МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

РОЗРОБЛЕННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СЕРВІСУ МОНІТОРИНГУ ВІДВІДУВАНЬ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія
програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
першого (бакалаврського) рівня освіти
Леденчук Станіслав Вадимович
Керівники
кандидат педагогічних наук, доцент
Вінник Максим Олександрович
кандидат фізико-математичних наук,
доцент Єрмолаєв Вадим Анатолійович
Рецензент
кандидат педагогічних наук, доцент
Єрмакова-Черченко Наталія
Олександрівна

Херсон – 2020

ЗМІСТ

ВСТУП

Актуальність. На сьогодні питання відвідуваності занять студентами є дуже актуальним, нажаль не всі студенти доброякісно відносяться до обов'язку ходити на пари. В наш час ІТ, використовувати старі методи контролю в вигляді журналу не є раціональним адже данні можна сфальсифікувати чи просто не занести до нього. Також важливо враховувати людський фактор при якому дані переважно заносить староста який може «прикрити» товаришів. Автоматизація контролю має виправити цю ситуацію та полегшити процес моніторингу.

Тому **метою** дипломної роботи стало створення зручної та зрозумілої клієнтської частини додатку для моніторингу за відвідуваністю. Додаток повинен бути крос-платформним, щоб користувачі могли використовувати його незалежно від платформи.

Об'єктом дослідження є облік відвідування занять студентами та технології для розробки клієнтського інтерфейсу.

Предметом дослідження є засоби автоматизації обліку відвідувань занять студентами шляхом сучасних технологій.

Задачі:

1. Розглянути існуючу систему моніторингу відвідувань.
2. Пошук та аналіз аналогів розроблюваної системи.
3. Ознайомлення та вивчення засобів для крос-платформної розробки.
4. Написання програмного коду додатку.

Структура й обсяг роботи. Дипломна робота складається з вступу, трьох розділів, висновків, списку використаної літератури та додатку. Повний обсяг дипломної роботи складає 44 сторінки.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд системи управління відвідуваності в університеті

Державний вищий навчальний заклад (далі - ВНЗ) є суб'єктом освітньої діяльності, яка здійснюється з метою задоволення освітніх потреб особи, суспільства і держави [1]. На сьогодні в кожному університеті існує своя система для обліку відвідуваності студентів. Управління ВНЗ здійснюють за допомогою багатьох відділів та підрозділів. Кожен з цих підрозділів невід’ємно пов'язаний з іншим. Незважаючи на те що у кожному закладі назви усіх цих розділів може різнитися, але їх сутність є досить схожою. Пропоную розглянути структурні підрозділи які здійснюють контроль відвідуваності. Переважно в більшості закладів керівництво (тобто вирішення основних питань діяльності закладу) здійснюється ректоратом (найвищим адміністративним органом). В дійсності саме цей орган є головним і вирішує усі питання що пов’язані з роботою та управлінням ВНЗ. Таким чином, він здійснює керування усіма структурними підрозділами вищого навчального закладу та забезпечує їх ефективну роботу. Далі, у системі управління навчальними процесами йде «навчальний підрозділ». Навчальний підрозділ – це орган що здійснює підготовку студентів за обраними напрямками [2]. Він складається з організаційних і навчально-наукових підрозділів (факультетів). Факультети в свою чергу мають власний керуючий орган (деканат) та власні наукові-дослідні підрозділи (кафедри), які здійснюють навчально-виховну та методичну діяльність. В свою чергу останні із згаданих органів, безпосередньо керують освітнім процесом та слідкують за його виконанням. Нижче зображено основну схему структурної організації управління ВНЗ, що здійснює та контролює навчальний процес.

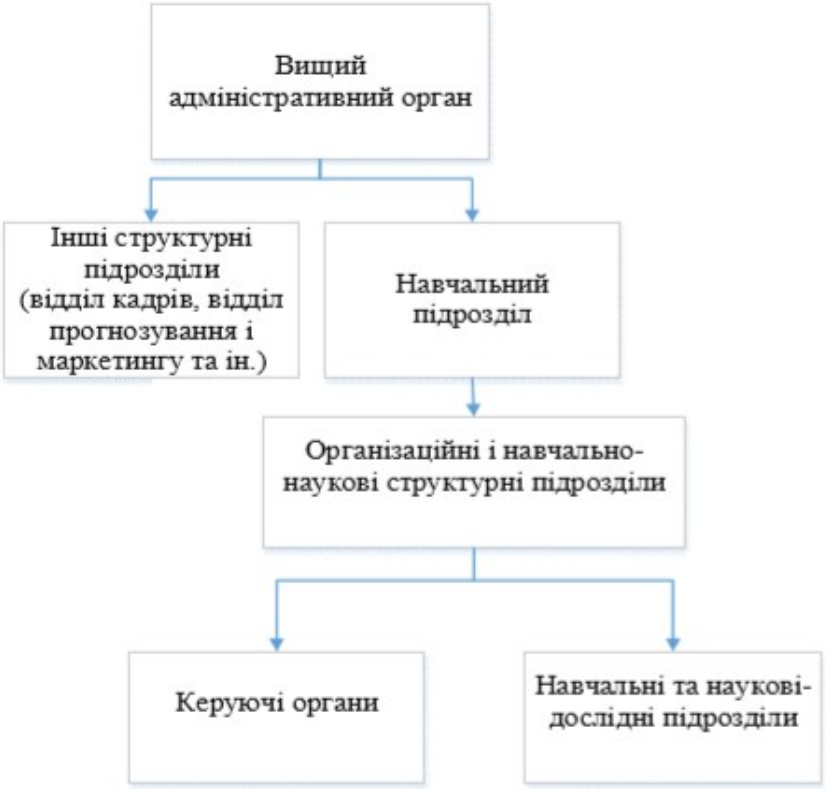


Рис. 1.1 Схема структурної організації управління ВНЗ

Проблеми з відвідуваністю виникали протягом усієї історії від початку створення ВНЗ. На сьогоднішній день це питання стало ще актуальнішим, адже існує багато причин що підштовхують студентів до порушення своїх обов'язків відвідувати заняття. Незмінним залишається одне – потреба в обліку відвідуваності занять. Існуючі системи управління є ненадійними та незручними, саме тому було прийняте рішення про створення власного додатку який би допомагав вирішувати ці проблеми. Пропоную детальніше ознайомитися з існуючим процесом обліку відвідуваності. У кожній групі існує староста який вручну вносить данні кожного студента у журнал обліку який він отримує в деканаті. Вигляд журналу обліку відвідуваності наведено на рисунку нижче.

Рис. 1.2 Вигляд журналу обліку занять

В нього староста вносить данні про присутність чи відсутність студента на занятті, переважно цей процес відбувається кожного дня. Розглянемо детальніше:

- журнал передається до відповідного керуючого органу (деканату) де данні перевіряються та переносяться для зберігання.

- в деканаті же вносяться ці дані в спеціальну систему(базу даних) самостійно, або це робить староста.

Потім згідно з використовуваним методом, дані обробляються та роблять висновки. Насамперед вираховують кількість пропусків кожного студента і дивляться чи була на те причина. Студенти що мають багато пропусків без належної на те причини за рішенням навчального закладу можуть потрапити у списки на виключення. Увесь цей процес обліку існує вже давно та має купу недоліків. Помилка може трапитись на будь-якому кроці, тим паче що інформація обробляється вручну. Наприклад не доброчесні (але добрі товариші) старости можуть подати не точні данні таким чином «прикривши» своїх товаришів. Деякі викладачі можуть вести власний журнал обліку і таким чином нівелювати цю проблему, але це робить процес ще більш громіздким і змушує викладача особисто відносити данні до деканату і перевіряти їх. Як наслідок уся ця схема досить повільна та потребує чіткого контролю. Проблемою є також те що у зв’язку з повільністю внесення та перевірці

даних студент може бути вчасно не попередженим про проблеми з відвідуваністю аж до моменту перевищення ліміту. Розглянувши цю систему, очевидно, що усі ці проблеми потрібно вирішити зручним способом, яким є створення мобільного додатку.

1.2. Пошук та аналіз існуючих систем

На сьогоднішній день в світі існують аналоги програмних продуктів, які дозволяють здійснювати автоматизований процес обліку відвідуваності. Проте, потрібно відзначити, що кожен програмний продукт має свої методи для вирішення проблеми і є рішенням для конкретного навчального закладу. Тому важливим фактором є пошук таких систем, аналіз їх переваг та недоліків та вибір критеріїв які повинен мати розроблюваний продукт. Мною було розглянуто велика кількість засобів для здійснення контролю відвідуваності. І було обрано одні з найпопулярніших систем, такі як: «BioTime», «MyAttendanceTracker», «Smiles.Школьная карта». Розглянемо їх трішки детальніше:

- «BioTime» - загалом система створена для співробітників різних компаній [3]. Особливо актуальною стала останнім часом у зв’язку з епідемією. Система дозволяє дистанційно відмітитись на робочому місці за допомогою мобільного додатку. Співробітник заходить у додаток який визначає його місцезнаходження та ставить відмітку про наявність яку можна підтвердити за допомогою відбитку пальця та зробивши «селфі». Нижче на рисунках можна побачити як саме співробітник відмічається, та як за цим слідкує керівник.

