ДОДАТОК А

Вид вхідного документа

Міністерство освіти і науки України

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

НАКАЗ

«_____» _____2024 р. **м. Харків** №_____

Про направлення на передатестаційну практику

У відповідності до навчального плану нижчевказаних здобувачів 4-го курсу першого (бакалаврського) рівня вищої освіти денної форми навчання, факультету АКТ спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології за освітньою програмою «Системна інженерія» направити з 13.05.2024 р. по 01.06.2024 р. на передатестаційну практику з можливістю використання дистанційних технологій, закріпити за ними теми кваліфікаційних робіт, призначити керівників робіт та практики.

№ п/п	Прізвище, ім'я та по батькові здобувача вищої освіти	Група	Місце проход-ження практики	Керівник практики (посада, П.І.Б.)	Тема кваліфікаційної роботи (англійською мовою)	Керівник кваліфікацій-ної роботи (посада. ПІБ)	Примітки
1	2	3	4	5	6	7	8
1	Алпатов Ілля Михайлович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розроблення технічного засобу автоматизації розподілення потоку деталей на конвеєрної лінії	проф. Новоселов С.П.	
2	Бірюков Олексій Олександрович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для моніторингу з використанням бездротових мереж	доц. Стародубцев М.Г.	

1	2	3	4	5	6	7	8
3	Вернигора Данило Сергійович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації віддаленого управління мобільним роботом	доц. Демська Н.П.	
4	Жижерін Вадим Едуардович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розроблення системи автоматизації технологічного процесу переміщення деталей конвеєром мобільного робота	проф. Новоселов С.П.	
5	Калюжний Максим Валерійович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розроблення програмного модуля для відображення телеметрії з датчиків позиціонування мобільного робота	проф. Новоселов С.П.	
6	Клевакін Олександр Олександрович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка технічного засобу дронів методом 3D друку	проф. Омаров Ш. А.	
7	Ключник Євгенія Сергіївна	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розроблення програмного забезпечення для формування маршруту переміщення свердла для верстатів з числовим програмним керуванням	доц. Бронніков А.І.	
8	Козак Владислав Вадимович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації сенсорного комплексу для моніторингу виробничих приміщень	доц. Стародубцев М.Г.	
1	2	3	4	5	6	7	8

9	Лаврик Владислав Юрійович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Моделювання триланкового маніпулятора типової модульної конструкції з магнітними шарнірами	доц. Демська Н.П.	
10	Левенець Ілля Олександрович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації підтримки академічної доброчесності	проф. Сезонова І.К.	
11	Мороз Максим Васильович	АКТСІ-20-2	АТ «Укртранс газ», м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для розсилки e-mail повідомлень по результатам аналізу подій з автоматизованою розсилкою в месенджери	доц. Іванов Л.С.	
12	Семененко Максим Андрійович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Комп'ютерне моделювання тензорезисторних вимірювальних систем	проф. Ромашов Ю.В.	
13	Тіщенко Владислав Вікторович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка комп'ютерно-інтегрованої системи автоматизації приладобудівного виробничого підрозділу	проф. Сезонова І.К.	
14	Трохін Владислав Віталійович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка бази даних обліку продукції для автоматизованої багаторівневої внутрішньо-складської підсистеми зберігання	доц. Чала О.О.	

1	2	3	4	5	6	7	8
15	Халімонов Ян Ігорович	АКТСІ-20-2	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка інтелектуального модулю для автоматизованого визначення умов життєдіяльності у житлових та робочих приміщеннях	доц. Сотник С.В.	
16	Безрук Вадим Дмитрович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка моделі мобільного колісного робота для пошуку вибухонебезпечних об'єктів	доц. Демська Н.П.	
17	Бендеберя Марія Олександрівна	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка функціональної моделі маніпулятора на базі ABB ROBOT STUDIO	доц. Бронніков А.І.	
18	Васильченко Єлизавета Романівна	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації охоронно-пожежної сигналізації на приладобудівному підприємстві	доц. Сотник С.В.	
19	Вінниченко Софія Олександрівна	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка компоненту MES-системи для оптимізації виробничих процесів підприємства	доц. Колеснік Л.В.	
20	Демченко Владислав Владленович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для керування рухом групи роботів-маніпуляторів	доц. Демська Н.П.	

1	2	3	4	5	6	7	8
21	Дзюба Сергій Сергійович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розроблення системи автоматизації для контролю мікроклімату у приміщенні із застосуванням IoT- технологій	доц. Янушкевич Д.А.	
22	Жвакін Євгеній Олексійович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для багатокольорового FDM/FFF 3D друку	доц. Разумов-Фризюк Є.А.	
23	Журков Іван Вадимович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Оптимізація роботи 3D-принтерів з використанням програмного забезпечення Klipper	доц. Замірець О. М.	
24	Іщенко Владислав Костянтинович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Проектування системи автоматизації універсального промислового інкубатора	проф. Цимбал О.М.	
25	Кирпота Федір Володимирович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації контролю обмеженого середовища портативної ділянки зеленого обіходу	доц. Сотник С.В.	
26	Логінов Микита Олександрович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для універсальної powerstation	проф. Овчаренко В. Є.	
27	Олінкевич Ярослав Віталійович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Інтеграція ERP-рішення на базі веб-сервісу для оптимізації управління ресурсами промислової компанії	доц. Колеснік Л.В.	

