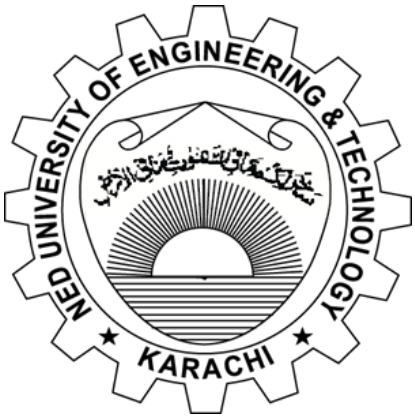


Department of Software Engineering
NED University of Engineering and Technology

Complex Engineering Problem
Fully Developed Project

Smart Teaching Assistant

Object Oriented Programming (SE-201)
Theory Teacher: Miss Asma
Lab Teacher: Miss Sheerina
Section B | Group 1

Members:
Muhammad Awwab Khan (SE-23051)
Nabeed Jamshed (SE-23054)
Muhammad Talha (SE-23057)
Talha Hussain (SE-23062)

Table of Contents

Abstract

The Smart Teaching Assistant is a comprehensive application designed to facilitate teachers in managing multiple classes and quizzes, capturing and analyzing student responses via Optical Mark Recognition (OMR), and maintaining a digital whiteboard for class sessions. This project uses the power of Qt for the graphical interface, SQLite for database management, and OpenCV for image processing. The Purpose of this application is to streamline classroom management, enhance teaching efficiency, and provide detailed insights into student performance.

Introduction

Background and Motivation

With the increasing reliance on digital solutions in education, there is a growing need for tools that assist teachers in managing classroom activities efficiently. Traditional methods of handling class activities, quizzes, and whiteboard sessions can be time-consuming and prone to errors. The Smart Teaching Assistant aims to address these challenges by providing an integrated platform that combines classroom management, quiz handling, and digital whiteboard capabilities into a single application.

Objectives

The primary objectives of the Smart Teaching Assistant project are:

- To provide a user-friendly interface for managing multiple classes and students.
- To enable teachers to create, manage, and grade quizzes efficiently.
- To offer a digital whiteboard for interactive teaching sessions.
- To utilize Optical Mark Recognition (OMR) for automated quiz grading.
- To provide detailed reports and insights into student performance.

Scope of the Project

The scope of the project includes the development of a desktop application using the Qt framework, integrating image processing capabilities using OpenCV, and managing data storage and retrieval through SQLite. The application is designed to be used by teachers for classroom management, quiz handling, and interactive teaching sessions.

Literature Review

Relevant Research

The development of a Smart Teaching Assistant application is informed by several areas of research, including educational technology, human-computer interaction, and automated assessment systems. The use of technology in education has been widely studied, highlighting the benefits of integrating digital tools into the classroom to enhance learning experiences and improve administrative efficiency. Key research areas include:

1. **Educational Technology Integration**:
    - Studies have shown that technology can significantly enhance the learning experience by providing interactive and engaging tools for both students and teachers (Hwang & Wu, 2014).

    - The role of Learning Management Systems (LMS) in organizing and managing educational content has been pivotal. Our application takes inspiration from the functionalities offered by LMSs but focuses on a more streamlined and user-friendly approach tailored for classroom management.
2. **Automated Assessment Systems**:
    - The use of Optical Mark Recognition (OMR) for automated grading is well-documented (Pope, 1976). Our project leverages this technology to simplify the quiz grading process, reducing manual effort and minimizing errors.
    - The inclusion of negative marking and detailed analysis of quiz results are features that align with modern educational assessment strategies, which emphasize accuracy and fairness in student evaluations.
3. **Human-Computer Interaction (HCI)**:
    - Research in HCI provides insights into designing systems that operate on human gestures (Nielsen, 1993). Our application aims to implement these principles by implementing a clean, responsive, and accessible virtual whiteboard for teachers to conduct live sessions with ease.

Existing Solutions and Improvements

Several existing solutions offer functionalities similar to our Smart Teaching Assistant application, including popular LMS platforms like Moodle and Blackboard, and specialized tools for automated grading. However, our project differentiates itself in several key ways:

1. **Targeted Functionality**:
    - Our application focuses specifically on classroom management, automated grading and interactive teaching sessions. This specialization allows us to provide a more streamlined and user-friendly experience for teachers.
2. **Ease of Use**:
    - Our application is designed with simplicity in mind. The user interface is intuitive, requiring minimal training for teachers to start using it effectively.
3. **Cost-Effectiveness**:
    - By using open-source tools and technologies, we are able to offer a cost-effective solution that can be easily adopted by educational institutions without significant financial investment.
4. **Flexibility and Customization**:
    - The architecture of our application allows for easy customization and scalability. Schools can tailor the application to meet their specific needs and integrate it with other systems if required.