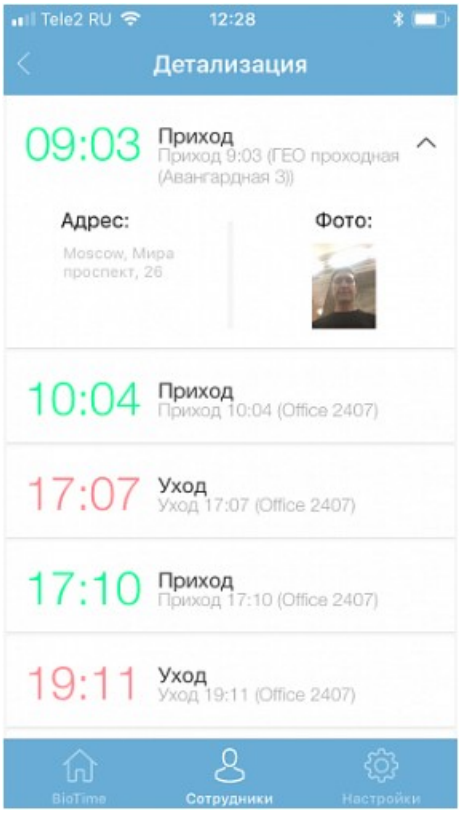


Рис. 1.3 Вигляд для керівника

Рис. 1.4 Відмітка співробітника

- «MyAttendance Tracker» - це веб-ресурс що надає змогу для контролю відвідуваності [4]. Після першого запуску з’явиться помічник що допоможе користувачу розібратись в усіх деталях сайту. Так як продукт розроблявся як помічник викладача тому він сам повинен вносити данні студентів та формулювати списки. Цікавою особливістю системи є створення різноманітних звітів про відвідуваність, та що викладач сам вирішує кому до яких звітів надавати доступ. На рисунку далі зображено вікно створеної мною групи в яку я попередньо додав кількох студентів.

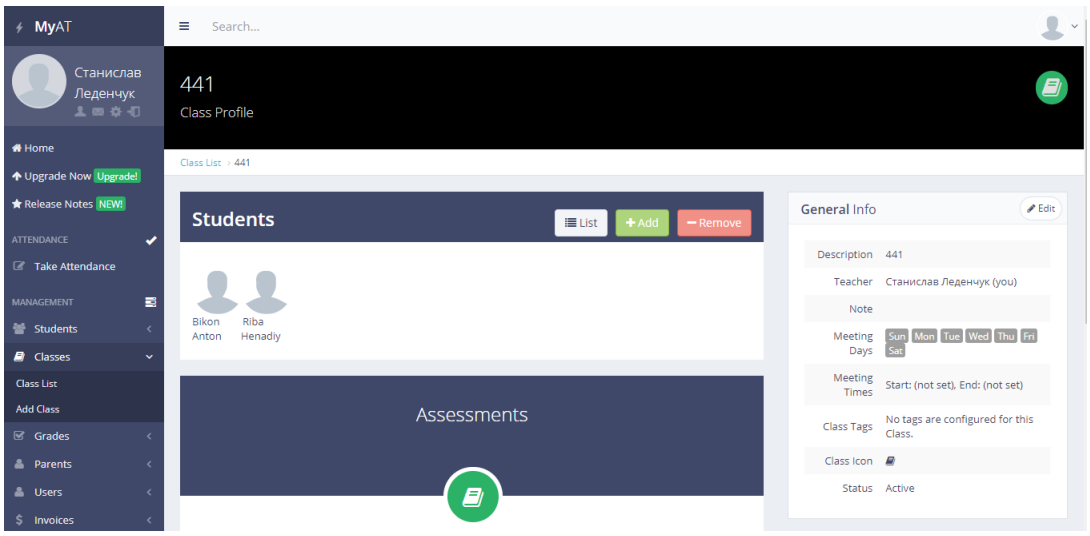


Рис. 1.5 Вікно створеної групи з доданими студентами

- «Smiles. Школьная карта» - система для забезпечення безпеки на території школи [5]. Учням та співробітникам видаються ідентифікаційні картки, які використовують для проходження через турнікети системи. Це дозволяє оберегти учнів від стороннього втручання та фіксує час (коли дитина пішла або прийшла) в системі та відправляє батькам на телефон СМС в якому інформує про дії учня. Також в системі створений електронний щоденник та журнал що дозволяє слідкувати за пропусками та оцінками учнів. Система є цілковитим аналогом щоденника для учня та журналу для вчителя. На ілюстрації нижче зображено вікно продукту.

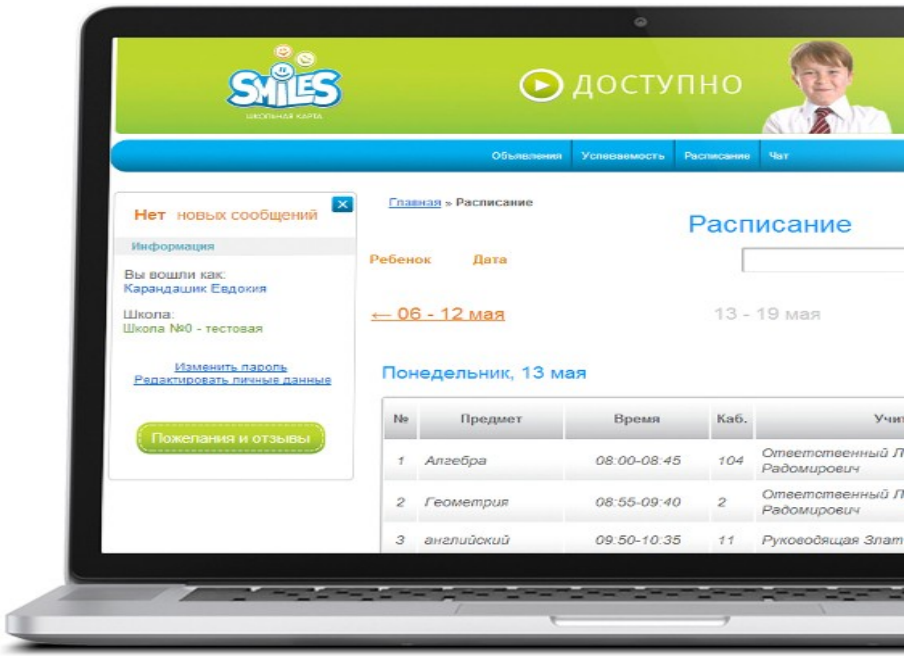


Рис. 1.6 Вигляд додатку «Smiles. Школьная карта»

Кожен із цих засобів забезпечує унікальний підхід до вирішення проблеми відвідуваності. У таблиці нижче наведена порівняльна характеристика даних продуктів.

Таблиця 1.1

Порівняльна характеристика програмних продуктів

Назва продукту	«BioTime»	«MyAttendance Tracker»	«Smiles. Школьная карта»
Інтерфейс користувача	Зрозумілий, але не для всіх	Зрозумілий.	Зрозумілий, орієнтований на дітей.
Допомога користувачу	Присутня документація	Присутні підказки	Присутнє меню «допомога»
Функціонал для студентів	Реалізований	Реалізований	Реалізований
Робота основних	Неможлива, оскільки потрібне	Неможлива, оскільки це веб-	Неможлива оскільки потребує

Продовження табл. 1.1

функції без доступу до Інтернету	підключення до зовнішньої БД на сервері	орієнтована система	підключення до зовнішньої БД на сервері
Зв’язок із студентами	Присутній	Присутній	Присутній
Потреба у додаткових апаратних інтерфейсах	Не потребує	Не потребує	Потребує встановлення турнікетів
Доступність	Безкоштовно перші три місяці	Безкоштовний	Витрати на устаткування

І так порівнявши данні продукти можна зробити висновок що основними недоліками даних систем є вартість необхідного для належної роботи устаткування, великий персонал для підтримки працездатності систем та підтримки клієнтів. Серед переваг можна виділити доволі зрозумілий інтерфейс, реалізація зв’язку зі студентами, можливість відслідковувати пересування, хоча як на мене це не зовсім правильне рішення адже дані про пересування це особисто конфіденційна інформація і не кожен згоден на те щоб за ним постійно стежили. Отже розроблювана система повинна мати зручний та зрозумілий інтерфейс користувача, повинна забезпечувати легкий спосіб підтвердження присутності та не потребувати додаткових апаратних інтерфейсів.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1. Створення та опис діаграми варіантів використання

У розроблюваній клієнтській частині системи користувачі поділяються за декількома ролями: студент, та викладач. У них є як спільні так і відмінні одне від одного функції. Наприклад і студент і викладач може переглянути свої пари, але інтерфейс трішки різниться тим що студент може бачити лише свої пари та викладача що їх веде. Викладач в свою чергу може також бачити свої пари та списки усіх груп зі студентами що повинні бути присутні. На рисунку 2.1 представлена діаграма варіантів використання додатку.