1	2	3	4	5	6	7	8
28	Пара Ілля Ігорович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка віртуального макету для дослідження навчання роботів на основі досвіду	ст. викл. Гурін Д.В.	
29	Проценко Дмитро Євгенійович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації управління роботом секретарем	проф. Євсєєв В.В.	
30	Решетняк Максим Сергійович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Розробка системи автоматизації для керування металошукачем з використанням Arduino	проф. Євсєєв В.В.	
31	Шкітін Дмитро Олександрович	АКТСІ-20-3	ХНУРЕ, кафедра КІТАР, м. Харків	доц. Демська Н. П.	Моделювання триланкового маніпулятора типової модульної конструкції з механічними шарнірами	доц. Демська Н.П.	

В.о. ректора Ігор РУБАН

ДОДАТОК Б

Код програми

main.h

```
#pragma once
#include "ui.h"
#include "file.h"
#include "locale.h"
#include "models/QualifyingWork.h"
#include "models/Keyword.h"
#include "models/AnalizeResult.h"
#include "QualifyingManager.h"
#include "KeywordsManager.h"
using namespace std;
int main();
```

main.cpp

```
#include "main.h"
int main(){
	setLanguageSupport();
	UI::Controller ui;
	Controllers::QualifyingManager qfManager("db.txt");
	Controllers::KeywordsManager keywords("keywords.txt");
	for (int i = 0;;) {
		ui.drawMenu();
		//В залежності від обраного розділу меню - відобразимо потрібний зміст
		if (ui.getActiveMenu() == UI::MenuIndex::ManageDb) {
			UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
					UI::setColor(UI::ConsoleColor::Gray);
			cout << " У цьому розділі можна додати кваліфікаційну роботу до бази данних." << endl;
			cout << " Натиснить ";
			UI::setColor(UI::ConsoleColor::WhiteOnBlue);
			cout << "[Enter]";
```

```
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб створити новий запис, або ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::Analize) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                vector<map<string, string>> results = keywords.analize(qfManager.getList());
                if (results.empty()) {
                        UI::setColor(UI::ConsoleColor::Red);
                        cout << "Записів для аналізу не знайдено.";
                }
                else {
                        vector<UI::Cell> head = {
                                {18,"title","Тема роботи"},
                                {13,"result","Результат"},
                                {18,"teacher","ПІБ керівніка"},
                                {18,"student","ПІБ автора"},
                                {18,"keyword","Ключове слово"},
                        };
                        ui.drawTable(head, results);
                }
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::AddRecordForm) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                Models::QualifyingWork item;
                UI::resetColor();
                cout << " Вкажіть ПІБ керівніка: ";
                getline(cin, item.teacherName);
                cout << " Вкажіть ПІБ здобувача: ";
                getline(cin, item.studentName);
                cout << " Вкажіть академічну групу: ";
```

```
                getline(cin, item.group);
                cout << " Вкажіть тему кваліфакаційної роботи: ";
                getline(cin, item.title);
                string year;
                cout << " Вкажіть рік: ";
                getline(cin, year);
                item.year = stoi(year);
                qfManager.addItem(item, true);
                UI::setColor(UI::ConsoleColor::Green);
                cout << endl << " Запис створено! " << endl;
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " Натиснить ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[Enter]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб створити новий запис, або ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::ViewDb) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
        ui.drawTable(ui.defaultTableHead,qfManager.toMap(qfManager.getList()));
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::ManageKeywords) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " У цьому розділі можна додати ключові слова до бази
данних." << endl;
                cout << " Натиснить ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[Enter]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб створити новий запис, або ";
```

```
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
            }
            else if (ui.getActiveMenu() == UI::MenuIndex::ManageKeywordsForm) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                Models::Keyword item;
                UI::resetColor();
                cout << " Вкажіть ключове слово яке потрібно додати: ";
                getline(cin, item.value);
                keywords.addItem(item, true);
                UI::setColor(UI::ConsoleColor::Green);
                cout << endl << " Запис створено! " << endl;
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " Натиснить ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[Enter]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб створити новий запис, або ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
            }
            else if (ui.getActiveMenu() == UI::MenuIndex::SearchByStudent) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " У цьому розділі можна знайти кваліфікаційну роботу за ПІБ
автора" << endl;
                cout << " Натиснить ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[Enter]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб почати пошук, або ";
```

```
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::SearchStudentForm) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                string query;
                cout << " Введіть ПІБ автора для пошука робіт: ";
                getline(cin, query);
                vector<Models::QualifyingWork> results =
qfManager.searchByStudent(query);
                if (results.empty()) {
                        UI::setColor(UI::ConsoleColor::Red);
                        cout << " Співпадінь не знайдено.";
                }
                else {
                        ui.drawTable(ui.defaultTableHead, qfManager.toMap(results));
                }
        }
        else if (ui.getActiveMenu() == UI::MenuIndex::SearchByTeacher) {
                UI::placeCursorAt(0, ui.getYScreenOffset() + 4);
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " У цьому розділі можна знайти кваліфікаційну роботу за ПІБ
керівника" << endl;
                cout << " Натиснить ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[Enter]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб почати пошук, або ";
                UI::setColor(UI::ConsoleColor::WhiteOnBlue);
                cout << "[ESC]";
                UI::setColor(UI::ConsoleColor::Gray);
                cout << " - щоб повернутись в головне меню" << endl;
        }
```

```
            else if (ui.getActiveMenu() == UI::MenuIndex::SearchTeacherForm) {
                string query;
                cout << " Введіть ПІБ керівника для пошука робіт: ";
                getline(cin, query);
                vector<Models::QualifyingWork> results =
qfManager.searchByTeacher(query);
                if (results.empty()) {
                    UI::setColor(UI::ConsoleColor::Red);
                    cout << " Співпадінь не знайдено.";
                }
                else {
                    ui.drawTable(ui.defaultTableHead, qfManager.toMap(results));
                }
            }
            ui.setKey(_getch());
        }
        return 0;
};
```

