

Програмний проект для роботи з даними успішності студентів

Гайлунь Владислав
Шевченко Родіон
Мамонт Владислав
Бондаренко Іван
Третьак Александр

КНТ-223



ПОСТАНОВКА ЗАДАЧІ ТА МЕТИ



- Мета: Розробити програмний проект який дозволяє централізовано зберігати, аналізувати та управляти даними про студентів. Портал забезпечує доступ до інформації про факультети, групи, академічні бали, успішність та інші показники, створюючи зручний інструмент для адміністрації, викладачів і студентів. Основна мета – автоматизація обробки даних, спрощення доступу до інформації та підвищення ефективності управління навчальним процесом.

- Задачі:
 - Забезпечити збереження даних про студентів, факультети, групи та їхню успішність у єдиній базі даних.

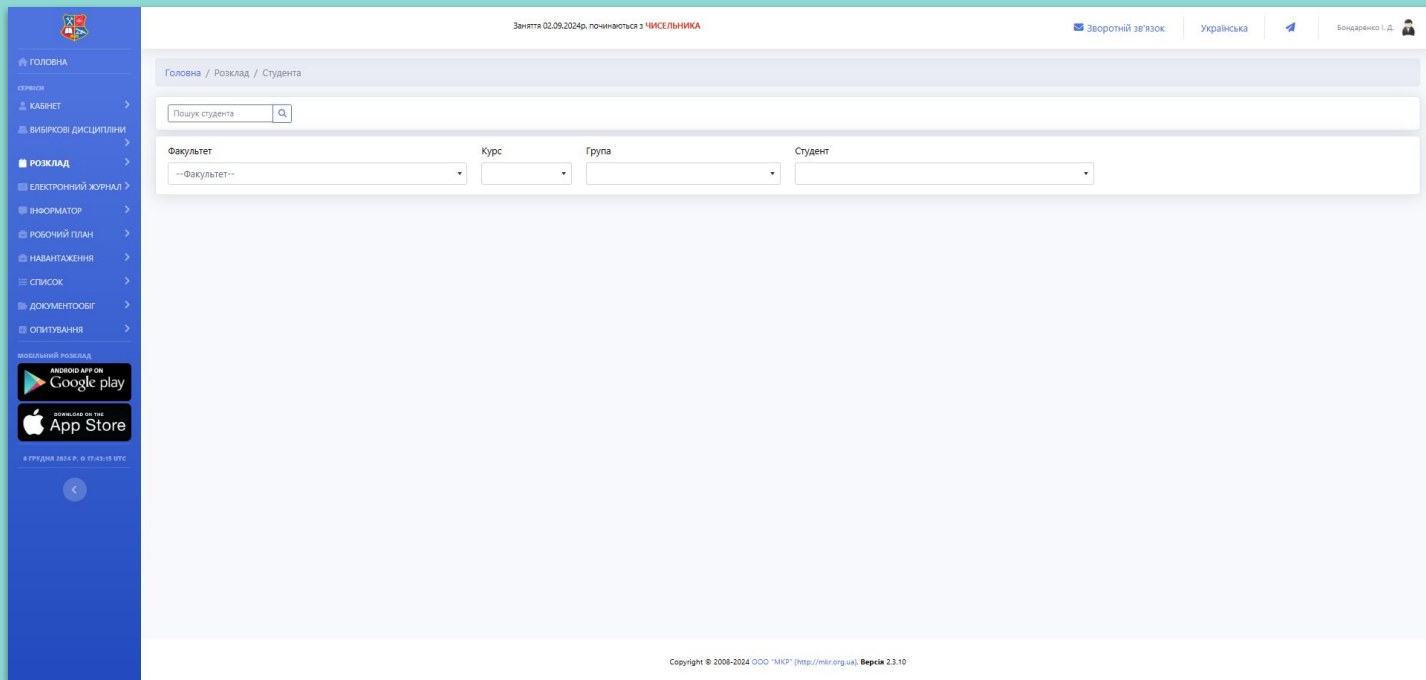
 - Розробити інтерфейс для перегляду інформації про студентів, їхні академічні бали та досягнення.

 - Реалізувати функцію пошуку і фільтрації студентів за факультетами, групами та іншими критеріями.

ІСНУЮЧІ АНАЛОГІ

<https://portal.zp.edu.ua/time-table/student> – інтернет портал університету "Запорізьська Політехніка".

Як на першому, так і на другому порталі робити пошук студента за факультетом, групою, спеціальністю, тощо.



The screenshot displays the web interface of the Zaporizhzhia Polytechnic portal. On the left is a blue sidebar with a navigation menu. The main content area is white and contains a search form for students. At the top of the page, there is a status bar with the date and time, a language selector, and a user profile icon.

Navigation Menu (Left Sidebar):

- ГОЛОВНА
- СЕРВИС
- КАБІНЕТ
- ВІСІРКОВІ ДИСЦИПЛІНИ
- РОЗКЛАД
- ЕЛЕКТРОННИЙ ЖУРНАЛ
- ІНФОРМАТОР
- РОБОЧИЙ ПЛАН
- НАВАНТАЖЕННЯ
- СПИСОК
- ДОКУМЕНТОБІГ
- ОПІТУВАННЯ
- Мобільний застосунок
- ANDROID APP ON Google play
- APPLE APP STORE
- © ГРУДЕНЬ 2024 Р. 0:17:43-18 UTC

Page Header:

Заняття 02.09.2024р. починається з 11:55/ЛІНІЯ

Зворотній зв'язок | Українська | Бондаренко І.Д.

Search Form:

Головна / Розклад / Студента

Пошук студента

Факультет: --Факультет-- Курс: Група: Студент:

ІСНУЮЧІ АНАЛОГІ

<https://nz.ua/> - електронний щоденник, який дає можливість дивитися розклад, бали, предмети студента та т.д.

Можливості



Звіти



Щоденник



Відедуваність



Домашні завдання



Розклад



Дистанційне навчання



Нова Українська школа



Діаграми

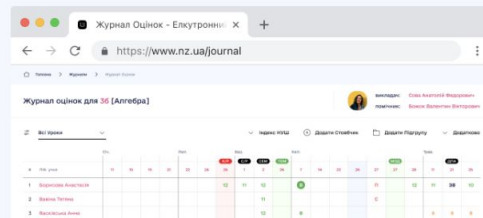
Як це виглядає

Журнали

Надають можливість зручного створення уроків, виставлення оцінок та аналізу успішності учнів, класів, школи.

Щоденники

Звіти



СЕРЕДОВИЩЕ ПРОЕКТУВАННЯ ТА РОЗРОБКИ ПРОЕКТУ

- ❖ Середовище проектування та розробки – Microsoft Visual Studio.
- ❖ Причини вибору:
 - зручність та зрозумілість внутрішнього інтерфейсу.
 - можливість інтегрування графічного інтерфейсу.
 - можливість легко оновленим проектом серед усіх колег по проекту.



ОСНОВНІ ФУНКЦІЇ

1. Збереження та обробка даних студентів.
2. Зручний пошук та фільтрація інформації.
3. Аналіз успішності.
4. Управління групами та факультетами.