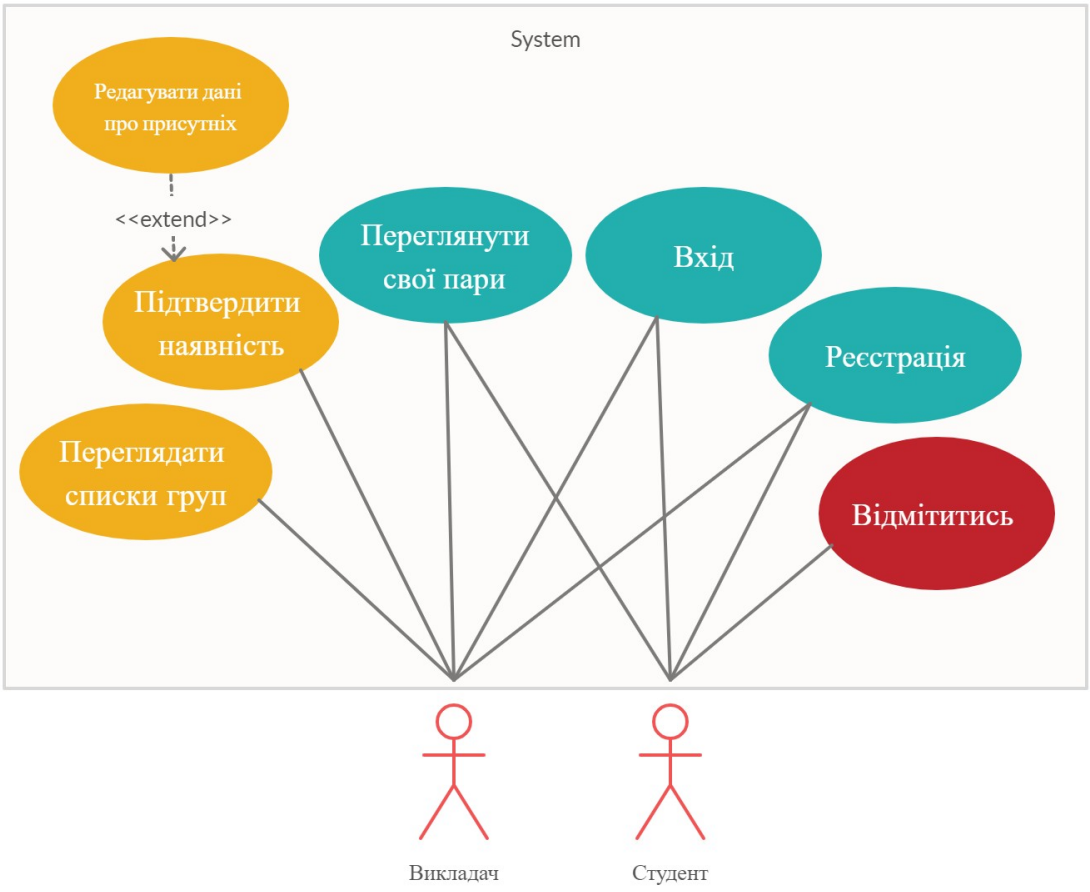


Рис 2.1 Діаграма варіантів використання

Розглянемо більш детально кожен із варіантів використання. Перше що необхідно зробити користувачу це авторизуватись. Зробити він це може кількома способами або адміністратор створить обліковий засіб на сервері або пройшовши звичайну форму реєстрації. Це необхідно щоб мати доступ до основного функціоналу користувача. Основною функцією на цьому етапі є визначення ролі користувача та надійне зберігання даних кожного клієнта. Нижче наведена таблиця з описом варіанту використання «Реєстрація»/«Вхід».

Таблиця 2.1

Варіант використання «Реєстрація»/»Вхід»

Мета використання	Метою є ідентифікація користувача та надання йому відповідних прав
Ролі	Студент, викладач
Передумова	Натиснути кнопку «Реєстрація»/«Вхід»
Збудник	Перший запуск додатку
Сценарій	1- Надання відповідних даних для реєстрації/входу 2- Натиснути кнопку «Реєстрація»/«Вхід»
Результат	Повідомлення про успішність/невдачу при реєстрації/авторизації

Далі розглянемо варіант використання «Переглянути свої пари». Данні про те які в користувача пари мають відображатись в залежності від того яка обрана роль. Кожен користувач буде бачити лише ті заняття які відносяться конкретно до нього. Та зможе бачити свої минулі/поточні/заплановані пари. Якщо це студент то натиснувши на потрібну пару він може відмітитись. У випадку викладача можна

переглянути списки усіх та окремих груп. У таблиці нижче наведено опис варіанту використання «Переглянути свої пари»

Таблиця 2.2

Варіант використання «Переглянути свої пари»

Мета використання	Відображення особистого розкладу кожного користувача
Ролі	Студент, викладач
Передумова	Успішна авторизація
Збудник	Вхід у додаток
Сценарій	1- Вибір в меню минулі/поточні/заплановані пари 2- Перегляд пар
Результат	Відображення потрібних пар

Наступним опишемо варіант використання «Відмітитись». Основною задачею цієї функції є можливість для студента відмітити свою наявність на парі. Варто знати, що студент може відмітитись лише підчас пари. У таблиці нижче наведено опис цього варіанту використання.

Таблиця 2.3

Варіант використання «Відмітитись»

Мета використання	Відмітка студента про присутність на парі
Ролі	Студент

Продовження табл. 2.3

Передумова	Успішна авторизація
Збудник	Вхід у додаток
Сценарій	1- Вибір потрібної пари

	2- Натискання кнопки «Відмітитись»
Результат	Відправлення даних на підтвердження викладачу

Викладачу надається унікальний функціонал з можливістю перегляду списків груп що повинні бути на парі. У цій функції викладач може як переглянути списки усіх груп так і обрати конкретну групу. Нижче в таблиці наведено опис цього використання.

Таблиця 2.4

Варіант використання «Переглядати списки груп»

Мета використання	Переглянути списки груп та студентів
Ролі	Викладач
Передумова	Успішна авторизація
Збудник	Вхід у додаток
Сценарій	1- Вибір конкретної пари із меню минулі/поточні/заплановані 2- Перегляд списків студентів
Результат	Відображення списків студентів та груп

Варіант використання «Редагувати дані про присутніх» дозволяє викладачу в разі необхідності редагувати та виправляти дані про деяких студентів. Наприклад якщо у студента немає телефону чи якісь проблеми із ним викладач може самостійно внести дані, або ж навпаки якщо якийсь студент вирішив схитрити та відмітитись «зайцем» викладач може зняти мітку та в майбутньому такому студентові не позаздриш. В таблиці нижче наведено опис цього варіанту використання.

Таблиця 2.5

Варіант використання «Редагувати дані про присутніх»

Мета використання	Можливість відредагувати списки студентів що відмітились/не відмітились
Ролі	Викладач
Передумова	Успішна авторизація
Збудник	Натискання на певну пару
Сценарій	1- Натискання на конкретну пару 2- Змінення стану кнопки про наявність
Результат	Змінення даних про присутність студентів на парі

І врешті решт викладач може підтвердити списки студентів за допомогою варіанту використання «Підтвердити наявність». Дані в свою чергу відправляються на сервер де і зберігаються. Нижче в таблиці наведено детальний опис цього варіанту.

Таблиця 2.6

Варіант використання «Підтвердити наявність»

Мета використання	Можливість підтвердити списки студентів що відмітились
Ролі	Викладач
Передумова	Успішна авторизація

Продовження табл. 2.6

Збудник	Натискання на певну пару
Сценарій	1- Натискання на конкретну пару 2- Натискання на кнопку «Підтвердити»
Результат	Відправлення даних про присутніх до серверу

2.2. Що таке крос-платформна розробка та її переваги

Що ж настав час розповісти за крос-платформну розробку і навіщо її використовувати [6]. Крос-платформна розробка дозволяє програмному забезпеченню працювати більш ніж на одній операційній

системі (далі ОП) чи платформі під яку воно і розроблялось. Це дозволяє суттєво зменшити кількість часу необхідного для розробки під окрему ОП, зменшити витрати і персонал необхідний для створення та підтримки продукту, дозволить вносити загальні зміни для усіх ОП.

Переваги крос-платформної розробки:

1. Єдиний код для додатку. Тобто не потрібно для кожної системи писати свій код зі своєю логікою та нюансами, а достатньо створити один який потім буде інтерпретований під кожну систему;
2. Можливість оновлення та внесення змін безпосередньо для усіх платформ одночасно Так як код є в більшості своїм єдиний для кожної платформи то зрозуміло що і зміни які ми вносимо будуть відображатись для кожної системи;
3. Економія часу, витрат та персоналу для розробки додатку.

Недоліки крос-платформної розробки:

1. Можлива дещо повільніша робота додатку. Досліди із різними крос-платформними фраймворками показали що створені додатки дещо повільніше працюють. Це пов’язано із тим що в більшості своїм використовуються бібліотеки .Net які потім інтерпретуються під кожну платформу [7];
2. Втрачається змога використання особливостей кожної із систем;
3. Великий розмір створених додатків.

2.3. Xamarin як зручний засіб для створення крос-платформних додатків

Отже що таке Xamarin? Це американська компанія що займається розробкою програмного забезпечення. Була заснована в травні 2011 року Міґелем де Ікаса і Нетом Фрідманом. Цей засіб дозволяє створювати

сучасні продуктивні додатки для різних платформ: Android, iOS та Windows . Інтерфейс під кожну платформу створюється нативний з використанням нативних контролів та з використанням нативного інструментарію операційних систем відповідних мобільних. Користуючись Xamarin ми отримуємо цілком нативний інтерфейс. Загальним є логіка яка знаходиться на вищих рівнях інтерфейсу й пишеться за допомогою мови C#. Все це компілюється в нативний код мобільних платформ за рахунок чого ми отримуємо доволі високу продуктивність. Дуже важливим моментом є компіляція та швидкість яка є дуже близькою до нативної. У випадку використання iOS використовується Ahead of time (AOT) компіляція в результаті ми отримуємо цілковитий ARM (від англ. Advanced RISC Machine) binary для загрузки в Apple`s App Store. У випадку Android ми можемо використовувати аналогічно Just In Time (JIT) компіляцію, також підтримуються Android AOT компіляція [8].