ui.h

```
#pragma once
#include <iostream>
#include <string>
#include "locale.h"
#include <conio.h>
#include <cstdarg>
#include "navigation.h"
#include "QualifyingManager.h"
#include "table.h"
#include "helpers.h"

using namespace std;
namespace UI {
    /// <summary>
    /// Помічник для відображення елементів консолі
```

```
/// </summary>
class Controller: public UI::Navigation {
public:
	Controller();

	/// <summary>
	/// Друкує навігаційне меню
	/// </summary>
	void drawMenu();
	/// <summary>
	/// Очистити консоль
	/// </summary>
	void clear();
	/// <summary>
	/// Друкує заголовок
	/// </summary>
	/// <param name="title">Заголовок</param>
	void drawHeader(string title);
	/// <summary>
	/// Додати вертикальне меню
	/// </summary>
	/// <param name="x">Початкова координата x</param>
	/// <param name="y">Початкова координата y</param>
	/// <param name="arg">Розділи меню</param>
	void drawVerticalNavigation(int x, int y, const char* arg, ...);
	/// <summary>
	/// Додати горизонтальне меню
	/// </summary>
	/// <param name="x">Початкова координата x</param>
	/// <param name="y">Початкова координата y</param>
	/// <param name="arg">Розділи меню</param>
	void drawHorizontalNavigation(int x, int y, const char* arg, ...);
	/// <summary>
	/// Друкує список
	/// </summary>
```

```
/// <param name="list"></param>
void drawList(vector<map<string,string>> list);
/// <summary>
/// Відобразити таблицю
/// </summary>
/// <param name="head">Комірки</param>
/// <param name="body">Список записів таблиці</param>
void drawTable(vector<UI::Cell> head, vector<map<string, string>> body);
/// <summary>
/// Вивести вміст із зазначенним кольором
/// </summary>
/// <param name="content"></param>
/// <param name="color"></param>
void print(string content, UI::ConsoleColor color) {
	setColor(color);
	cout << content;
};
/// <summary>
/// Активний розділ меню
/// </summary>
/// <returns></returns>
UI::MenuIndex getActiveMenu() {
	return selectedMenuIndex;
};
/// <summary>
/// Колбек коли обран розділ меню
/// </summary>
void onNavigationItemSelect() override;
/// <summary>
/// Колбек коли натиснута кнопка ESC
/// </summary>
void onEscape() override;
int getXScreenOffset() {
	return xScreenOffset;
};
```

```
		int getYScreenOffset() {
			return yScreenOffset;
		};
		/// <summary>
		/// Превстановленна шапка таблиці
		/// </summary>
		vector<UI::Cell> defaultTableHead = {
			{18,"teacher","ПІБ керівніка"},
			{18,"student","ПІБ автора"},
			{24,"title","Тема роботи"},
			{18,"group","Академічна группа"},
			{6,"year","Рік"},
		};
	private:
		MenuIndex selectedMenuIndex = MenuIndex::Home;
	protected:
		int xScreenOffset = 1;
		int yScreenOffset = 1;
	};
};
```