РОЗПОДІЛ РОЛЕЙ У КОМАНДІ

Гайлунь Владислав:

→ Розробка інтерфейсу



Шевченко Родіон:

→ Розробка класів і архітектури



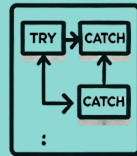
Мамонт Владислав:

→ Розробка та робота з базою даних



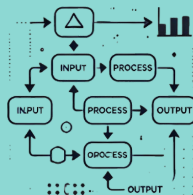
Бондаренко Іван:

→ Обробка виняткових ситуацій

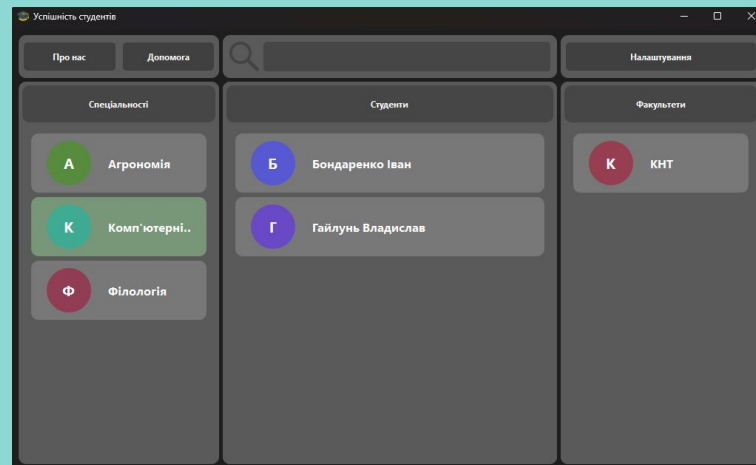
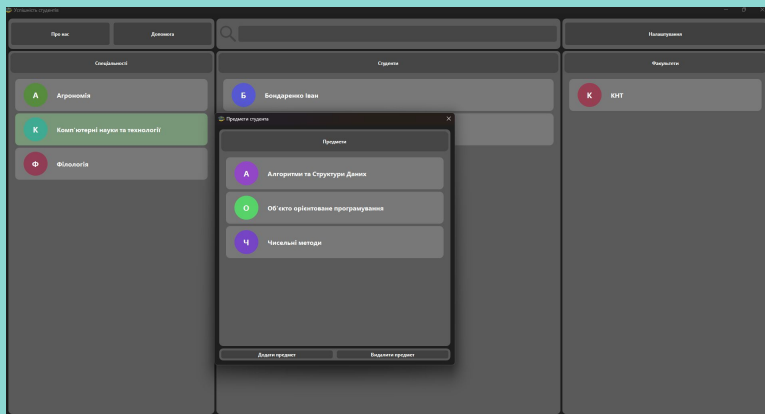
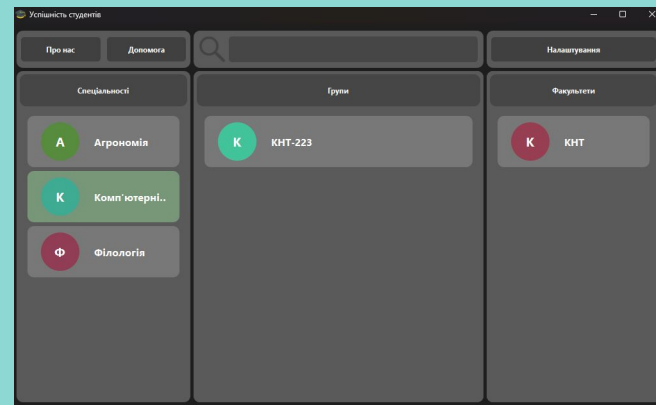
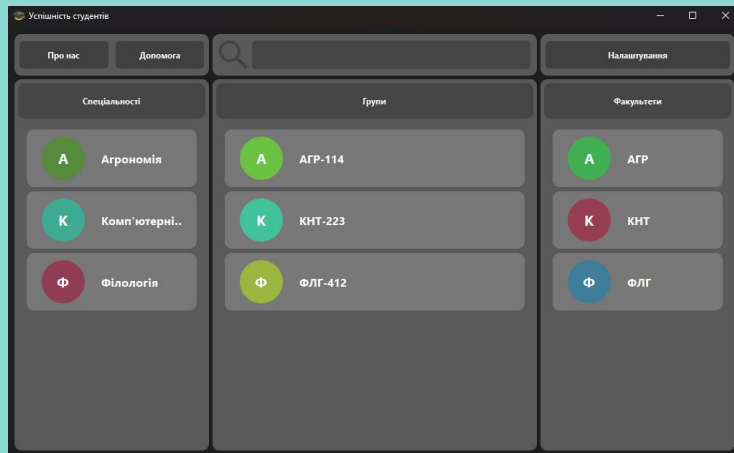


Третяк Александр:

→ Реалізація алгоритмів



ПРОТОТИП





ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



РОЗРОБЛЕНІ КЛАСИ АЛГОРИТМІВ

Клас	Опис класу
SubjectInfo	Клас відповідає за зберігання у собі інформації щодо предметів та балів.
StudInfo	Клас містить у собі дані пов'язані зі студентами, а саме: повне ім'я студента, його спеціальність, групу, факультет, та предмети, які він вивчає.
StudentBlock	Клас, який відповідає за збереження масиву студентів та всієї інформації про них.

РОЗРОБЛЕНІ КЛАСИ ІНТЕРФЕЙСУ

Клас	Опис класу
<code>MainWindow_C</code>	Головний клас який відповідає за відображення всього вікна програми та взаємодії з нею.
<code>configBlock</code>	Клас для відображення та роботи з блоками. Має методи для відображення та налаштування блоків, параметрично задається кількість блоків які будуть виведені у віджет, їх стилі, та обробки натиску, а також заголовок та сортування блоків.
<code>blockWidget</code>	Є допоміжним класом <code>configBlock</code> тому що саме даний клас має стилі для блока(<code>iv</code>). Має вбудований клас <code>circleQWidget</code> який потрібен для кола з певною літерою на блоках.

РОЗРОБЛЕНІ КЛАСИ ІНТЕРФЕЙСУ



Клас	Опис класу
<code>smartText</code>	Є допоміжним класом <code>blockWidget</code> (а також користувацьким) для зменшення тексту в блоках. Якщо батьківський блок менший за текст, ті літери, які не поміщаються у віджет будуть приховані а в кінці батька буде три крапки (сигналізуючи про те, що текст зміщено, або він занадто довгий).
<code>WarningDialog</code>	Користувацький клас розроблений для відображення помилок/повідомлень/попереджень. По суті це діалогове вікно, яке можна кастомізувати задаючи текст, колір, та інші мілкі деталі які приємно використовувати.

РОЗРОБЛЕНІ КЛАСИ ІНТЕРФЕЙСУ



Клас	Опис класу
<code>SmallMessage_C</code>	Маленьке повідомлення яке може містити в собі текст та має параметричний колір, а також може розміщуватись у різних частинах батьківського віджету відносно заданих параметрів.
<code>counterTimer</code>	клас для відслідковування кількості таймерів головного вікна (яким є <code>MainWindow_C</code>) це треба для коректної обробки натисків/сортування і інших речей на блоки.
<code>smartTextBase</code>	Абстрактний клас, який треба для того, щоб похідні класи мали змогу змінювати поведінку розумного тексту.



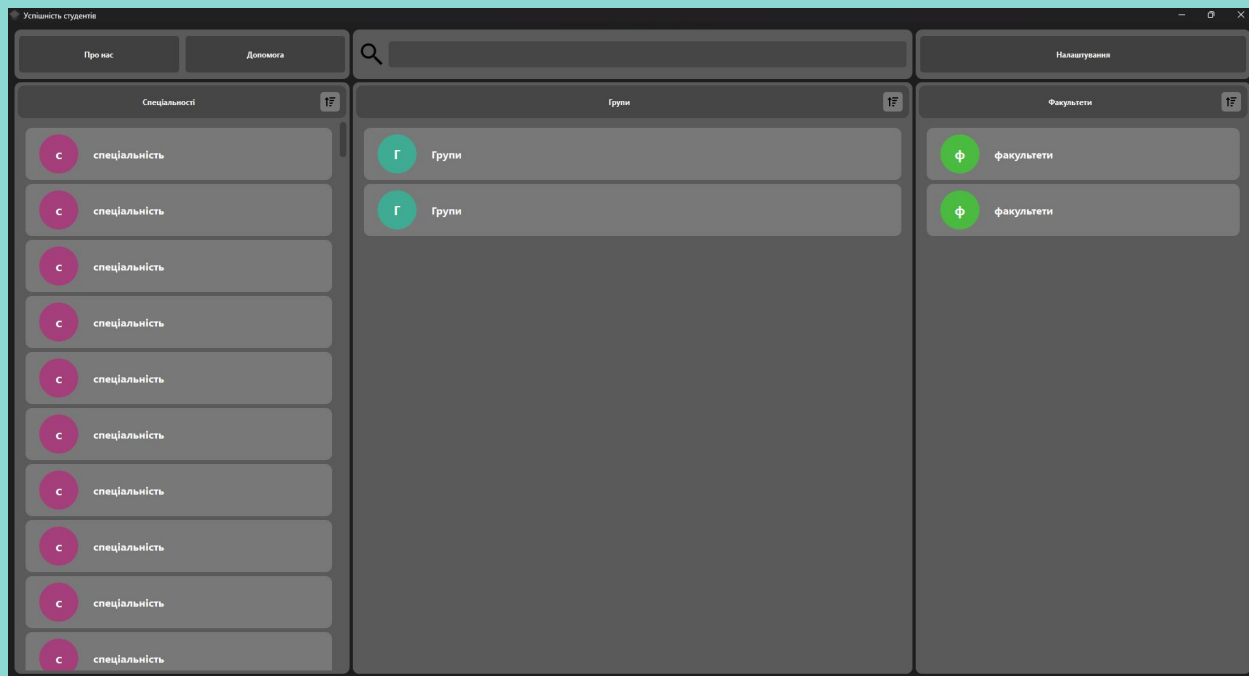
КЛАСС ДЛЯ ОБРОБКИ ВИНЯТКОВИХ СИТУАЦІЙ

Клас	Опис класу
Ex	Клас забезпечує збереження коду помилки та надає доступ до нього через метод отримання.

РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

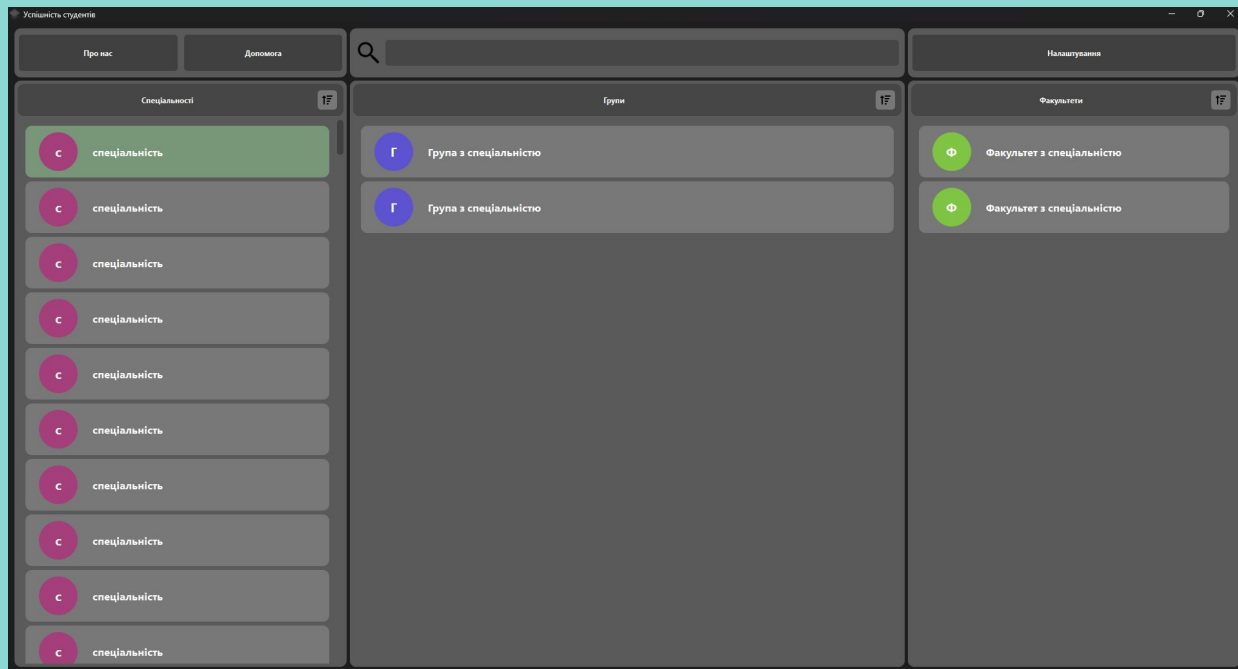


- Основна сторінка програми, яка має багато вкладок: три основні, які відповідають за спеціальності, групи та факультети. Та ще декілька допоміжних, таких як: про нас, допомога, налаштування, пошук.



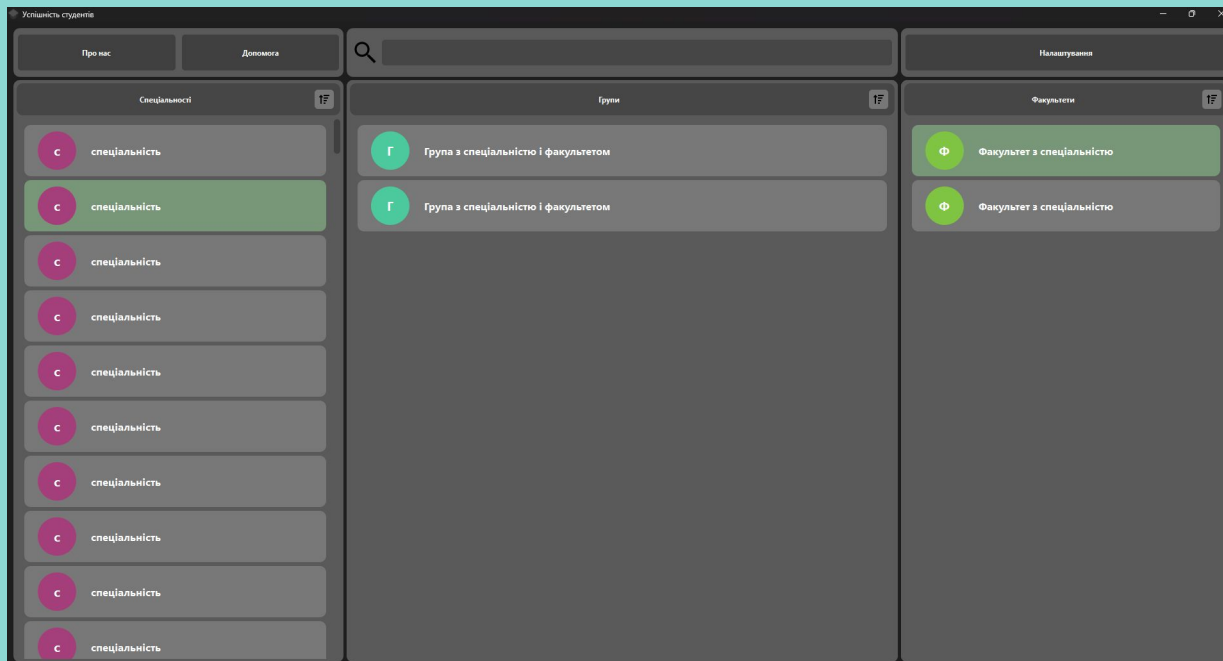
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- При виборі спеціальності центральне вікно з групами змінює свій вигляд та починає відображати тільки ті групи, які належать до обраної спеціальності.