Слід відмітити що платформа Xamarin для своєї роботи використовує IDE Visual Studio.

2.4. Xamarin.Forms як засіб для створення інтерфейсу

Що таке Xamarin.Forms? Це зручний інструмент для створення інтерфейсу користувача для різних платформ на основі загального коду. Дозволяє створювати інтерфейси в XAML - (англ. eXtensible Application Markup Language) це мова розмітки для додатків, за допомогою коду мови C#. Ці інтерфейси підготовлюються до перегляду для кожної платформи як власні елементи керування [9]. Переваги Xamarin.Forms :

1. Ще більше загального коду. Що мається на увазі, якщо в звичайному Xamarin використовується 40-50% загального коду то тут використовується до 90%, що в свою чергу є вагомим аргументом з боку розробки;

2. Багато готових елементів керування. Тобто не потрібно самому все створювати а можна скористатись вже готовими елементами що входять до пакету Xamarin.Forms;
3. Зручні шаблони розташування елементів інтерфейсу. Кожна платформа використовує свої так звані стандарти та шаблони як повинні розміщуватись елементи на сторінці. Xamarine.Forms надає доступ до Layouts елементів що дозволяє налагодити правильне масштабування інтерфейсу під різні розміри мобільних екранів;
4. Можливість перевикористання коду в Xamarine Native App. Тобто якщо ви на Xamarine.Forms створили якісь елементи інтерфейсу і хочете їх використати в існуючих Xamarine Native додатках, ви можете це зробити так як реалізована інтеграція між цими двома підходами розробки;
5. Робота та зміни в режимі реального часу одразу відображаються на пристрої (емуляторі чи реальному телефоні). Під час написання коду якщо змінити наприклад колір елементу чи додати щось нове то зміни безпосередньо можна буде побачити і на реальному пристрої чи емуляторі який ми використовуємо для тестування;
6. Можливість роботи як на Windows так і на Mac OS. Створюючи свої додатки ми можемо працювати як в звичайному Visual Studio так і в Visual Studio for Mac;
7. Можливість перетворення будь-якої .Net бібліотеки в нативну. Передбачає можливості створення бібліотек на C# які потім збираються під нативні бібліотеки різних платформ. Наприклад написав якусь бібліотеку на C# можете її збирати і отримувати нативну бібліотеку під iOS, Android, macOS, tvOS і навіть Linux.

На рисунку нижче зображено схему роботи Xamarin.Forms.

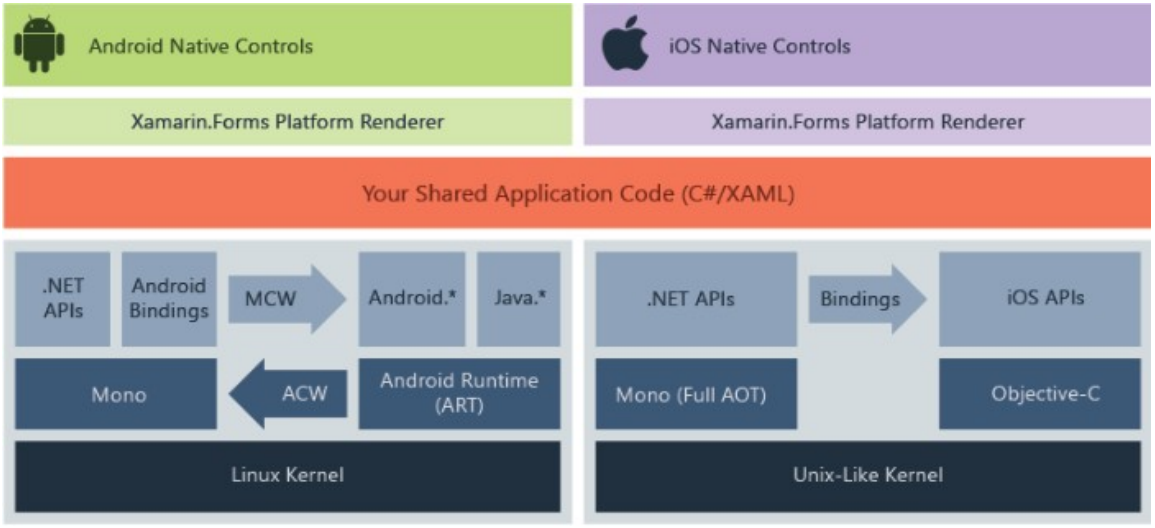


Рис. 2.2 Схема роботи Xamarin.Forms

2.5. Огляд IDE Visual Studio

Отже для роботи з Xamarin потрібне середовище розробки, яким є Visual Studio. Це інтегроване середовище розробки (IDE) розроблене компанією Microsoft що включає в себе ряд інструментів для роботи з багатьма платформами. Дата випуску - 1997 рік [10].

Переваги:

1. Велика спільнота – починаючи з 1997 року Visual Studio встигла завоювати доволі велику аудиторію користувачів адже надає доступ до основних функцій безкоштовно;
2. Постійні оновлення – так як Visual Studio є доволі популярним IDE тому для нього доволі часто виходять оновлення в яких виправляються помилки та покращується робота;
3. Можливість використання усіх додаткових засобів Xamarin та Xamarin.Forms – так як це все розробила одна компанія Microsoft то Visual Studio надає доступ до усіх засобів що розроблені для цього фреймворку.
4. Інтуїтивний стиль кодування – під час написання коду Visual Studio автоматично його редагує вставляючи потрібні відступи

та застосовуючи різнокольорове кодування для кращої читабельності;

5. Можливість налагодження – під час виникнення помилки Visual Studio надає зручні засоби для пошуку та усунення їх. Наприклад розробник може виконувати код по кожному рядку встановлюючи інтелектуальні точки переривання.

Отже основною перевагою є все ж таки підтримка, рівень інтеграції з усіма необхідними компонентами розробки та постійні оновлення що спрощують роботу та виправляють помилки які виникають в ході роботи. На рисунку нижче бачимо саме середовище:

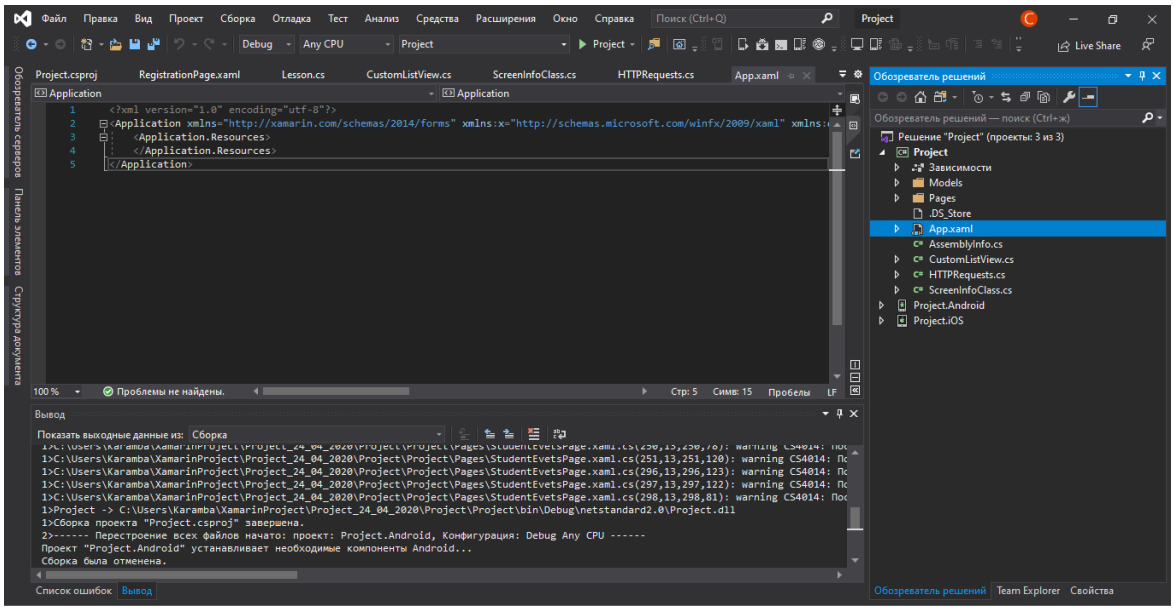


Рис. 2.3 Вигляд IDE середовища Visual Studio

РОЗДІЛ 3

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СЕРВІСУ

3.1. Створення екранів інтерфейсу

Отже для початку ми приступили до створення головних екранів додатку. В розроблюваному проєкті ми створюємо папку Pages в якій і будемо тримати усі створені екрани. На рисунку нижче зображена ця папка та кожен з екранів.