ui.cpp

```
#include "ui.h"
namespace UI {

	Controller::Controller() : UI::Navigation() {
	};
	void Controller::drawMenu() {
		clear();
		//В залежності від обраного розділу меню - будемо виводити різний зміст
		switch (selectedMenuIndex) {
		//Головне меню
		case UI::MenuIndex::Home:
			drawHeader("Головне меню");
			drawVerticalNavigation(
```

```
                xScreenOffset,
                4,
                "Аналіз назви кваліфікаційної роботи",
                "Поповнити базу даних",
                "Перегляд бази даних кафедри",
                "Додати ключові слова",
                "Пошук робіт за прізвищем автора",
                "Пошук робіт за прізвищем керівника"
                , NULL);
        resetColor();

        UI::placeCursorAt(xScreenOffset, yScreenOffset+1);
        UI::setColor(UI::ConsoleColor::WhiteOnBlue);
        cout << "[Стрілки вгору - вниз]";
        UI::setColor(UI::ConsoleColor::Gray);
        cout << " - Навігація по меню ";
        UI::setColor(UI::ConsoleColor::WhiteOnBlue);
        cout << "[Enter]";
        UI::setColor(UI::ConsoleColor::Gray);
        cout << " - Перехід до розділу меню";
        break;
//Розділ аналізу назви
case UI::MenuIndex::Analize:
        drawHeader("Аналіз назви кваліфікаційної роботи");
        UI::placeCursorAt(xScreenOffset, yScreenOffset + 1);
        UI::setColor(UI::ConsoleColor::WhiteOnBlue);
        cout << "[ESC]";
        UI::setColor(UI::ConsoleColor::Gray);
        cout << " - Повернутись до головного меню ";
        resetColor();
        break;
//Розділ створення ключових слів
case UI::MenuIndex::ManageKeywords:
case UI::MenuIndex::ManageKeywordsForm:
        drawHeader("Додати ключові слова");
```

```
            UI::placeCursorAt(xScreenOffset, yScreenOffset + 1);
            resetColor();
            break;
    //Розділ створення записів бази кваліфікаційних робіт
    case UI::MenuIndex::ManageDb:
    case UI::MenuIndex::AddRecordForm:
            drawHeader("Поповнити базу даних");
            UI::placeCursorAt(xScreenOffset, yScreenOffset + 1);
            resetColor();
            break;
    //Перегляд бази квал. робіт
    case UI::MenuIndex::ViewDb:
            drawHeader("Перегляд бази даних кафедри");
            UI::placeCursorAt(xScreenOffset, yScreenOffset + 1);
            UI::setColor(UI::ConsoleColor::WhiteOnBlue);
            cout << "[ESC]";
            UI::setColor(UI::ConsoleColor::Gray);
            cout << " - Повернутись до головного меню ";
            resetColor();
            break;
    //Пошук робіт за ПІБ керівника
    case UI::MenuIndex::SearchByTeacher:
    case UI::MenuIndex::SearchTeacherForm:
            drawHeader("Пошук робіт за прізвищем керівника");
            resetColor();
            break;
    //Пошук робіт за ПІБ студента
    case UI::MenuIndex::SearchByStudent:
    case UI::MenuIndex::SearchStudentForm:
            drawHeader("Пошук робіт за прізвищем автора");
            resetColor();
            break;
    default:
            break;
    };
```

```
};

void Controller::clear() {
	cout << "\033c";
};

void Controller::drawHeader(string title) {
	placeCursorAt(xScreenOffset, yScreenOffset);
	print(title, UI::ConsoleColor::LightBlue);
};

void Controller::drawVerticalNavigation(int x, int y, const char* arg, ...) {
	va_list args;
	placeCursorAt(x, y);
	int i = 0;
	for (va_start(args, arg); arg != NULL; arg = va_arg(args, const char*)) {
		placeCursorAt(x, y + i);
		if (getNavigationVerticalIndex() == i) {
			setColor(UI::ConsoleColor::Red);
		}
		else {
			setColor(UI::ConsoleColor::White);
		}
		cout << "|" << arg;
		i++;
	};
	setVerticalLength(i);
	va_end(args);
};

void Controller::drawHorizontalNavigation(int x, int y, const char* arg, ...) {
	va_list args;
	placeCursorAt(x, y);
	int width = 0;
	int i = 0;
```

```
        for (va_start(args, arg); arg != NULL; arg = va_arg(args, const char*)) {
                placeCursorAt(x + width, y);
                if (getNavigationHorizontalIndex() == i) {
                        setColor(UI::ConsoleColor::Red);
                }
                else {
                        setColor(UI::ConsoleColor::White);
                }
                cout << "|" << arg;
                i++;
                width += String::getCharLength(arg);
        };
        setHorizontalLength(i);
        va_end(args);
};

void Controller::drawList(vector<map<string, string>> list) {
        for (const auto& row : list) {
                for (const auto& pair : row) {
                        cout << pair.second;
                };
        };
};

void Controller::drawTable(vector<UI::Cell> head, vector<map<string, string>> body) {
        Table table;
        table.render(head, body);
};

void Controller::onNavigationItemSelect() {
        if (selectedMenuIndex == UI::MenuIndex::Home) {
                int index = getNavigationVerticalIndex();
                selectedMenuIndex = UI::MenuIndex(index+1);
        }
        else if (selectedMenuIndex == UI::MenuIndex::ManageDb) {
```

```
                selectedMenuIndex = UI::MenuIndex::AddRecordForm;
        }
        else if (selectedMenuIndex == UI::MenuIndex::SearchByStudent) {
                selectedMenuIndex = UI::MenuIndex::SearchStudentForm;
        }
        else if (selectedMenuIndex == UI::MenuIndex::SearchByTeacher) {
                selectedMenuIndex = UI::MenuIndex::SearchTeacherForm;
        }
        else if (selectedMenuIndex == UI::MenuIndex::ManageKeywords) {
                selectedMenuIndex = UI::MenuIndex::ManageKeywordsForm;
        }
    };

    void Controller::onEscape() {
        if (selectedMenuIndex != UI::MenuIndex::Home) {
                selectedMenuIndex = UI::MenuIndex::Home;
        }
    };
}
```

file.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <vector>

using namespace std;

/// <summary>
/// Класс читання та запису файлів
/// </summary>
class FileManager {
```

```
public:
	FileManager(const char* path) {
		_path = path;
	};
	/// <summary>
	/// Зчитує файл
	/// </summary>
	/// <returns></returns>
	vector<map<string,string>> read();
	/// <summary>
	/// Записує у файл
	/// </summary>
	/// <param name="data">Дані які треба записати</param>
	/// <returns></returns>
	bool write(vector<map<string, string>> data);
	/// <summary>
	/// Встановити розділовий знак для комірок значень
	/// </summary>
	/// <param name="delimiter">Розділовий знак</param>
	void setDelimiter(string delimiter) {
		componentDelimiter = delimiter;
	};
	/// <summary>
	/// Встановити розділовий знак для пари ключ - значення
	/// </summary>
	/// <param name="delimiter">Розділовий знак</param>
	void setKeyDelimiter(string delimiter) {
		keyPairDelimiter = delimiter;
	};
	/// <summary>
	/// Шлях до файлу
	/// </summary>
	/// <returns></returns>
	const char* getFilePath();
	/// <summary>
```