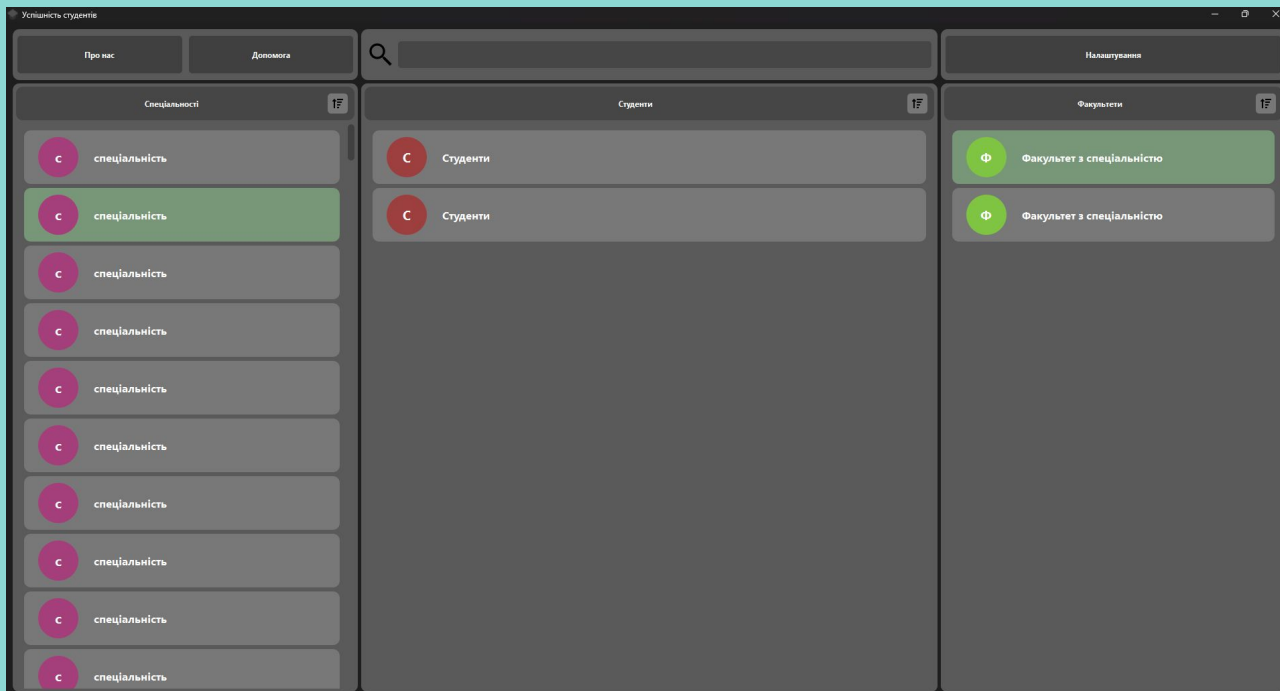
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- При виборі спеціальності, а також факультету, у центральному вікні ми зможемо побачити всі групи, які належать до вибраного факультету та обраної спеціальності.



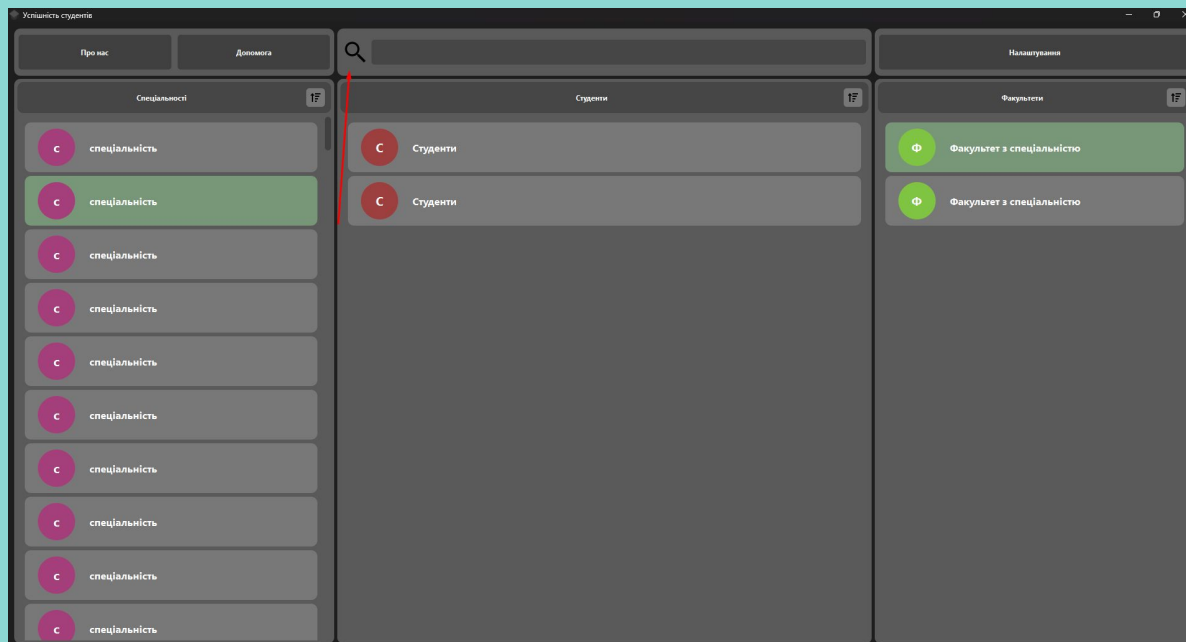
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- Якщо вибрати групу, то центральне вікно змінюється на вікно відображення студентів, які навчаються у цій групі.



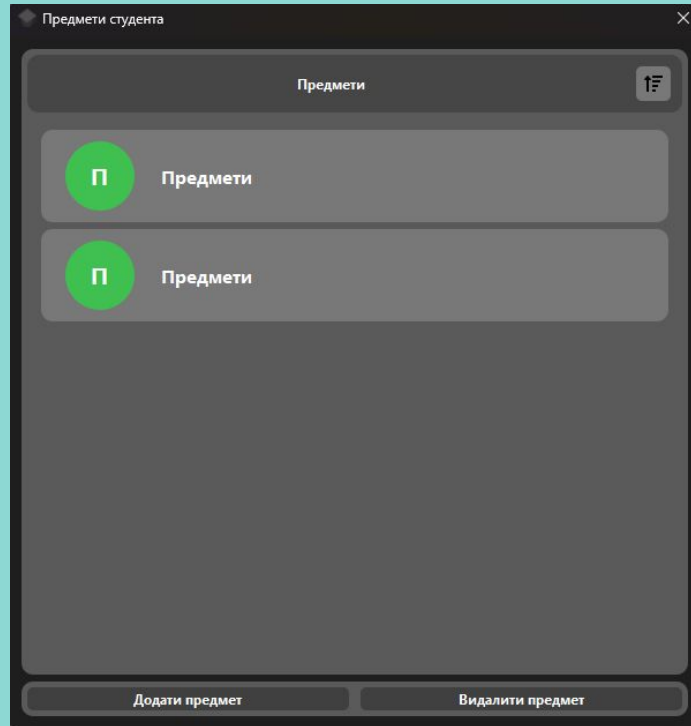
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

→ Також по центру зверху ми маємо строку пошуку студента. Пошук студента відбувається за специфічним стандартом вказуючи спеціальність, факультет, групу та піб студента.



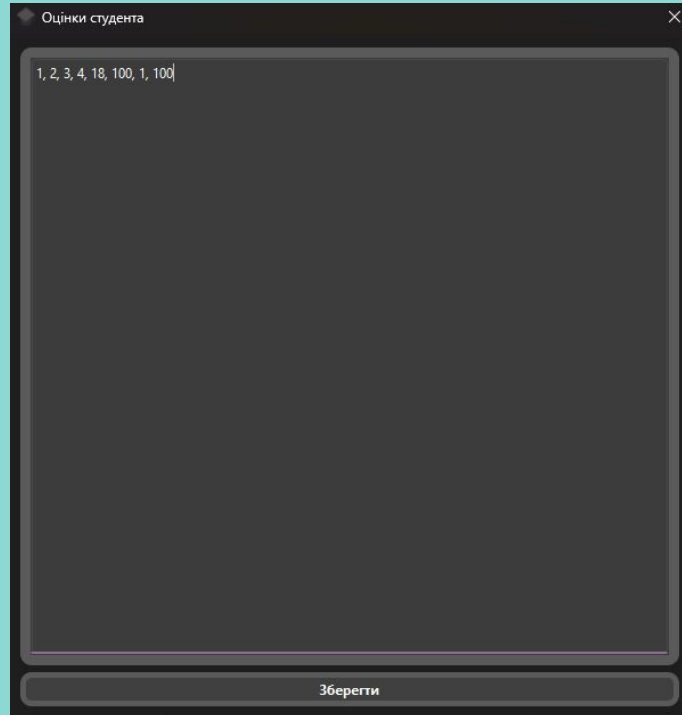
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- Вибираючи студента, на екрані отримуємо меню, яке показує перелік предметів, які вивчає вибраний студент.



РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

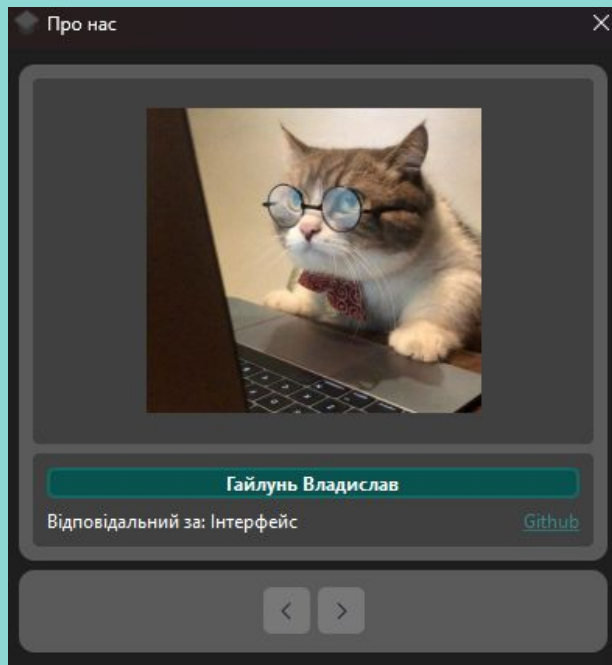
- Вибравши будь який предмет з переліку, ми потрапляємо до меню редагування балів з цього предмета у вибраного раніше студента. Бали виставляються в рядок, через кому.



The screenshot shows a window titled "Оцінки студента" (Student Grades) with a close button (X) in the top right corner. Inside the window, there is a text input field containing the sequence "1, 2, 3, 4, 18, 100, 1, 100". Below the input field is a large, empty rectangular area, likely for a list of subjects or students. At the bottom of the window, there is a button labeled "Зберегти" (Save).

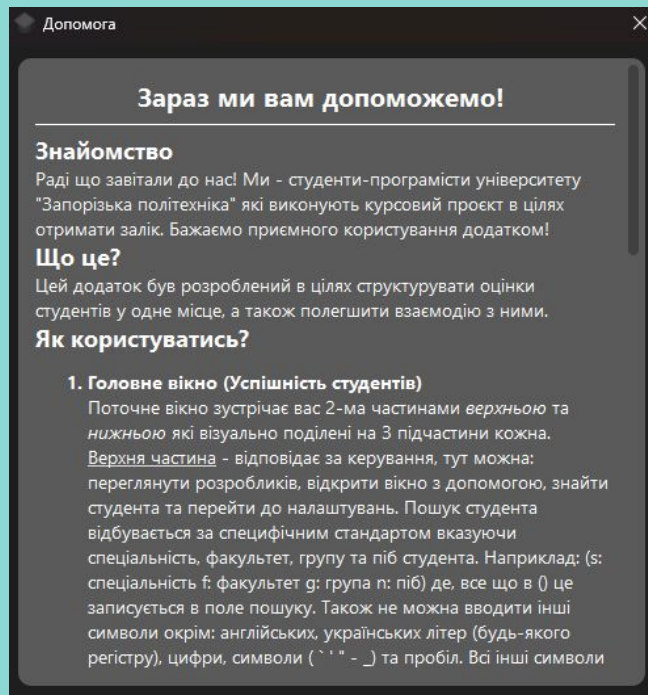
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- Натиснувши на вкладку "Про нас" у верхньому лівому кутку сторінки, ми побачимо вікно з інформацією про кожного учасника групи: його фотографію, ім'я, його роль у проекті та посилання на GitHub



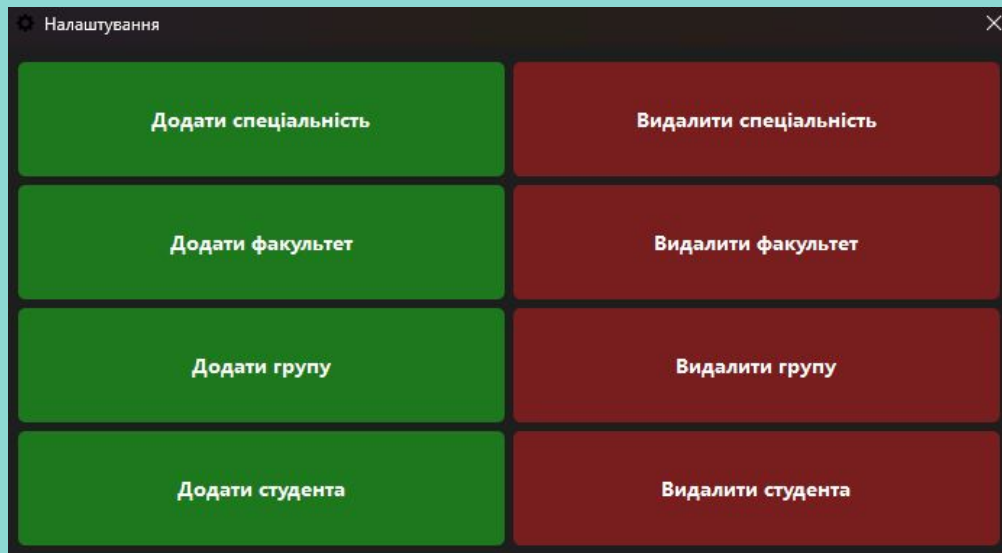
РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

→ У тому ж верхньому лівому кутку, але трохи правіше знаходиться вкладка "Допомога". Натиснувши її, Ви побачите детальну інструкцію для користувачів : Як користуватися програмою.



РОЗРОБЛЕНИЙ ІНТЕРФЕЙС

- У верхньому правому кутку ми бачимо меню "Налаштування". При натисканні туди, ми побачимо вікно, завдяки якому зможемо налаштувати наявність студентів, факультетів, груп та спеціальностей.



РОЗРОБЛЕНИЙ ІНТЕРФЕЙС



→ Ось декілька прикладів роботи меню "Налаштування" :

➤ Додавання факультету:

➤ Додавання студента:

➤ Видалення студента:

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

- ➔ База даних, зроблена на основі SQLite, у якій реалізовано збереження всіх даних у проєкті: предметів, факультетів, студентів, їх балів, т.д.

```
Code - databd.cpp
databd.cpp
1 #include "mainwindow.h"
2
3 void CreateTables() {
4     QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
5     db.setDatabaseName("myDatabase.db");
6     // double tables()
7     if (db.open()) {
8         qDebug() << "Database is open.";
9         QSqlQuery query;
10        query.exec("PRAGMA foreign_keys = ON;");
11        // БД для спеціальностей
12        if (query.exec(
13            "CREATE TABLE IF NOT EXISTS specialty ("
14            "id INTEGER PRIMARY KEY,"
15            "specialty TEXT);")
16        ) {
17            qDebug() << "Error creating specialty table:" << query.lastError().text();
18        }
19
20        // БД для факультетів
21        if (query.exec(
22            "CREATE TABLE IF NOT EXISTS faculty ("
23            "id INTEGER PRIMARY KEY,"
24            "specialty TEXT,"
25            "faculty TEXT);")
26        ) {
27            qDebug() << "Error creating faculty table:" << query.lastError().text();
28        }
29
30        // БД для груп
31        if (query.exec(
32            "CREATE TABLE IF NOT EXISTS class_group ("
33            "id INTEGER PRIMARY KEY,"
34            "specialty TEXT,"
35            "faculty TEXT,"
36            "class_group INTEGER);")
37        ) {
38            qDebug() << "Error creating class_group table:" << query.lastError().text();
39        }
40
41        // БД для студентів
42        if (query.exec(
43            "CREATE TABLE IF NOT EXISTS students ("
44            "id INTEGER PRIMARY KEY,"
45            "name TEXT,"
46            "age INTEGER,"
47            "faculty TEXT,"
48            "specialty TEXT,"
49            "class_group INTEGER);")
50        ) {
51            qDebug() << "Error creating students table:" << query.lastError().text();
52        }
53
54        // БД для предметів
55    }
```