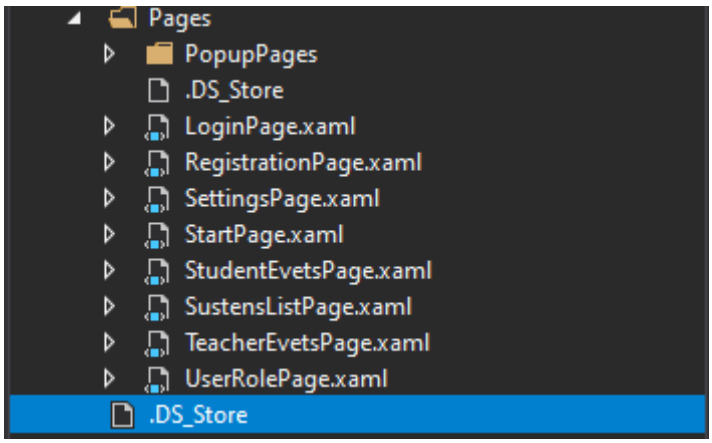


Рис. 3.1 Папка екранів проєкту

Розглянемо більш детально кожен з екранів по порядку:

1. *StartPage*

Власне кажучи як зрозуміло з назви екрану, це перше що бачить перед собою користувач зайшовши до додатку. На цьому екрані розміщена привітальне слово та дві кнопки дозволяючи перейти до меню реєстрації та входу до додатку відповідно.

Розглянемо приклад створення кнопки «Вхід»:

Код на XAML:

```
<buttons:SfButton Clicked="OnLogin"
                        BackgroundColor="Transparent"
                        BackgroundImage="login_button.png"
                        Text="Вхід"
                        FontSize="20"
                        HeightRequest="65"
                        WidthRequest="300"
                        RelativeLayout.XConstraint = "{ConstraintExpression
Type=RelativeToParent, Property=Width, Factor=0.5,Constant=-150}"
                        RelativeLayout.YConstraint = "{ConstraintExpression
Type=RelativeToParent, Property=Y, Constant=270}">

                    <VisualStateManager.VisualStateGroups>
                       <VisualStateGroup x:Name="CommonStates">
                           <VisualState x:Name="Normal">
                               <VisualState.Setters>
                                  <Setter Property="TextColor" Value="White" />
                               </VisualState.Setters>
                           </VisualState>

                           <VisualState x:Name="Pressed">
                               <VisualState.Setters>
                                   <Setter Property="TextColor" Value="#cbcbcb" />
                               </VisualState.Setters>
                           </VisualState>
                       </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
```

```
        </buttons:SfButton>
```

Clicked - це подія що виникає коли користувач натискає та відпускає button за допомогою пальця чи мишки.

<buttons:SfButton Clicked="OnLogin"

</buttons:SfButton> - ось так ми додали кнопку, тепер їй потрібні координати.

RelativeLayout.XConstraint = "{ConstraintExpression Type=RelativeToParent, Property=Width, Factor=0.5,Constant=-150}" – це координата X.

Property=Width значить що X дорівнює ширині екрану.

Factor=0.5 - так X ми ділимо навпіл тобто X = width\2

Constant=-150 віднімає від X 150

Тобто: X = width\2 – 150 - так ми розмістили кнопку X по центру екрана.

RelativeLayout.YConstraint = "{ConstraintExpression Type=RelativeToParent, Property=Y, Constant=270}"

По Y ми задали значення 270.

Ширина та висота кнопки:

HeightRequest="65"

WidthRequest="300"

Clicked="OnLogin" – це функція що буде визиватись при натисканні.

Далі йде створення анімації стану кнопки:

<Setter Property="TextColor" Value="White" /> в нормальному стані текст буде білого кольору.

<Setter Property="TextColor" Value="#cbcbcb" /> а при натисканні буде змінюватись колір на #cbcbcb.

Слід відмітити що таким же чином ми створювали кнопку «Реєстрація».

Для використання даного екрану створено код на мові C#

```
Namespace Project.Pages
{
    public partial class StartPage : ContentPage
    {
        public StartPage()
        {
            NavigationPage.SetHasNavigationBar(this, false);
```

```
            InitializeComponent();
        }

        private void OnLogin(object sender, EventArgs e)
        {
            Navigation.PushAsync(new LoginPage(), true);
        }
        private void OnRegistration(object sender, EventArgs e)
        {
            Navigation.PushAsync(new UserRolePage(), true);
        }
    }
    }
```

NavigationPage.SetHasNavigationBar(this, false); - прибираємо навігаційну панель;

InitializeComponent(); - ініціюємо UI елементи описані в StartPage.xaml.

private void OnLogin(object sender, EventArgs e) – подія натиску кнопки вхід.

Navigation.PushAsync(new LoginPage(), true); - додали в масив NavigationPage наступний екран LoginPage та перейшли на нього.

private void OnRegistration(object sender, EventArgs e) - подія натиску кнопки реєстрація.

Navigation.PushAsync(new LoginPage(), true);- додали в масив NavigationPage наступний екран UserRolePage та перейшли на нього.

Нижче зображено вигляд даного екрану на реальному пристрої (Asus ZenFone Max Plus, версія Android - 7.0).

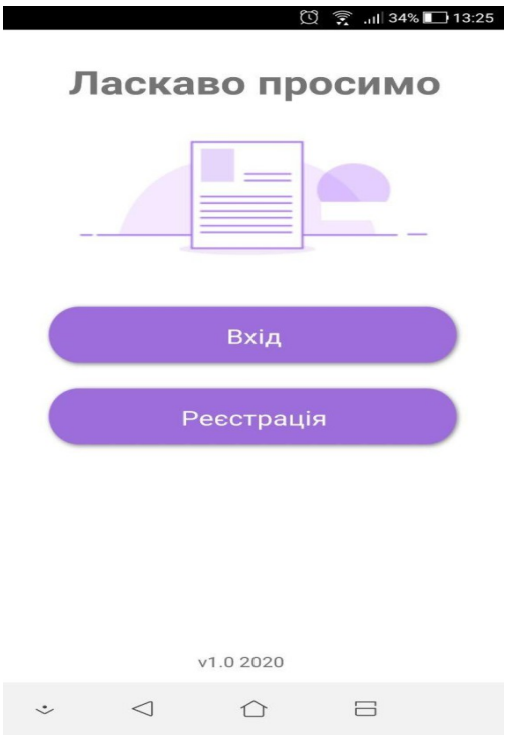


Рис. 3.2 Перша сторінка додатку

2. *UserRolePage*

Якщо користувач натискає на кнопку «Реєстрація» він потрапляє до вікна вибору ролі, в якому необхідно вказати ким являється юзер викладачем чи студентом. Нижче наведено приклад створення ролі «студент»:

```
private void OnSelectStudent(object sender, EventArgs e)
        {
            User user = new User();
            user.role = User.student;

             Navigation.PushAsync(new RegistrationPage(user), true);
              }
```

Спочатку створюємо об’єкт «User» і присвоюємо йому роль «Student». Потім переходимо на наступний екран «RegistrationPage» і передаємо до нього об’єкт «User».

Серед нового функціоналу що з’являється на даному екрані слід відмітити можливість повернення назад натисканням відповідної кнопки. Створення події натиску на кнопку повернення до попереднього екрану.

```
private void OnBack(object sender, EventArgs e)
        {
```

```
            Navigation.PopAsync(true);
        }
```

Вигляд даного екрану наведено нижче:

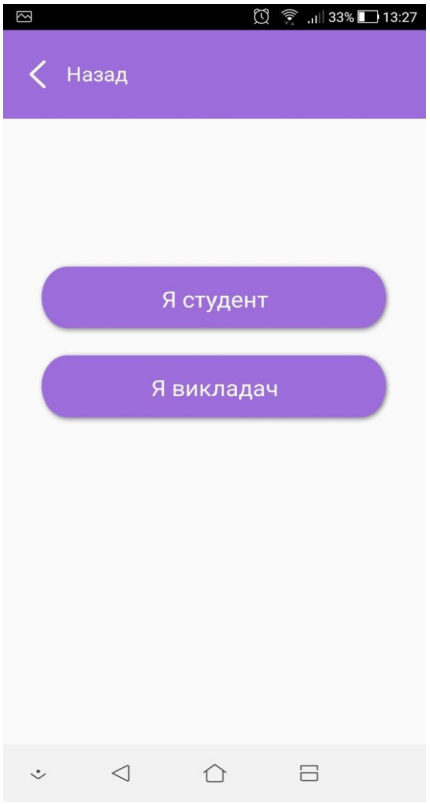


Рис. 3.3 Екран вибору ролі

3. RegistrationPage

Цей екран створений для того щоб користувач міг зареєструватись в системі. Він з'являється після того як обирається відповідна роль.