```
        /// Показує, чи вдалось успішно прочитати вхідний файл
        /// </summary>
        /// <returns></returns>
        bool isInitialized() {
                return initialized;
        };
private:
        /// <summary>
        /// Розділовий знак для комірки значення
        /// </summary>
        string componentDelimiter = ";";
        /// <summary>
        /// Розділовий знак для пари ключ-значення
        /// </summary>
        string keyPairDelimiter = ":";
protected:
        /// <summary>
        /// Шлях до файлу
        /// </summary>
        const char* _path;
        /// <summary>
        /// Показує чи був успішно прочитан файл
        /// </summary>
        bool initialized = false;
};
```

file.cpp

```
#include "file.h"

const char* FileManager::getFilePath() {
        return _path;
};

vector<map<string, string>> FileManager::read() {
        ifstream file(getFilePath());
```

```
vector<map<string, string>> output;

if (!file.is_open()) {
	initialized = false;
	return output;
}
initialized = true;
string line;
	while (getline(file, line)) {
	remove_if(line.begin(), line.end(), isspace);
	string clone = line;
	size_t index = 0;
	string component;
	map<string, string> pair;
	while ((index = clone.find(componentDelimiter)) != string::npos) {
		component = clone.substr(0, index);
		size_t valueIndex = component.find(keyPairDelimiter);

		if (valueIndex != string::npos) {
			string key = component;
			string value = component;
			key.erase(valueIndex);
			value.erase(0, valueIndex + 1);

			pair.insert(make_pair(key, value));
		};

		clone.erase(0, index + componentDelimiter.length());
	};
	output.push_back(pair);
};

file.close();

return output;
```

```
};

bool FileManager::write(vector<map<string, string>> data) {
	ofstream file(getFilePath());
	if (!file.is_open()) {
		return false;
	};

	for (const auto& row : data) {
		for (const auto& pair : row) {
			file << pair.first << keyPairDelimiter << pair.second << componentDelimiter;
		};
		file << endl;
	};

	file.close();

	return true;
};
```

locale.h

```
/*
@brief Файл для підтримки української локалізації
*/
#pragma once

#include <locale>
#include "windows.h"

/// <summary>
/// Додає підтримку української мови до консолі
/// </summary>
void setLanguageSupport();
```

locale.cpp

```
#include "locale.h"

void setLanguageSupport() {
	setlocale(LC_ALL, "Ukr");
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
};
```

models/QualifyingWork.h

```
#pragma once
#include <string>
#include <iostream>
#include <map>

using namespace std;

namespace Models {
	/// <summary>
	/// Кваліфікаційна робота
	/// </summary>
	struct QualifyingWork {
		/// <summary>
		/// ПІБ викладача
		/// </summary>
		string teacherName;
		int year;
		/// <summary>
		/// Академічна група
		/// </summary>
		string group;
		/// <summary>
		/// ПІБ Автора
		/// </summary>
		string studentName;
```

```
            /// <summary>
            /// Назва роботи
            /// </summary>
            string title;

            int primaryKey;

            map<string, string> getPair() {
                map<string, string> pair;
                pair.insert(make_pair("teacher", teacherName));
                pair.insert(make_pair("student", studentName));
                pair.insert(make_pair("title", title));
                pair.insert(make_pair("group", group));
                pair.insert(make_pair("year", to_string(year)));
                return pair;
            };
        };
};
```

models/Keyword.h

```
#pragma once
#include <string>
#include <iostream>
#include <map>

using namespace std;

namespace Models {
    /// <summary>
    /// Ключове слово
    /// </summary>
    struct Keyword {
        /// <summary>
        /// Значення
        /// </summary>
```

```
		string value;
		int primaryKey;
		/// <summary>
		/// Трансформує модель у формат пари ключ-значення для відображення та
запису
		/// </summary>
		/// <returns></returns>
		map<string, string> getPair() {
			map<string, string> pair;
			pair.insert(make_pair("value", value));
			return pair;
		};
	};
};
```

models/AnalizeResult.h

```
#pragma once
#include <string>
#include <iostream>
#include <map>
#include "QualifyingWork.h"

using namespace std;

namespace Models {
	/// <summary>
	/// Результат аналізу робіт
	/// </summary>
	struct AnalizeResult {
		Models::QualifyingWork value;
		bool result = false;
		string keyword;

		/// <summary>
```

```
        /// Трансформує модель у формат пари ключ-значення для відображення та
запису
        /// </summary>
        ///     <returns></returns>
        map<string, string> getPair() {
            map<string, string> pair;
            pair.insert(make_pair("title", value.title));
            pair.insert(make_pair("student", value.studentName));
            pair.insert(make_pair("teacher", value.teacherName));
            if (result) {
                pair.insert(make_pair("result", "Знайдено"));
            }
            else {
                pair.insert(make_pair("result", "Не знайдено"));
            }
            pair.insert(make_pair("keyword", keyword));
            return pair;
        };
    };
};
```

QualifyingManager.h