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

Функція `CreateTables` - відповідає за створення баз даних для зберігання спеціальностей, факультетів, груп, студентів, предметів та оцінок.

```
void CreateTables() {
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("mydatabase.db");
    // dropAllTables();
    if (db.open()) {
        cqcout << "Database is open.";
        QSqlQuery query;
        query.exec("PRAGMA foreign_keys = ON;");
        // БД для спеціальностей
        if (!query.exec(
            "CREATE TABLE IF NOT EXISTS specialty ("
            "id INTEGER PRIMARY KEY,"
            "specialty TEXT);")
        ) {
            cqcout << "Error creating specialty table:" << query.lastError().text();
        }

        // БД для факультетів
        if (!query.exec(
            "CREATE TABLE IF NOT EXISTS faculty ("
            "id INTEGER PRIMARY KEY,"
            "specialty TEXT,"
            "faculty TEXT);")
        ) {
            cqcout << "Error creating faculty table:" << query.lastError().text();
        }

        // БД для груп
        if (!query.exec(
            "CREATE TABLE IF NOT EXISTS class_group ("
            "id INTEGER PRIMARY KEY,"
            "specialty TEXT,"
            "faculty TEXT,"
            "class_group INTEGER);")
        ) {
            cqcout << "Error creating class_group table:" << query.lastError().text();
        }
    }
}
```

```
// БД для студентів
if (!query.exec(
    "CREATE TABLE IF NOT EXISTS students ("
    "id INTEGER PRIMARY KEY,"
    "name TEXT,"
    "age INTEGER,"
    "faculty TEXT,"
    "specialty TEXT,"
    "class_group INTEGER);")
) {
    cqcout << "Error creating students table:" << query.lastError().text();
}

// БД для предметів
if (!query.exec(
    "CREATE TABLE IF NOT EXISTS predmet ("
    "id INTEGER PRIMARY KEY,"
    "predmet TEXT,"
    "student_id TEXT,"
    "FOREIGN KEY(student_id) REFERENCES students(id)"
    " ON DELETE CASCADE" // Видалення предмету, якщо видален студент
    " ON UPDATE CASCADE" // Оновлення посилання при зміні айді
    );")
) {
    cqcout << "Error creating predmet table:" << query.lastError().text();
}

// БД для оцінок
if (!query.exec(
    "CREATE TABLE IF NOT EXISTS grades ("
    "id INTEGER PRIMARY KEY,"
    "predmet_id TEXT,"
    "student_id TEXT,"
    "grade TEXT,"
    "FOREIGN KEY(predmet_id) REFERENCES predmet(id),"
    "FOREIGN KEY(student_id) REFERENCES students(id)"
    " ON DELETE CASCADE "
    " ON UPDATE CASCADE "
    );")
) {
    cqcout << "Error creating grades table:" << query.lastError().text();
}
}
```

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

Функції ініціалізації, а саме: initializeSpecialties, initializeFaculties, initializeClassGroups, initializeStudents, initializePredmets, initializeGrades - відповідають за те, щоб брати дані з бази даних та повертати їх з типом `QStringList` для заповнення структур даних.

```
QStringList initializeSpecialties(QSqlQuery& query) {
    QStringList specialties;
    if (!query.exec("SELECT * FROM specialty;")) {
        qDebug() << "Error fetching specialties:" << query.lastError().text();
        return specialties;
    }

    while (query.next()) {
        QString specialtyData = QString("ID: %1, Specialty: %2")
            .arg(query.value("id").toInt())
            .arg(query.value("specialty").toString());
        specialties << specialtyData;
    }
    return specialties;
}
```

```
QStringList initializeFaculties(QSqlQuery& query) {
    QStringList faculties;
    if (!query.exec("SELECT * FROM faculty;")) {
        qDebug() << "Error fetching faculties:" << query.lastError().text();
        return faculties;
    }

    while (query.next()) {
        QString facultyData = QString("ID: %1, Specialty: %2, Faculty: %3")
            .arg(query.value("id").toInt())
            .arg(query.value("specialty").toString())
            .arg(query.value("faculty").toString());
        faculties << facultyData;
    }
    return faculties;
}
```

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ



```
QStringList initializeClassGroups(QSqlQuery& query) {
    QStringList classGroups;
    if (!query.exec("SELECT * FROM class_group;")) {
        qDebug() << "Error fetching class groups:" << query.lastError().text();
        return classGroups;
    }

    while (query.next()) {
        QString classGroupData = QString("ID: %1, Specialty: %2, Faculty: %3, Class Group: %4")
            .arg(query.value("id").toInt())
            .arg(query.value("specialty").toString())
            .arg(query.value("faculty").toString())
            .arg(query.value("class_group").toInt());
        classGroups << classGroupData;
    }

    return classGroups;
}
```

```
QStringList initializeStudents(QSqlQuery& query) {
    QStringList students;
    if (!query.exec("SELECT * FROM students;")) {
        qDebug() << "Error fetching students:" << query.lastError().text();
        return students;
    }

    while (query.next()) {
        QString studentData = QString("ID: %1, Name: %2, Age: %3, Faculty: %4, Specialty: %5, Class Group: %6")
            .arg(query.value("id").toInt())
            .arg(query.value("name").toString())
            .arg(query.value("age").toInt())
            .arg(query.value("faculty").toString())
            .arg(query.value("specialty").toString())
            .arg(query.value("class_group").toInt());
        students << studentData;
    }

    return students;
}
```

Ще приклади роботи функцій ініціалізації

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

Функція `getStudentIDforPredmet` - обробляє випадок, при якому у двох різних студентів може бути однакова спеціальність. Вона зчитує раніше обраного студента за його даними, а саме ім'ям, групою і т.д. , для додавання предмету.

```
432 int getStudentIDforPredmet(QSqlQuery& query, const QString& StudyName, int studentId, const QString& SpecialtyName, const QString& FacultyName, const QString& GroupName) {
433     query.prepare(
434         "SELECT id FROM students "
435         "WHERE class_group = :classGroup AND name = :StudyName"
436     );
437     query.bindValue(":classGroup", GroupName); // class_group
438     query.bindValue(":StudyName", StudyName); // name
439
440     if (query.exec() && query.next()) {
441         studentId = query.value(0).toInt();
442         return studentId;
443     }
444     else {
445         qDebug() << "Error: Cannot find student ID. Reason:" << query.lastError().text();
446         return studentId = -1;
447     }
448 }
```

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

Функції типу `add` , а саме: `addSpecialty`, `addFaculty`, `addGroup`, `addStudent`, `addSubject`, `addGrades`. Вони відповідають за внесення даних у базу. Ці функції забезпечують додавання інформації про студента, спеціальність, факультет, групу, предмет та оцінки.

```
void addSpecialty(QSqlQuery& query, const QString& specialty) {
    // Перевірка на повтор

    query.prepare("SELECT COUNT(*) FROM specialty WHERE specialty = :specialty");
    query.bindValue(":specialty", specialty);

    if (!query.exec()) {
        qDebug() << "Error checking specialty existence:" << query.lastError().text();
        return;
    }

    if (query.next() && query.value(0).toInt() > 0) {
        qDebug() << "Error: Specialty already exists:" << specialty;
        return;
    }

    // Видача ID
    int nextId = getNextAvailableId(query, "specialty", "id");

    if (nextId != -1) {
        qDebug() << "Next available ID for specialty is:" << nextId;
    }
    else {
        qDebug() << "No available IDs.";
    }

    query.prepare("INSERT INTO specialty (id, specialty) VALUES (:id, :specialty)");
    query.bindValue(":id", nextId);
    query.bindValue(":specialty", specialty);
    if (!query.exec()) {
        qDebug() << "Error adding specialty";
    }
    else {
        qDebug() << "Specialty added successfully!";
        getSpecialty(query);
    }
}
```

```
void addFaculty(QSqlQuery& query, const QString& specialty, const QString& faculty) {
    query.prepare("SELECT COUNT(*) FROM specialty WHERE specialty = :specialty");
    query.bindValue(":specialty", specialty);
    if (!query.exec() || !query.next() || query.value(0).toInt() == 0) {
        qDebug() << "This specialty is not exist";
        return;
    }

    // Перевірка на повтор

    query.prepare("SELECT COUNT(*) FROM faculty WHERE faculty = :faculty");
    query.bindValue(":faculty", faculty);

    if (!query.exec()) {
        qDebug() << "Error checking faculty existence:" << query.lastError().text();
        return;
    }

    if (query.next() && query.value(0).toInt() > 0) {
        qDebug() << "Error: Faculty already exists:" << faculty;
        return;
    }

    // Видача ID
    int nextId = getNextAvailableId(query, "faculty", "id");

    if (nextId != -1) {
        qDebug() << "Next available ID for faculty is:" << nextId;
    }
    else {
        qDebug() << "No available IDs.";
    }

    query.prepare("INSERT INTO faculty (id, specialty, faculty) VALUES (:id, :specialty, :faculty)");
    query.bindValue(":id", nextId);
    query.bindValue(":specialty", specialty);
    query.bindValue(":faculty", faculty);

    if (!query.exec()) {
        qDebug() << "Error adding faculty" << query.lastError().text();
    }
    else {
        qDebug() << "Faculty added successfully!";
        getFaculty(query);
    }
}
```