За допомогою цієї функції ми приймаємо об'єкт User який ми передали з екрану UserRolePage.

```
this.user = user;
```

Далі перевіряємо роль користувача та в titleLabel додаємо відповідний текст. Якщо обрана роль студента:

```
if (user.role == User.student)
            {
                titleLabel.Text = "Реєстрація студента";
            }
```

Якщо ж обрали роль викладача то ми прибираємо кнопку вибору групи та підіймаємо усі UI елементи по осі У

```
if (user.role == User.teacher)
            {
                titleLabel.Text = "Реєстрація викладача";
```

```
            bunchPancakeView.HeightRequest = 0;
            emailErrorLabel.TranslationY = -55;
            emailPancakeView.TranslationY = -55;
            passwordErrorLabel.TranslationY = -55;
            passwordPancakeView.TranslationY = -55;
        }
```

Далі відбувається подія заповнення даних про користувача. Студент повинен вказати обрати зі списку свою групу та факультет викладачу ж потрібно обрати лише факультет та вказати ПІБ, email та пароль. Вікна для вибору груп та факультету створені в екрані «FacultyBunchListPopupPage» та представлені у вигляді випадного списку за допомогою бібліотеки «Rg.Plugins.Popup». Сам же екран створюється та зберігається в папці PopupPages наступним образом:

```
public partial class FacultyBunchListPopupPage : PopupPage
```

Даний клас приймає: registrationPageInterface, USER та чи буде даний екран списком із груп чи факультетів, ці данні передаються з екрану RegistrationPage.

```
        this.isBunch = isBunch;
        this.isFaculty = isFaculty;
        this.user = user;
        this.registrationPageInterface = registrationPageInterface;
        this.faculty = faculty;
```

Далі ми отримуємо дані з серверу та додаємо їх у колекцію facultyArrayList. Також додаємо до ListView усі об’єкти Faculty з колекції facultyArrayList

```
if (await HTTPRequests.GetInstance().getFacultysAndBunchs())
            {
                errorLabel.IsVisible = false;
                listView.ItemsSource =
HTTPRequests.GetInstance().facultyArrayList;
```

Аналогічно відбувається і для груп лише використовуємо змінну «bunch». Далі якщо користувач обере один з факультетів чи групу відбувається присвоєння йому цього значення в об’єкті User. Разом з цим відбувається оновлення кнопки facultyButton на екрані RegistrationPage, за допомогою інтерфейсу registrationPageInterface який ми передали з екрану RegistrationPage:

```
if (isFaculty)
            {
                Faculty selectedFaculty = (Faculty)e.SelectedItem;
```

```
            user.faculty = selectedFaculty;
            registrationPageInterface.refreshUserFaculty();
        }
```

На рисунку показано випадний екран зі списком вибору груп:

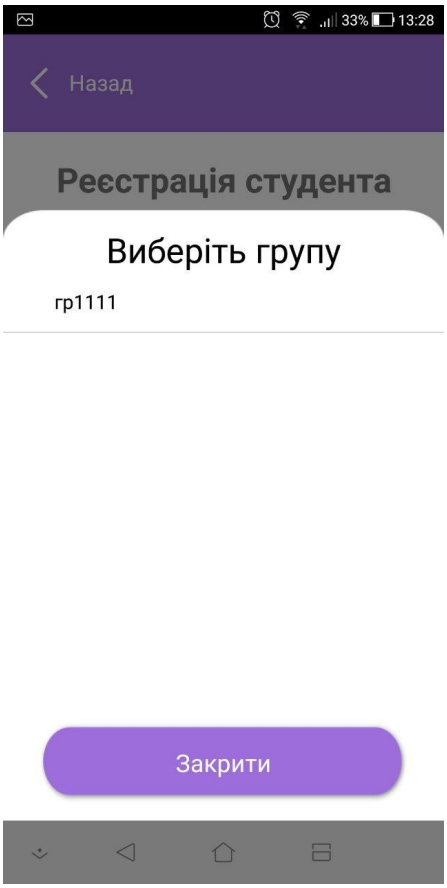


Рис. 3.4 Екран вибору групи

Якщо користувач натискає кнопку «Закрити» йде виконання сценарію повернення на попередню сторінку RegistrationPage:

```
private async void OnClose(object sender, EventArgs e)
        {
            await PopupNavigation.Instance.PopAsync();
        }
```

Якщо ж користувач введе некоректну інформацію чи не заповнить якесь з полів які являються обов’язковими до заповнення він побачить помилку в тому місці де вона допущена.

На рисунках нижче показані створені екрани для реєстрація студента та викладача відповідно.

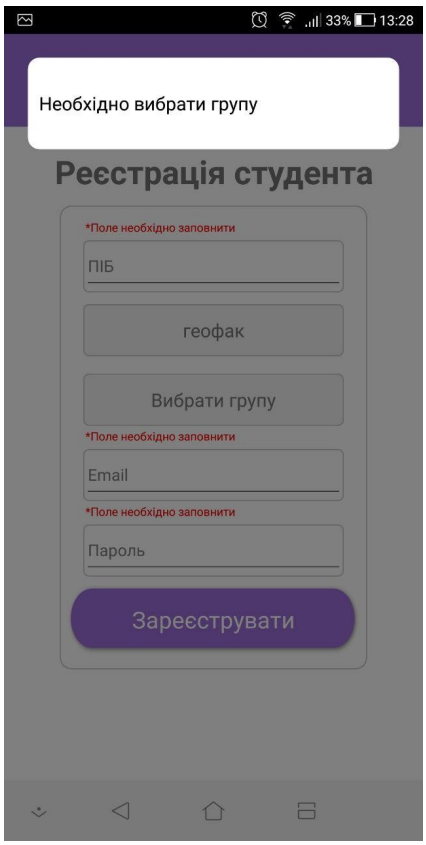


Рис. 3.5 Екран реєстрації студента

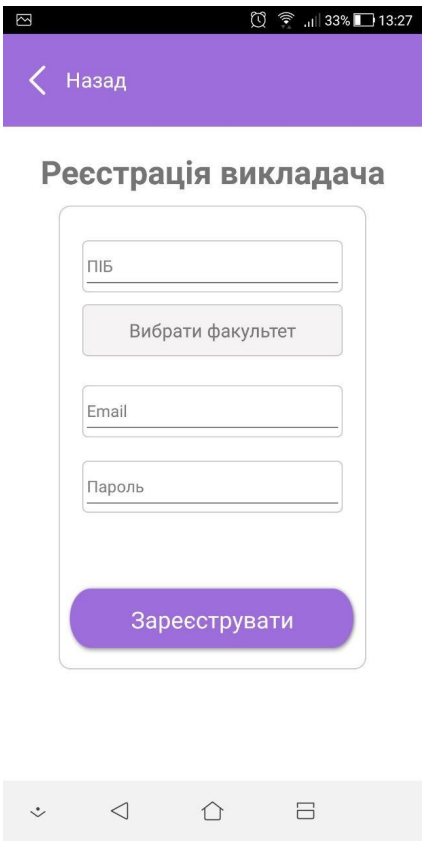


Рис. 3.6 Екран реєстрації для викладача

4. LoginPage

Це екран для входу в додаток для вже зареєстрованих користувачів. Що відбувається після того як користувач нажав кнопку «Вхід»?

По перше створюється об’єкт User та присвоюємо йому змінні email і password. Ті що ввів користувач в поля emailEntry та passwordEntry

```
User user = new User();
user.email = emailEntry.Text;
      user.password = passwordEntry.Text;
```

Далі перевіряємо щоб поля не були пусті

```
if (emailEntry.Text.Length==0)
          {
              await emailErrorLabel.ScaleTo(1, 200);
          }
          else
          {
              await emailErrorLabel.ScaleTo(0, 200);
                }
```

Аналогічно і для паролю:

```
if (passwordEntry.Text.Length == 0)
            {
                await passwordErrorLabel.ScaleTo(1, 200);
            }
            else
            {
                await passwordErrorLabel.ScaleTo(0, 200);
                  }
```

Якщо ці поля не пусті відправляємо запит до серверу:

```
if (user.password.Length>0 && user.email.Length>0)
            {
                      activity.IsVisible = true;
```

Якщо запит повернув null отже зв’язку з сервером немає

```
if (await HTTPRequests.GetInstance().authUser(user) == null)
                 {
```

Якщо метод authUser повернув false показуємо AllertTopDialogPage:

```
                    activity.IsVisible = false;
                    showAllerDialog("Помилка з'єднання з сервером");

                       }
```

Далі перевіряємо чи запит отримав токен якщо так отже введено невірний пароль або такого користувача не існує:

```
else if (user.access_token.Length == 0)
                {
                    activity.IsVisible = false;
                    showAllerDialog("Невірний пароль або email");
                       }
```

Також реалізована функція «запам’ятати мене» що спрощує процес майбутніх сеансів.

```
if (saveSwitch.IsToggled)
                    {
                        Preferences.Set("email", user.email);
                        Preferences.Set("password", user.password);
                    }
                    else
                    {
                        Preferences.Set("email", "");
                        Preferences.Set("password", "");
                           }
```

Якщо все ж данні були введені вірно і такий користувач є на сервері отримуємо його, та переходимо на відповідні сторінки для студента та викладача. Для студента це буде сторінка StudentEvetsPage, а для викладача сторінка TeacherEvetsPage.

```
wait HTTPRequests.GetInstance().getUser(user);
```

```
                    if (user.role == User.student)
                    {
                        await Navigation.PushAsync(new StudentEvetsPage(user),
true);
                    }

                    if (user.role == User.teacher)
                    {
                      await Navigation.PushAsync(new TeacherEvetsPage(user), true);
                           }
```

5. StudentEvetsPage та TeacherEvetsPage

Це екрани на які потравляє студент чи викладач після реєстрації/входу в додаток. Основна логіка для цих сторінок однакова. Обидва користувачі бачать перед собою списки своїх пар (данні про які беруться з серверу). Пари ж відображаються у трьох варіантах це: «Минулі», «Поточні» та «Заплановані». У «Минулих» парах користувачі можуть бачити відповідно свої минулі пари, разом із позначкою про присутність. Відповідно «Поточні» це ті пари що будуть сьогодні а «майбутні» це ті пари що будуть в найближчому майбутньому. Зазначу що обидва користувачі можуть бачили лише свої пари. Студент бачить назву пари, дату і час проведення та викладача і при натисканні на пару може відмітитись на ній. Викладач же бачить все те саме та групи що для яких проводиться дана пара, і при натисканні на пару може побачити списки студентів відсортованих за групами або ж усіх разом. Також у викладача є можливість змінити стан кожного студента (присутній/відсутній) та відправити ці дані на сервер. Нагадаю що відмітитись студенти можуть лише під час проведення пари. В додатку В показаний створений код для зміни стану присутності на парі.