```
#pragma once

#include <vector>
#include "models/QualifyingWork.h"
#include "file.h"

using namespace std;
namespace Controllers {
    /// <summary>
    /// Контролер бази кваліфікаційних робіт
    /// </summary>
    class QualifyingManager : public FileManager {
    public:
```

```
        QualifyingManager(const char* path);
        /// <summary>
        /// Додати новий запис до бази
        /// </summary>
        /// <param name="item">Запис</param>
        /// <param name="overrideFile">Чи потрібно перезаписати файл бази</param>
        /// <returns></returns>
        int addItem(Models::QualifyingWork& item, bool overrideFile);
        int getPrimaryKey() {
                return primaryIndex;
        };
        int increasePrimaryKey() {
                primaryIndex++;
                return primaryIndex;
        };
        vector<Models::QualifyingWork> getList();
        vector<map<string, string>> toMap(vector<Models::QualifyingWork> input);
        /// <summary>
        /// Пошук єлементів бази за параметром ПІБ викладача
        /// </summary>
        /// <param name="query">Пошуковий запит</param>
        /// <returns></returns>
        vector<Models::QualifyingWork> searchByTeacher(string query);
        /// <summary>
        /// Пошук записів за ПІБ автора
        /// </summary>
        /// <param name="query">Пошуковий запит</param>
        /// <returns></returns>
        vector<Models::QualifyingWork> searchByStudent(string query);
private:
        vector<Models::QualifyingWork> list;
        /// <summary>
        /// Оновити файл бази
        /// </summary>
        /// <returns></returns>
```

```
        bool override();
    protected:
        int primaryIndex = 0;
    };
};
```

QualifyingManager.cpp

```
#include "QualifyingManager.h"

namespace Controllers {
    QualifyingManager::QualifyingManager(const char* path) : FileManager(path) {
        vector<map<string, string>> data = read();

        for (const auto& row : data) {
            Models::QualifyingWork item;
            if (row.find("teacher") != row.end()) {
                item.teacherName = row.at("teacher");
            }
            if (row.find("student") != row.end()) {
                item.studentName = row.at("student");
            }
            if (row.find("title") != row.end()) {
                item.title = row.at("title");
            }
            if (row.find("group") != row.end()) {
                item.group = row.at("group");
            }
            if (row.find("year") != row.end()) {
                item.year = stoi(row.at("year"));
            }
            addItem(item, false);
        };
    };

    int QualifyingManager::addItem(Models::QualifyingWork& item, bool overrideFile) {
```

```
        int primaryKey = getPrimaryKey() + 1;
        item.primaryKey = primaryKey;
        list.push_back(item);
        increasePrimaryKey();

        if (overrideFile) {
            override();
        }
        return primaryKey;
    };

    bool QualifyingManager::override() {
        vector<map<string, string>> data;
        for (Models::QualifyingWork row : list) {
            data.push_back(row.getPair());
        };
        return write(data);
    };

    vector<Models::QualifyingWork> QualifyingManager::getList(){
        return list;
    };

    vector<map<string, string>>
QualifyingManager::toMap(vector<Models::QualifyingWork> input) {
        vector<map<string, string>> output;
        for (Models::QualifyingWork row : input) {
            map<string, string> pair = row.getPair();
            output.push_back(pair);
        }
        return output;
    };

    vector<Models::QualifyingWork> QualifyingManager::searchByTeacher(string query) {
        vector<Models::QualifyingWork> results;
```

```
            for (Models::QualifyingWork row : getList()) {
                size_t result = row.teacherName.find(query);
                if (result != string::npos) {
                    results.push_back(row);
                }
            }
            return results;
        };

        vector<Models::QualifyingWork> QualifyingManager::searchByStudent(string query) {
            vector<Models::QualifyingWork> results;
            for (Models::QualifyingWork row : getList()) {
                size_t result = row.studentName.find(query);
                if (result != string::npos) {
                    results.push_back(row);
                }
            }
            return results;
        };
};
```

KeywordsManager.h

```
#pragma once

#include <vector>
#include "file.h"
#include "models/Keyword.h"
#include "models/AnalizeResult.h"
#include "models/QualifyingWork.h"

using namespace std;
namespace Controllers {
    /// <summary>
    /// Менеджер ключових слів
    /// </summary>
```

```
class KeywordsManager : public FileManager {
public:
	KeywordsManager(const char* path);
	/// <summary>
	/// Додати запис до бази
	/// </summary>
	/// <param name="item">Ключове слово</param>
	/// <param name="overrideFile">Чи потрібно перезаписати файл бази</param>
	/// <returns></returns>
	int addItem(Models::Keyword& item, bool overrideFile);
	int getPrimaryKey() {
		return primaryIndex;
	};
	int increasePrimaryKey() {
		primaryIndex++;
		return primaryIndex;
	};
	/// <summary>
	/// Вміст бази
	/// </summary>
	/// <returns></returns>
	vector<Models::Keyword> getList();
	/// <summary>
	/// Трансформує список ключових слів у формат відображення
	/// </summary>
	/// <param name="input"></param>
	/// <returns></returns>
	vector<map<string, string>> toMap(vector<Models::Keyword> input);
	/// <summary>
	/// Проводить аналіз квал. робіт
	/// </summary>
	/// <param name="input"></param>
	/// <returns></returns>
	vector<map<string, string>> analize(vector<Models::QualifyingWork> input);
private:
```