Methodology

System Design

The Smart Teaching Assistant application is designed as a modular system with a clear separation of concerns, allowing for maintainability and scalability. The architecture consists of several key components:

1. **User Interface (UI)**:
    - Built using QML and Qt Quick Controls, the UI is designed to be intuitive and responsive.
    - The StackView component is used to navigate between different views, such as managing classes, students, quizzes, and whiteboard sessions.
2. **Backend Logic**:
    - Implemented in C++, the backend handles all the business logic, including database interactions, image processing for OMR, and managing the whiteboard, capturing video streams.

    - Signals and slots mechanism is used extensively to ensure smooth communication between the UI and the backend.
3. **Database**:
    - SQLite is used as the database engine to store persistent data, including information about students, classes, quizzes, marks and whiteboards.
    - The database schema is designed to ensure data integrity and efficient querying.
4. **Image Processing**:
    - OpenCV is utilized for capturing and processing images, particularly for OMR functionality and interactive whiteboards.
    - This component handles tasks such as capturing images from the webcam, processing them to detect marks, and analyzing quiz results.

Technologies Used

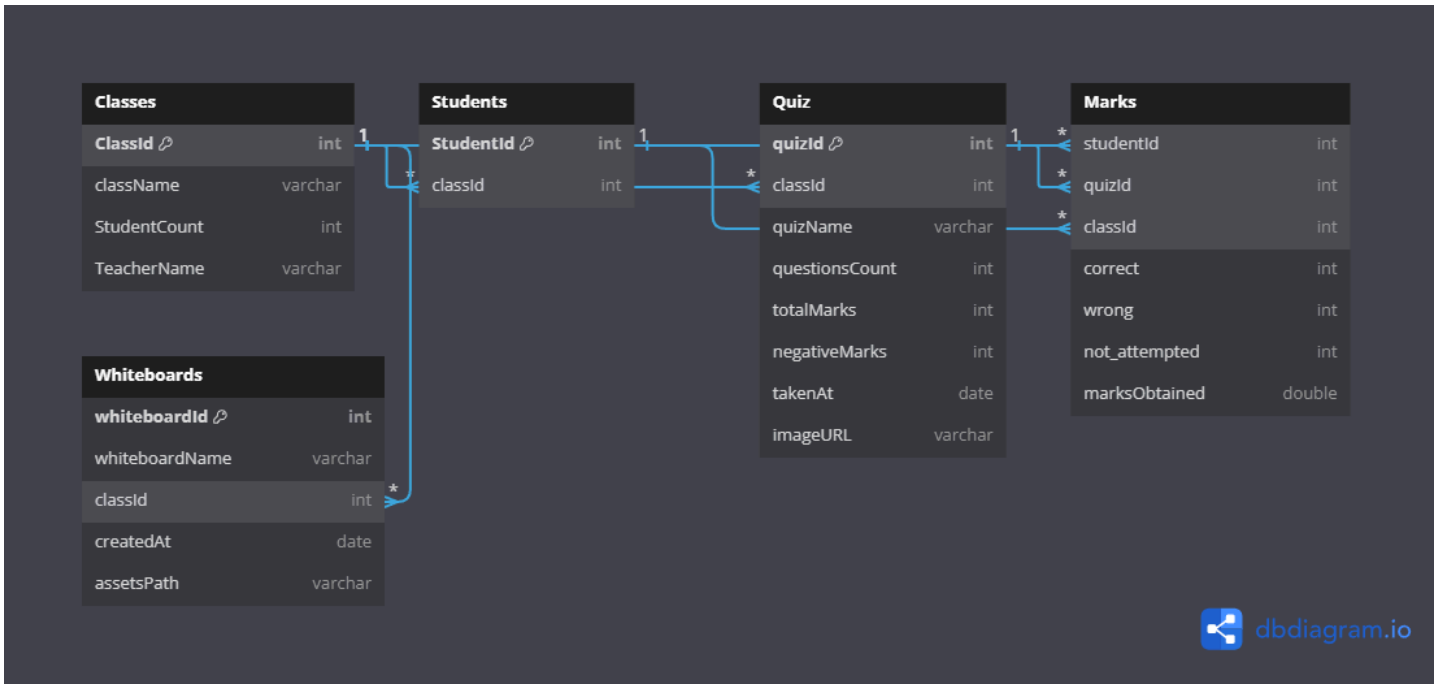
- **Programming Languages**: C++, QML
- **Frameworks**: Qt, Qt Quick Controls, Qt Widgets
- **Database**: SQLite
- **Libraries**: OpenCV for image processing
- **Tools**: Qt Creator for development, QMake for build automation

