/**
 * Prints full AST (java + comments + javadoc) of the java file.
 * @param file java file
 * @return Full tree
 * @throws IOException Failed to open a file
 * @throws CheckstyleException error while parsing the file
 */
public static String printJavaAndJavadocTree(File file)
        throws IOException, CheckstyleException {
    final DetailAST tree = parseFile(file, true);
    return printJavaAndJavadocTree(tree);
}

/**
 * Prints full tree (java + comments + javadoc) of the DetailAST.
 * @param ast root DetailAST
 * @return Full tree
 */
private static String printJavaAndJavadocTree(DetailAST ast) {
    final StringBuilder messageBuilder = new StringBuilder();
    DetailAST node = ast;
    while (node != null) {
        if (node.getType() == TokenTypes.BLOCK_COMMENT_BEGIN
                && JavadocUtils.isJavadocComment(node)) {
            final String javadocTree = parseAndPrintJavadocTree(node);
            messageBuilder.append(javadocTree);
        }
        else {
```

```
                messageBuilder.append(getIndentation(node))
                    .append(getNodeInfo(node))
                    .append(LINE_SEPARATOR)
                    .append(printJavaAndJavadocTree(node.getFirstChild()));
            }
            node = node.getNextSibling();
        }
        return messageBuilder.toString();
    }

    /**
     * Parses block comment as javadoc and prints its tree.
     * @param node block comment begin
     * @return string javadoc tree
     */
    private static String parseAndPrintJavadocTree(DetailAST node) {
        final DetailNode tree = DetailNodeTreeStringPrinter.parseJavadocAsDetailNode(node);

        final String rootPrefix = getIndentation(node);
        final String prefix = rootPrefix.substring(0, rootPrefix.length() - 2) + "  ";
        return DetailNodeTreeStringPrinter.printTree(tree, rootPrefix, prefix);
    }

    /**
     * Parse a file and print the parse tree.
     * @param text the text to parse.
     * @param withComments true to include comments to AST
```

```
     * @return the AST of the file in String form.
     * @throws CheckstyleException if the file is not a Java source.
     */
    public static String printAst(FileText text, boolean withComments)
throws CheckstyleException {
        return printTree(parseFileText(text, withComments));
    }

    /**
     * Print AST.
     * @param ast the root AST node.
     * @return string AST.
     */
    private static String printTree(DetailAST ast) {
        final StringBuilder messageBuilder = new StringBuilder();
        DetailAST node = ast;
        while (node != null) {
            messageBuilder.append(getIndentation(node))
                    .append(getNodeInfo(node))
                    .append(LINE_SEPARATOR)
                    .append(printTree(node.getFirstChild()));
            node = node.getNextSibling();
        }
        return messageBuilder.toString();
    }

    /**
     * Get string representation of the node as token name,
     * node text, line number and column number.
```

```
     * @param node DetailAST
     * @return node info
     */
    private static String getNodeInfo(DetailAST node) {
        return TokenUtils.getTokenName(node.getType())
                + " -> " + excapeAllControlChars(node.getText())
                + " [" + node.getLineNo() + ':' + node.getColumnNo() + ']';
    }

    /**
     * Get indentation for an AST node.
     * @param ast the AST to get the indentation for.
     * @return the indentation in String format.
     */
    private static String getIndentation(DetailAST ast) {
        final boolean isLastChild = ast.getNextSibling() == null;
        DetailAST node = ast;
        final StringBuilder indentation = new StringBuilder();
        while (node.getParent() != null) {
            node = node.getParent();
            if (node.getParent() == null) {
                if (isLastChild) {
                    // only ASCII symbols must be used due to
                    // problems with running tests on Windows
                    indentation.append("`--");
                }
                else {
                    indentation.append("|--");
                }
```

```
            }
            else {
                if (node.getNextSibling() == null) {
                    indentation.insert(0, "    ");
                }
                else {
                    indentation.insert(0, "|   ");
                }
            }
        }
        return indentation.toString();
    }

    /**
     * Replace all control chars with excaped symbols.
     * @param text the String to process.
     * @return the processed String with all control chars excaped.
     */
    private static String excapeAllControlChars(String text) {
        final String textWithoutNewlines = NEWLINE.matcher(text).replaceAll("\\\\n");
        final String textWithoutReturns = RETURN.matcher(textWithoutNewlines).replaceAll("\\\\r");
        return TAB.matcher(textWithoutReturns).replaceAll("\\\\t");
    }

    /**
     * Parse a file and return the parse tree.
     * @param file the file to parse.
```

```
     * @param withComments true to include comment nodes to the tree
     * @return the root node of the parse tree.
     * @throws IOException if the file could not be read.
     * @throws CheckstyleException if the file is not a Java source.
     */
    private static DetailAST parseFile(File file, boolean withComments)
            throws IOException, CheckstyleException {
        final FileText text = new FileText(file.getAbsoluteFile(),
            System.getProperty("file.encoding", "UTF-8"));
        return parseFileText(text, withComments);
    }

    /**
     * Parse a text and return the parse tree.
     * @param text the text to parse.
     * @param withComments true to include comment nodes to the tree
     * @return the root node of the parse tree.
     * @throws CheckstyleException if the file is not a Java source.
     */
    private static DetailAST parseFileText(FileText text, boolean
withComments)
            throws CheckstyleException {
        final FileContents contents = new FileContents(text);
        final DetailAST result;
        try {
            if (withComments) {
                result = TreeWalker.parseWithComments(contents);
            }
            else {
```

```
                result = TreeWalker.parse(contents);
            }
        }
        catch (RecognitionException | TokenStreamException ex) {
            final String exceptionMsg = String.format(Locale.ROOT,
                "%s occurred during the analysis of file %s.",
                ex.getClass().getSimpleName(), text.getFile().getPath());
            throw new CheckstyleException(exceptionMsg, ex);
        }

        return result;
    }
}
```