```
        vector<Models::Keyword> list;
        /// <summary>
        /// Оновлення бази
        /// </summary>
        /// <returns></returns>
        bool override();
    protected:
        /// <summary>
        /// Індекс останнього доданого запису
        /// </summary>
        int primaryIndex = 0;
    };
};
```

KeywordsManager.cpp

```
#include "QualifyingManager.h"

namespace Controllers {
    QualifyingManager::QualifyingManager(const char* path) : FileManager(path) {
        vector<map<string, string>> data = read();

        for (const auto& row : data) {
            Models::QualifyingWork item;
            if (row.find("teacher") != row.end()) {
                item.teacherName = row.at("teacher");
            }
            if (row.find("student") != row.end()) {
                item.studentName = row.at("student");
            }
            if (row.find("title") != row.end()) {
                item.title = row.at("title");
            }
            if (row.find("group") != row.end()) {
                item.group = row.at("group");
            }
```

```
                if (row.find("year") != row.end()) {
                        item.year = stoi(row.at("year"));
                }
                addItem(item, false);
        };
};

int QualifyingManager::addItem(Models::QualifyingWork& item, bool overrideFile) {
        int primaryKey = getPrimaryKey() + 1;
        item.primaryKey = primaryKey;
        list.push_back(item);
        increasePrimaryKey();

        if (overrideFile) {
                override();
        }
        return primaryKey;
};

bool QualifyingManager::override() {
        vector<map<string, string>> data;
        for (Models::QualifyingWork row : list) {
                data.push_back(row.getPair());
        };
        return write(data);
};

vector<Models::QualifyingWork> QualifyingManager::getList(){
        return list;
};

vector<map<string, string>>
QualifyingManager::toMap(vector<Models::QualifyingWork> input) {
        vector<map<string, string>> output;
        for (Models::QualifyingWork row : input) {
```

```
                map<string, string> pair = row.getPair();
                output.push_back(pair);
            }
            return output;
        };

        vector<Models::QualifyingWork> QualifyingManager::searchByTeacher(string query) {
            vector<Models::QualifyingWork> results;
            for (Models::QualifyingWork row : getList()) {
                size_t result = row.teacherName.find(query);
                if (result != string::npos) {
                    results.push_back(row);
                }
            }
            return results;
        };

        vector<Models::QualifyingWork> QualifyingManager::searchByStudent(string query) {
            vector<Models::QualifyingWork> results;
            for (Models::QualifyingWork row : getList()) {
                size_t result = row.studentName.find(query);
                if (result != string::npos) {
                    results.push_back(row);
                }
            }
            return results;
        };
};
```

navigation.h

```
#pragma once

#include "windows.h"
namespace UI {
    /// <summary>
```

```
/// Створення навигації
/// </summary>
class Navigation {
public:
	Navigation() {};

	/// <summary>
	/// Встановлює позицію для вертикальної навигації
	/// </summary>
	/// <param name="index"></param>
	void setVerticalIndex(int index) {
		activeVerticalIndex = index;
	};
	/// <summary>
	/// Встановлює позицію для горизонтальної навигації
	/// </summary>
	/// <param name="index"></param>
	void setHorizontalIndex(int index) {
		activeHorizontalIndex = index;
	};
	/// <summary>
	/// Встановлює кількість єлементів вертикальної навигації
	/// </summary>
	void setVerticalLength(int length) {
		activeVerticalLength = length;
	};
	/// <summary>
	/// Встановлює кількість єлементів горизонтальної навигації
	/// </summary>
	void setHorizontalLength(int length) {
		activeHorizontalLength = length;
	};
	void setKey(char key);

	int getNavigationVerticalIndex() {
```

```
                return activeVerticalIndex;
        }
        int getNavigationHorizontalIndex() {
                return activeHorizontalIndex;
        }
        /// <summary>
        /// Колбек коли обран пункт
        /// </summary>
        virtual void onNavigationItemSelect() = 0;
        /// <summary>
        /// Колбек коли нажата кнопка "ESC"
        /// </summary>
        virtual void onEscape() = 0;
    private:
        int activeVerticalIndex = 0;
        int activeHorizontalIndex = 0;

        int activeVerticalLength = 0;
        int activeHorizontalLength = 0;
    protected:
        char _key;
    };
};
```

navigation.cpp

```
#include "navigation.h"

void UI::Navigation::setKey(char key) {
    _key = key;
    if (key == 72) {
        if (activeVerticalIndex > 0) {
            activeVerticalIndex--;
        }
    }
    else if (key == 80) {
```

```
            if (activeVerticalIndex < activeVerticalLength) {
                activeVerticalIndex++;
            }
        }
        else if (key == 75) {
            if (activeHorizontalIndex > 1) {
                activeHorizontalIndex--;
            }
        }
        else if (key == 77) {
            if (activeHorizontalIndex < activeHorizontalLength) {
                activeHorizontalIndex++;
            }
        }
        else if (key == '\r') {
            onNavigationItemSelect();
        }
        if (GetAsyncKeyState(VK_ESCAPE)) {
            onEscape();
        }
};
```

table.h

```
#pragma once

#include <iostream>
#include <string>

#include <conio.h>
#include <cstdarg>
#include <map>
#include <vector>
#include "helpers.h"

using namespace std;
```

```
namespace UI {
	struct Cell {
		int width;
		string key;
		string label;
	};