На рисунках нижче відповідно зображені екрани для студентів та викладачів:

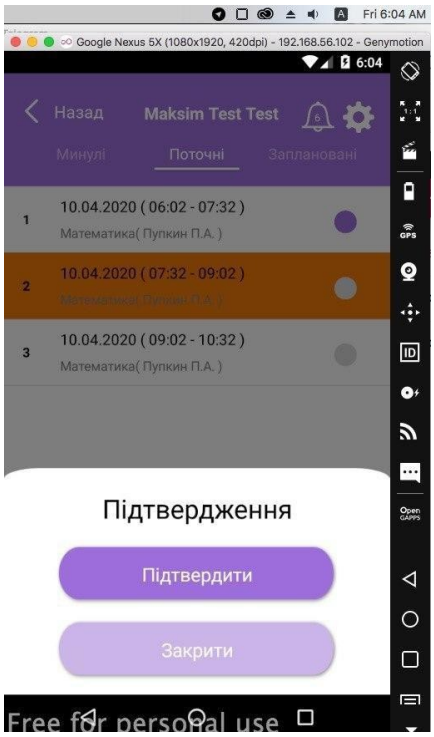


Рис. 3.7 Вигляд меню студента з можливістю підтвердження

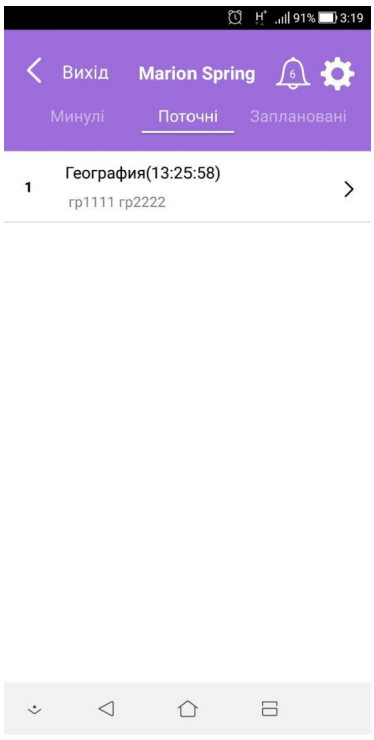


Рис. 3.8 Вигляд головного меню викладача

6. SustensListPage

Це екран для викладача за допомогою якого він може бачити списки студентів і групи. І так що ж відбувається коли викладач натискає на одну із груп в списку?

1. Викликається функція OnBunchSelectedItem:

```
private async void OnBunchSelectedItem(object sender, EventArgs e)
```

2. Отримуємо об’єкт bunch на який ми натиснули:

```
var button = sender as Button;
            Bunch bunch = button.BindingContext as Bunch;
            int number = 0;
```

3. Створюємо нову колекцію для списку студентів:

```
ObservableCollection<User> usersCollection = new
ObservableCollection<User>();
```

4. Далі створюємо цикл для отримання студентів з масиву studentsArrayList:

```
foreach (User student in bunch.studentsArrayList)
            {
                number = number + 1;
                student.number = number;
```

5. Додаємо студента в колекцію usersCollection з масиву bunch.studentsArrayList:

```
usersCollection.Add(student);
            }
```

6. Додаємо колекцію usersCollection до списку для відображення в ListView:

```
listView.ItemsSource = usersCollection;
```

Далі йде функція для оновлення горизонтального списку груп та вертикального списку студентів:

```
   private async void refreshListView()
```

Створюємо дві колекції для груп та студентів:

```
ObservableCollection<Bunch> bunchCollection = new ObservableCollection<Bunch>();
ObservableCollection<User> usersCollection = new ObservableCollection<User>();
```

Далі йде цикл в якому ми отримуємо групи з масиву bunchArrayList:

```
foreach (Bunch bunch in events.bunchArrayList)
            {
```

Всередині цього циклу йде інший на отримання студентів з масиву studentsArrayList:

```
foreach (User student in bunch.studentsArrayList )
                {
                    number = number + 1;
                    student.number = number;
```

Додаємо студентів до колекції:

```
  usersCollection.Add(student);
```

Додаємо групу до колекції

```
}
                bunchCollection.Add(bunch);
}
```

Таким чином викладач може бачити списки груп та студентів конкретних груп які повинні бути присутні на парі.

3.2. Створення запиту для роботи з API

Приклад створення запиту на реєстрацію нового користувача: Функція для авторизації:

Приймає аргумент об’єкта User. Вертає об’єкт User:

```
public async Task<User> authUser(User user)
{
```

Створюємо об’єкт масив List з ім’ям nvc:

```
var nvc = new List<KeyValuePair<string, string>>();
```

Додаємо в масив nvc два об’єкти KeyValuePair (ключ,значення) В даному випадку цей ключ - username, а значення – змінна email об’єкту user.

Це будуть аргументи ПОСТ Http запиту:

```
nvc.Add(new KeyValuePair<string, string>("username", user.email));
nvc.Add(new KeyValuePair<string, string>("password",
user.password));
```

Створюємо request запит HttpRequestMessage:

```
HttpRequestMessage request = new HttpRequestMessage();
```

Вказуємо адрес запиту:

```
request.RequestUri = new Uri(serverURL + "/api/auth/");
```

Вказуємо що це буде ПОСТ запит:

```
request.Method = HttpMethod.Post;
```

Ось тут ми передаємо аргументи для ПОСТ запиту, в даному випадку це email та пароль:

```
request.Content = new FormUrlEncodedContent(nvc);
```

Ось тут ми вказуємо в Headers що сесію після опрацювання запиту можно закрити:

```
request.Headers.Add("Connection", "close");
```

Створюємо об’єкт HttpClient який відправить наш запит HttpRequestMessage серверу:

```
HttpClient client = new HttpClient();
```

Створюємо об’єкт HttpResponseMessage - сюди ми віддамо відповідь від сервера. Робимо його нульовим так як у нас ще нема відповіді.

```
HttpResponseMessage response = null;
```

try{

Відправляємо наш запит request функцією SendAsync на сервер за допомогою класу HttpClient:

response = await client.SendAsync(request);

Якщо все пройшло вдало ми записуємо відповідь в об’єкт response

} catch (Exception e) {

Якщо сервер не відповів в консолі відображуємо причину функцією Console.WriteLine:

Console.WriteLine(e);

Завершуємо роботу функції authUser вертаємо null:

return null;

Перевіряємо якщо відповідь з серверу є:

```
if (response != null)
{
    try
    {
```

Записуємо текст відповіді від серверу в змінну content:

var content = await response.Content.ReadAsStringAsync();

Виводимо до консолі текст відповіді та змінну content:

Console.WriteLine("Auth debug : " + content);

Так як ми знаємо що відповідь у нас це JSON, розпаршуємо текст відповіді в JSON об’єкт за допомогою бібліотеки Newtonsoft.Json.Linq

JObject jObject = JObject.Parse(content);

Витягуємо значення detail та записуємо в змінну detail:

string detail = (string)jObject["detail"];

Перевіряємо щоб detail не було порожнім-

```
if (detail != null)
{
```

якщо detail не порожнє та містить строку No active account found, отже такого користувача нема або пароль введено невірно.

```
            if (detail.Contains("No active account found"))
            {
                user.access_token = "";
            }
        }
        else
        {
```

Якщо detail порожнє значить користувач є і пароль введено правильно. Записуємо в змінну access_token отриманий token access:

```
            user.access_token = (string)jObject["access"];
            user.refresh_token = (string)jObject["refresh"];
        }
```

Завершуємо роботу функції authUser вертаємо об’єкт user:

```
        return user;
    }
    catch (Exception e) }
```

Завершуємо роботу функції authUser вертаємо null. Якщо сервер прислав якиу-небудь невірну відповідь

```
        Console.WriteLine("Auth exeption : " + e);
        return null;
```

ВИСНОВКИ

У ході написання дипломної роботи були виконані такі завдання:

1. Розглянуто та проаналізовано існуючу систему обліку відвідування занять в університеті, яка на сьогоднішній день є застарілою та містить масу недоліків, зокрема можливість обходу системи, довга обробка даних та необхідність заповнення списків вручну.
2. Проаналізовано існуючі сервіси контролю відвідуваності: «BioTime», «MyAttendanceTracker», «Smiles.Школьная карта». Виділено їх недоліки та переваги, основні з яких враховані у створеному додатку.
3. Обґрунтовано функціонал клієнтської частини а саме: інтерфейс додатку простий та зрозумілий, додаток крос-платформний, створено дві ролі «викладач» та «студент», які можуть зайти, переглянути пари та «відмітитись». Слід зазначити, що у процесі реєстрації студент вказує номер групи та факультет, а викладач лише факультет.
4. На основі аналізу засобів для крос-платформної розробки, нами обрано фреймворк Xamarin.Forms.
5. Розроблені необхідні сторінки (екрани), а саме: реєстрація (для студента та викладача), стартова сторінка, вхід, підтвердження, перегляду пар та списку студентів (лише для викладача). Написані запити для роботи з серверною частиною, а саме на отримання переліку груп, списку пар та для підтвердження наявності.