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ

```
void addSubject(QSqlQuery& query, const QString& predmet, int studentId) {
    query.prepare("INSERT INTO predmet (predmet, student_id) VALUES (:predmet, :studentId)");
    query.bindValue(":predmet", predmet);
    query.bindValue(":studentId", studentId);

    if (!query.exec()) {
        qDebug() << "Error adding predmet:" << query.lastError().text();
    }
    else {
        qDebug() << "Predmet added successfully! Subject:" << predmet << ", Student ID:" << studentId;
    }
}
```

```
void addGrades(QSqlQuery& query, const QStringList& grades, int predmetId) {
    query.prepare("SELECT student_id FROM predmet WHERE id = :predmetId");
    query.bindValue(":predmetId", predmetId);

    if (!query.exec()) {
        qDebug() << "Error fetching student_id from predmet:" << query.lastError().text();
        return;
    }

    int studentId = -1;

    if (query.next()) {
        studentId = query.value("student_id").toInt();
    }
    else {
        qDebug() << "Error: No student found for predmet ID:" << predmetId;
        return;
    }

    clearGradesForStudentAndPredmet(query, predmetId, studentId);
    for (const QString& grade : grades) {
        query.prepare("INSERT INTO grades (grade, predmet_id, student_id) VALUES (:grade, :predmetId, :studentId)");
        query.bindValue(":grade", grade);
        query.bindValue(":predmetId", predmetId);
        query.bindValue(":studentId", studentId);

        if (!query.exec()) {
            qDebug() << "Error adding grade:" << query.lastError().text();
            break;
        }
        else {
            qDebug() << "Grade added successfully! Grade:" << grade << ", Predmet ID:" << predmetId;
        }
    }
}
```

Ще приклади роботи функцій додавання

РЕАЛІЗАЦІЯ РОБОТИ З ДАНИМИ



Функції типу `delete` , а саме: `DeleteSpecialty`, `DeleteFaculty`, `DeleteGroup`, `DeleteStudent`, `DeleteSubject`. Вони відповідають за видалення даних з бази. Ці функції забезпечують видалення даних із бази. Вони дозволяють видаляти інформацію про предмети, студентів, групи, факультети, спеціальності та оцінки.

```
void DeleteSpecialty(QSqlQuery& query, const QString& specialty) {
    // Перевірка наявності факультету в таблиці
    query.prepare("SELECT id FROM specialty WHERE specialty = :specialty");
    query.bindValue(":specialty", specialty);

    if (!query.exec()) {
        qDebug() << "Error checking specialty existence:" << query.lastError().text();
        return;
    }

    if (!query.next()) {
        qDebug() << "Error: No such specialty found";
        return;
    }

    // Беремо айді факультети
    int specialtyid = query.value(0).toInt();

    // Видаляємо всі факультети з таким айді
    query.prepare("DELETE FROM specialty WHERE id = :id");
    query.bindValue(":id", specialtyid);

    if (!query.exec()) {
        qDebug() << "Error deleting specialty:" << query.lastError().text();
    }
    else {
        qDebug() << "Specialty deleted successfully with id:" << specialtyid;
    }
}
```

```
void DeleteFaculty(QSqlQuery& query, const QString& specialty, const QString& faculty) {
    // Перевірка наявності факультету в таблиці
    query.prepare("SELECT id FROM faculty WHERE faculty = :faculty");
    query.bindValue(":faculty", faculty);

    if (!query.exec()) {
        qDebug() << "Error checking faculty existence:" << query.lastError().text();
        return;
    }

    if (!query.next()) {
        qDebug() << "Error: No such faculty found";
        return;
    }

    // Беремо айді факультети
    int facultyid = query.value(0).toInt();

    // Видаляємо всі факультети з таким айді
    query.prepare("DELETE FROM faculty WHERE id = :id");
    query.bindValue(":id", facultyid);

    if (!query.exec()) {
        qDebug() << "Error deleting faculty:" << query.lastError().text();
    }
    else {
        qDebug() << "Faculty deleted successfully with id:" << facultyid;
    }
}
```



РОЗРОБЛЕНІ АЛГОРИТМИ

Було розроблено алгоритм сортування злиттям mergeSort:

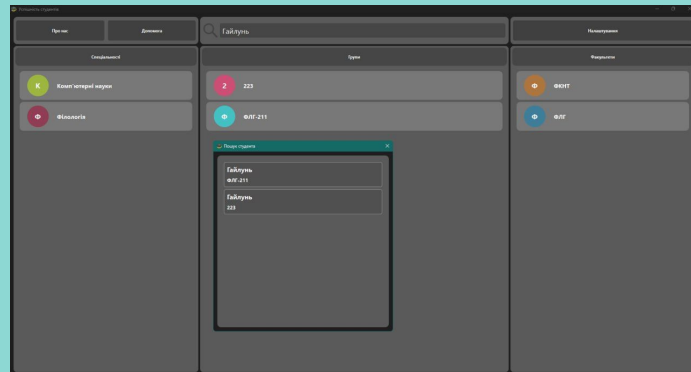
- **Поділ:** Початковий масив рекурсивно ділиться на дві рівні (або майже рівні) частини доти, доки кожна підгрупа не стане складатися з одного елемента, який вважається відсортованим.
- **Злиття:** Дві сусідні підгрупи порівнюються елемент за елементом, а їхні значення об'єднуються в новий відсортований масив. Це робиться шляхом вибору меншого елемента з кожної групи на кожному кроці.
- **Рекурсія:** Процес злиття триває доти, доки всі підгрупи не об'єднуються в один остаточний масив, повністю відсортований за зростанням (або спаданням).



РОЗРОБЛЕНІ АЛГОРИТМИ

Також було розроблено алгоритм пошуку студентів:

1. На початку текст перевіряється на відповідність регулярному виразу, щоб виключити некоректні запити.
2. Після цього масив об'єктів типу Student фільтрується відповідно до заданих критеріїв, таких як ім'я або інші параметри. Якщо відповідних студентів не знайдено, система виводить попередження про відсутність результатів.
3. У разі успішної фільтрації система відображає список знайдених студентів у вигляді діалогового вікна.
4. Після вибору користувачем дані про обраного студента зберігаються у відповідну структуру.



ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

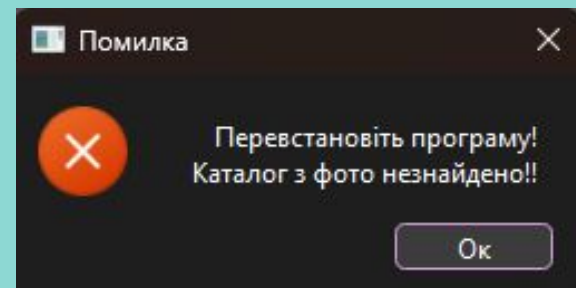
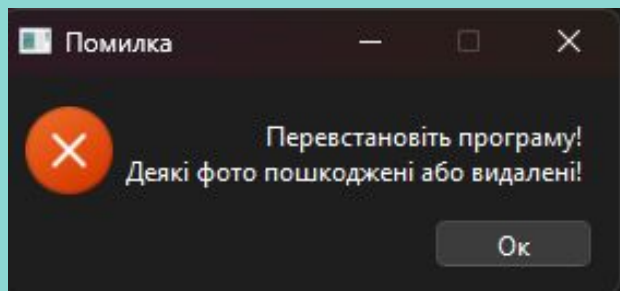
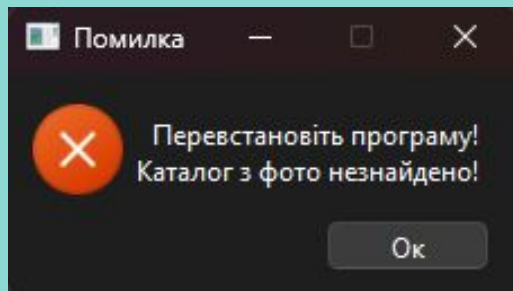
- Для обробки виняткових ситуацій у програмі присутній клас "Ex". Після обробки певної події, у випадку помилки ми отримуємо повідомлення на екрані, яке інформує нас, з чим саме виникла помилка.

```
#include <qapplication.h>
#ifndef EX_H
#define EX_H
class ex {
    int CodeError = 0;
public:
    ex(const int& NumError) { this->CodeError = NumError; }
    int getErrorCode() const { return CodeError; }
};
#endif // !EX_H
```

```
try {
    QIcon mainIcon("Images/title.png"); // Іконка для вікна програми
    QDir imagesDir("Images"); // Директорія Images
    if (!imagesDir.exists()) {
        throw ex(1);
    }
    if (mainIcon.isNull()) {
        throw ex(2);
    }
}
```

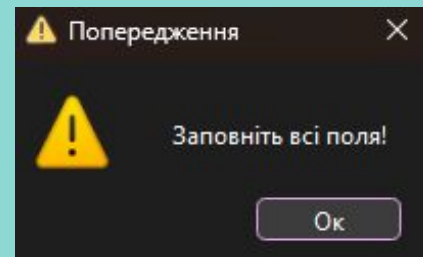
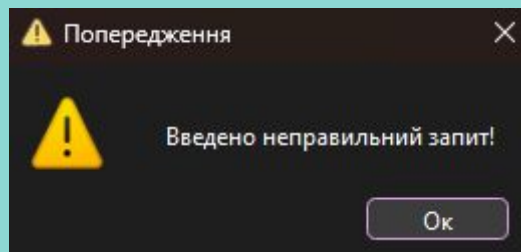
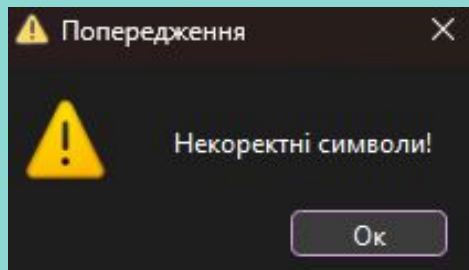
ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

- Ми маємо такі виняткові ситуації: відсутність фотографій при запуску програми, відсутність фотографій, коли програма була вже запущена, а фотографії були видалені у період користування програмою, відсутність деяких фото.



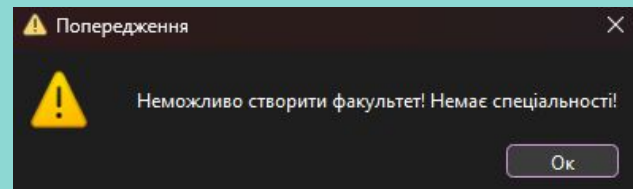
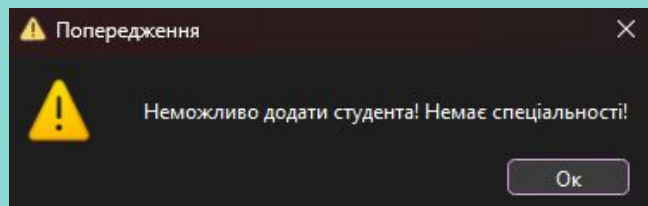
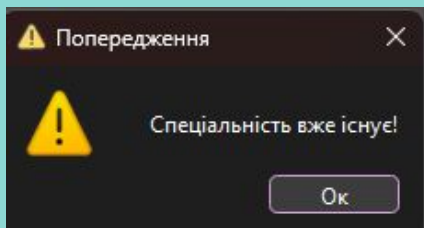
ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

- Некоректні символи при роботі з додаванням чи видаленням будь чого, некоректний формат пошуку, перевірка, чи заповнені всі поля при додаванні чи видаленні студентів у меню налаштування.

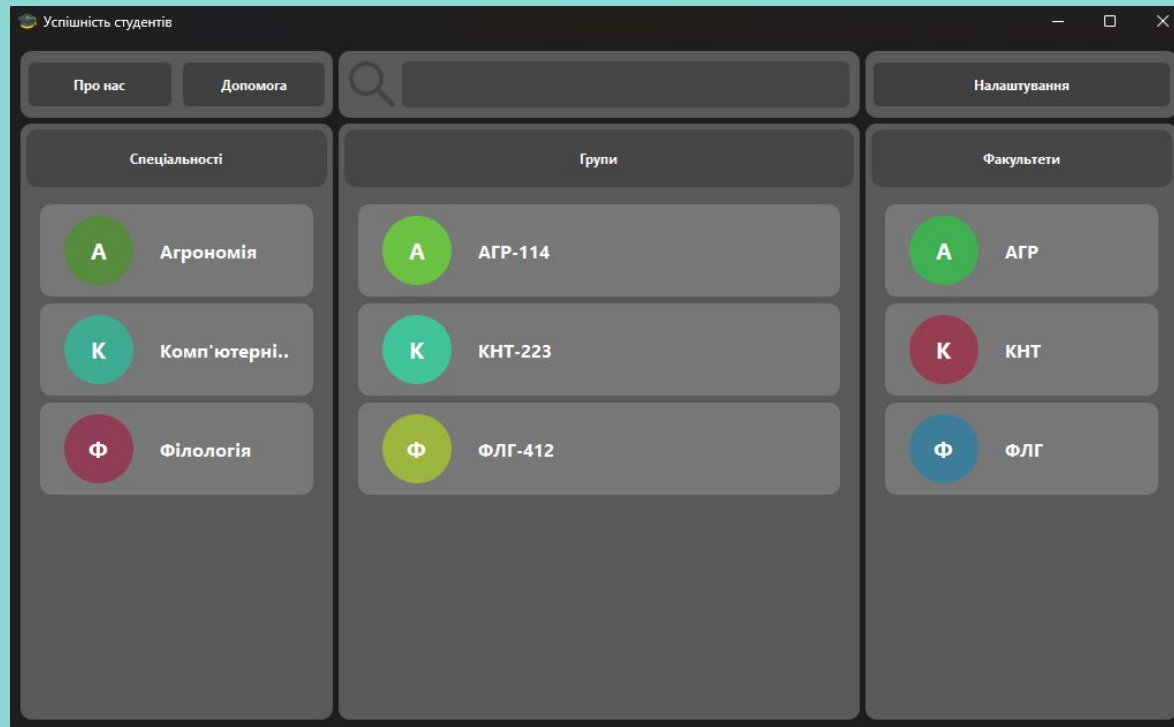


ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

→ Обробляє помилки при додаванні та видалення студентів, факультетів, груп, спеціальностей та предметів у базі даних.



ВІДЕО РОБОТИ ПРОГРАМИ



Посилання на повне відео для перегляду:

<https://drive.google.com/drive/folders/1UbyiX6dDWt0YrfQmYml0aOnMUefzqbgw?usp=sharing>

ВИСНОВКИ



Під час виконання курсового проєкту, було розроблено електронний щоденник для зберігання та роботи з даними про успішність студентів університету

Було проаналізовано сучасну модель освітньої системи та на її основі розроблено програму, яка дозволяє легко вчителю працювати зі студентами , оновлювати базу даних університету, додаючи до неї нові спеціальності, факультети , групи , студентів та їх успішність.

Розроблений програмний продукт має достатньо функцій, необхідних для повноцінного користування, але в перспективі , планується додати:

- Редагування назв вже доданих даних
- Відображення оцінок по фактичному часу її видачі
- Можливість додавати фотографії , які будуть відображатись в списку з даними.



ДЯКУЄМО ЗА УВАГУ!

Гайлунь Владислав
Шевченко Родіон
Мамонт Владислав
Бондаренко Іван
Третяк Олександр