	/// <summary>
	/// UI помічник відображення таблиць у консолі
	/// </summary>
	class Table {
	public:
		Table() {};
		/// <summary>
		/// Друкує таблицю до консолі
		/// </summary>
		/// <param name="head">Список колонок які треба відобразити</param>
		/// <param name="body">Список записів таблиці</param>
		void render(vector<Cell> head, vector<map<string, string>> body);
		/// <summary>
		/// Друкує комірку таблиці
		/// </summary>
		/// <param name="value">Значення комірки</param>
		/// <param name="width">Кількість символів (ширина) комірки</param>
		/// <param name="color">Колір комірки</param>
		void drawCell(string value, int width, UI::ConsoleColor color);
		/// <summary>
		/// Друкує горизонтальну розділову лінію
		/// </summary>
		/// <param name="width">Кількість символів (ширина) лішії</param>
		void drawDivider(int width);
	};
};
```

table.cpp

```
#include "table.h"

namespace UI {
	void Table::render(vector<Cell> head, vector<map<string, string>> body) {
		//Дізнаємось загальну ширину всіх комірок
		int width = -1;
		for (UI::Cell cell : head) {
			width += cell.width+1;
		}

		drawDivider(width);
		for (UI::Cell cell : head) {
			drawCell(cell.label, cell.width, UI::ConsoleColor::Gray);
		}

		setColor(UI::ConsoleColor::Gray);
		cout << "|" << endl;
		drawDivider(width);

		// Для кожного рядка надрукуємо кожну комірку шапки
		for (const auto& row : body) {
			for (UI::Cell cell : head) {
				string value = "";
				if (row.find(cell.key) != row.end()) {
					value = row.at(cell.key);
				}
				drawCell(value, cell.width, UI::ConsoleColor::White);
			}
			setColor(UI::ConsoleColor::Gray);
			cout << "|" << endl;
			drawDivider(width);
		}
	};
```

```
	void Table::drawCell(string value, int width, UI::ConsoleColor color) {
		setColor(UI::ConsoleColor::Gray);
		cout << "|";
		setColor(color);
		cout << " " << value;

		//Додамо пробіли щоб "добити" ширину комірки
		int dif = width - value.length();
		if (dif > 0) {
			int i = 0;
			while (i < (dif - 1)) {
				cout << " ";
				i++;
			}
		}
	};

	void Table::drawDivider(int width) {
		setColor(UI::ConsoleColor::Gray);
		cout << " ";
		int i = 0;
		while (i < width && width >= 0) {
			cout << "-";
			i++;
		}
		cout << endl;
	};
};
```

helpers.h

```
#pragma once

#include <iostream>
#include "windows.h"
```

```
using namespace std;
namespace UI {
	/// <summary>
	/// Індекси розділів програми
	/// </summary>
	enum MenuIndex {
		Home = 0,
		Analize = 1,
		ManageDb = 2,
		ViewDb = 3,
		ManageKeywords = 4,
		SearchByStudent = 5,
		SearchByTeacher = 6,
		AddRecordForm = 7,
		SearchStudentForm = 8,
		SearchTeacherForm = 9,
		ManageKeywordsForm = 10
	};
	/// <summary>
	/// Індекси кольорів консолі
	/// </summary>
	enum ConsoleColor {
		LightGray = 7,
		Gray = 8,
		Green = 10,
		LightBlue = 11,
		Red = 12,
		White = 15,
		WhiteOnBlue = 23
	};
	/// <summary>
	/// Встановлює курсор консолі на заданий рядок, колонку
	/// </summary>
	/// <param name="x">Координата x (знак)</param>
	/// <param name="y">Координата y (рядок)</param>
```

```
        void placeCursorAt(int x, int y);
        /// <summary>
        /// Встановлює колір консолі
        /// </summary>
        /// <param name="color">Колір</param>
        void setColor(UI::ConsoleColor color);
        /// <summary>
        /// Повертає колір консолі до стандартного
        /// </summary>
        void resetColor();
};

namespace String {
        /// <summary>
        /// Кількість символів char*
        /// </summary>
        /// <param name="input"></param>
        /// <returns>Кількість символів</returns>
        int getCharLength(const char* input);
};
```

helpers.cpp

```
#include "helpers.h"

namespace UI {
        void placeCursorAt(int x, int y) {
                COORD c;
                c.X = x;
                c.Y = y;
                SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), c);
        };

        void setColor(UI::ConsoleColor color) {
                SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
static_cast<int>(color));
```

```
	};

	void resetColor() {
		setColor(UI::ConsoleColor::White);
	};
}

namespace String {
	int getCharLength(const char* input) {
		int length = 0;
		while (input[length] != '\0') length++;
		return length;
	};
}
```

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 442351.012 ПЗ	Пояснювальна записка	А4, 104 с.
		Додаткові матеріали	
2		Демонстраційний матеріал у	А4, 15 с.
		вигляді презентації	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.442351.012 ВД		
Розроб.		Левенець І.О.			Розробка автоматизованої системи підтримки академічної доброчесності	Літера	Аркуш	Аркушів
Перевір.		Сезонова І. К.				Н	1	1
Затв.		Невлюдов І.Ш.				Кафедра КІТАР ХНУРЕ		