Створений додаток є крос-платформним, а значить може використовуватись на будь якому пристрої в стінах Херсонського державного університету, та за бажанням у будь-якому іншому навчальному закладі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ВНЗ [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу:https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BB%D0%B0%D0%B4_%D0%B2%D0%B8%D1%89%D0%BE%D1%97_%D0%BE%D1%81%D0%B2%D1%96%D1%82%D0%B8

2. Навчальний підрозділ [Електронний ресурс] – Режим доступу до ресурсу:https://pidruchniki.com/86521/menedzhment/strukturni_pidrozdili_vischogo_navchalnogo_zakladu

3. BioTime [Електронний ресурс] – Режим доступу до ресурсу: https://www.biotime.ru/advantages/mobilnye-resheniya.php

4. MyAttendanceTracker [Електронний ресурс] – Режим доступу до ресурсу: https://www.myattendancetracker.com/

5. «Smiles. Школьная карта» [Електронний ресурс] – Режим доступу до ресурсу: http://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:SmileS._%D0%A8%D0%BA%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BA%D0%B0%D1%80%D1%82%D0%B0_%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%88%D0%BA%D0%BE%D0%BB%D1%8B_%D0%B8_%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B9

6. Крос-платформеність [Електронний ресурс] / Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%8

2%D0%BE%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%BD%D1%96%D1%81%D1%82%D1%8C.

7. .Net [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/standard/class-library-overview

8. Xamarin [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Xamarin

9. Что такое Xamarin.Forms? [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin-forms.

10. Добро пожаловать в интегрированную среду разработки Visual Studio [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019.

11. Документація по Xamarine [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/

12. IT Shark Mobile Xamarin [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/it-shark-pro/mobile-xamarin.

13. Диспетчер визуальных состояний Xamarin. Forms [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/user-interface/visual-state-manager.

14. Описание среды разработки Microsoft Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://studbooks.net/2258619/informatika/opisanie_sredy_razrabotki_microsoft_visual_studio.

15. Xamarine.Forms [Електронний ресурс]. – 1201. – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/

16. Json.NET [Електронний ресурс] – Режим доступу до ресурсу: https://www.newtonsoft.com/json.

17. Sequence diagram [Електронний ресурс] – Режим доступу до ресурсу: https://flexberry.github.io/ru/fd_sequence-diagram.html.

18. UML. Диаграммы последовательности [Електронний ресурс] – Режим доступу до ресурсу: https://pro-prof.com/archives/2769.

19. XAML [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/XAML.

20. Система управління вищим навчальним закладом [Електронний ресурс] – Режим доступу до ресурсу: http://www.info-library.com.ua/books-text-4210.html.

21. Создание и настройка проєкта в Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/2.1.php.

22. Button в Xamarin.Forms [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/get-started/tutorials/button/?tabs=vswin.

23. Достоинства и недостатки Xamarin [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/microsoft/blog/415833/.

24. Кнопка Xamarin.Forms [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/user-interface/button.

25. Shelehin A. Подробно о Xamarin [Електронний ресурс] / Andrey Shelehin – Режим доступу до ресурсу: https://habr.com/ru/post/188130/.

26. Готовим Xamarin.Forms: настройка окружения и первые шаги [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/microsoft/blog/303630/.

27. Создание одностраничного приложения Xamarin.Forms [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/xamarin/get-started/quickstarts/single-page?pivots=windows.

28. Простым языком об HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/215117/.

29. Xamarin.Essentials [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/xamarin/essentials/.

30. Popup Page Plugin for Xamarin Forms [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/rotorgames/Rg.Plugins.Popup.

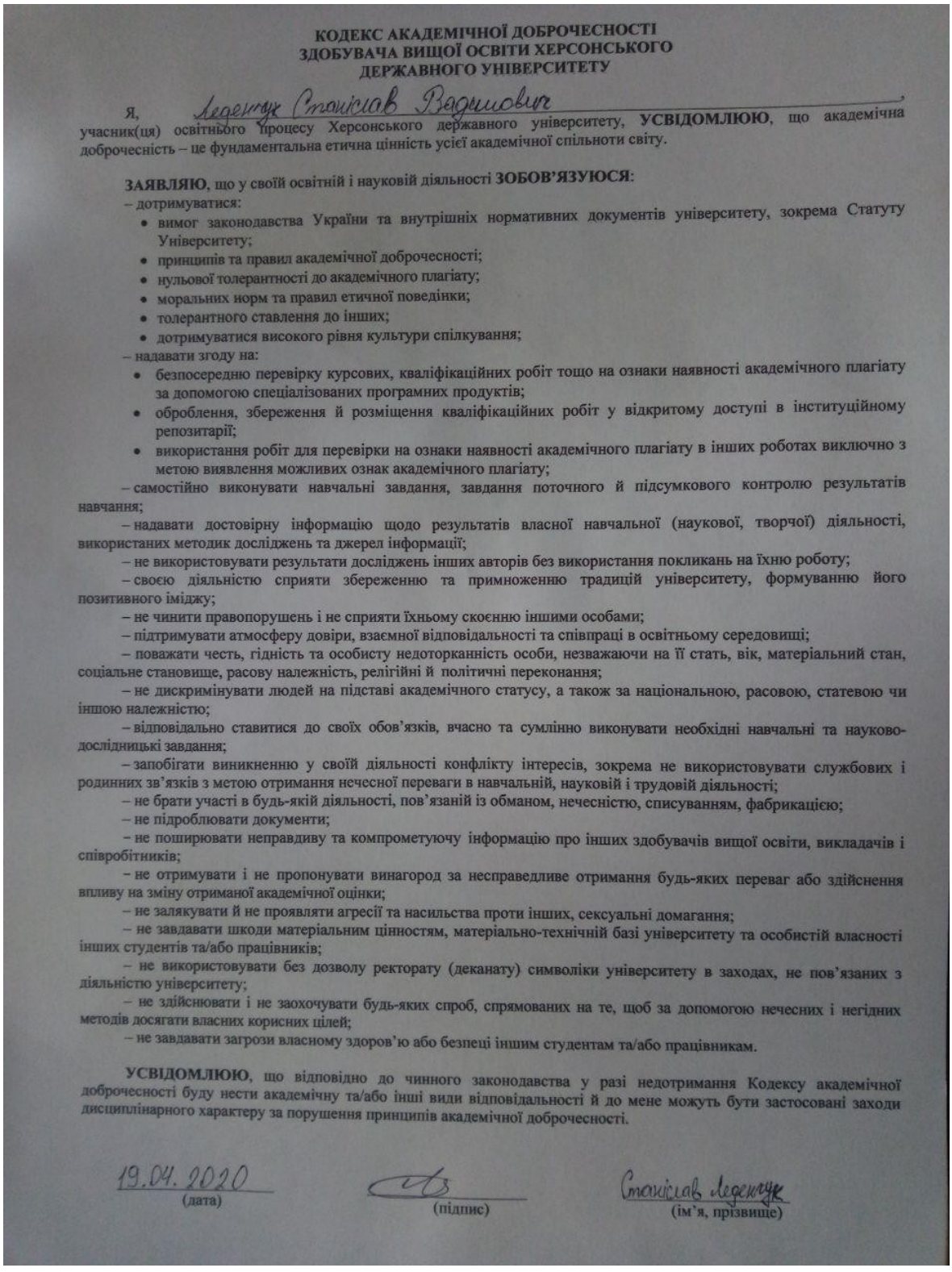
ДОДАТКИ

Додаток А

```
public async Task<bool> changeStatus(Mark mark,User user)
        {


            var nvc = new List<KeyValuePair<string, string>>();

            nvc.Add(new KeyValuePair<string, string>("is_attended", "true"));


            HttpRequestMessage request = new HttpRequestMessage();
            request.RequestUri = new Uri(serverURL + "/api/mark/"+mark.ID);
            request.Method = HttpMethod.Put;
            request.Content = new FormUrlEncodedContent(nvc);
            request.Headers.Add("Connection", "close");

            HttpClient client = new HttpClient();
            client.DefaultRequestHeaders.Add("Authorization", "Bearer " +
user.access_token);
            HttpResponseMessage response = null;
            Console.WriteLine("response : " + serverURL + "/api/mark/" +
mark.ID);
            try
            {
               response = await client.SendAsync(request);
            }
            catch (Exception e)
            {
               Console.WriteLine(e);
               return false;
            }
            if (response != null)
            {
               Console.WriteLine("response : " + response);
               try
               {
                  var content = await response.Content.ReadAsStringAsync();
                  Console.WriteLine("changeStatus : " + content);

                  return true;
               }
```

```
            catch (Exception e)
            {
                Console.WriteLine("changeStatus exeption : " + e);
                return false;
            }
        }

        return false;
    }


}
}
```

Додаток Б



КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Леденюк Станіслав Вадимович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

19.04.2020 (дата) (підпис) Станіслав Леденюк (ім'я, прізвище)

Рис. Б.1 Кодекс академічної доброчесності