Database Design

The database schema consists of several tables designed to store the necessary information for managing the application:

1. **students**:
    - `student_id` (INTEGER)
    - `class_id` (INTEGER, FOREIGN KEY REFERENCES classes(id))
2. **classes**:
    - `id` (INTEGER PRIMARY KEY)
    - `className` (TEXT)
    - `studentCount` (INTEGER)
    - `teacherName` (TEXT)
    - `centerName` (TEXT)
3. **quizzes**:
    - `quiz_id` (INTEGER PRIMARY KEY)
    - `class_id` (INTEGER, FOREIGN KEY REFERENCES classes(id))
    - `quiz_name`(TEXT)
    - `total_marks` (INTEGER)
    - `questions_count` (INTEGER)
    - `negative_marking` (REAL)
    - `answer_key` (TEXT)
    - `test_paper_img_path` (TEXT)
    - `taken_at` (TEXT DEFAULT CURRENT TIMESTAMP)
    -

4. **marks**:
    - `student_id` (INTEGER, FOREIGN KEY REFERENCES students(student_id))
    - `quiz_id` (INTEGER, FOREIGN KEY REFERENCES quizzes(quiz_id))
    - `correct` (INTEGER)
    - `wrong` (INTEGER)
    - `not_attempted` (INTEGER)
    - `marks_obtained` (REAL)
    - PRIMARY KEY (`student_id`, `quiz_id`, `class_id`)
5. **Whiteboards**:
    - `whiteboard_id` (INTEGER PRIMARY KEY)
    - `whiteboard_name` (TEXT)
    - `class_id` (INTEGER)
    - `created_at` (TEXT DEFAULT CURRENT TIMESTAMP)
    - `assets_path` (TEXT)
    - FOREIGN KEY (`class_id`) REFERENCES classes(`id`)



Implementation Details

Main Features

1. **Class Management**:
    - Teachers can create, view, and manage classes.
    - Each class can have multiple students enrolled.
2. **Student Management**:
    - Teachers can add, view, and manage students within each class.
    - The application maintains a database of student information linked to their respective classes.
3. **Quiz Management**:
    - Teachers can create quizzes, specifying the associated class.

    - Quiz details, including the number of questions and marking scheme, are stored in the database.
4. **Whiteboard Management**:
    - A digital whiteboard allows teachers to draw and save snapshots of their sessions.
    - Snapshots can be saved with a user-defined name and are stored in the specified folder.
5. **Automated Quiz Grading**:
    - The application uses a webcam to capture images of OMR sheets.
    - OpenCV processes the images to detect marked answers and calculate the results based on the quiz configuration.
    - Results are stored in the database and can be updated if a quiz is re-evaluated.

Key Functionalities

Student and Class Management

The **Student and Class Management** functionality allows teachers to manage students and their respective classes. This is facilitated by the `DatabaseHandler` class, which interacts with the SQLite database to perform various operations. The class provides methods to insert, update, delete, and query data related to students, classes, quizzes and whiteboards. The UI components, created using QML, call these methods to update the database and ensure that the user interface reflects any changes made.

- **CRUD Operation on Classes**: The application allows teachers to add new classes, edit existing classes, delete a class and view the contents of a class by implementing `addClass`, `editClass, deleteClass` methods in the `DatabaseHandler` class that handle these operations efficiently and securely.
- **Querying Students**: The `getStudents` method retrieves a list of students in a specific class, allowing the UI to display this information to the user.

Whiteboard Management

The **Whiteboard Management** functionality provides a digital whiteboard for teachers to use during sessions. This feature is implemented using a combination of QML for the user interface and C++ for backend processing.

- **Drawing Operations**: The `whiteboardManager` object handles the drawing operations on the whiteboard. Users can draw on the whiteboard using various colors provided in the UI. The whiteboard also supports erasing drawing using erasers.
- **Clearing the Board**: The whiteboard can be cleared using a button in the UI, which calls the `clearWhiteboard` method in the `whiteboardManager`.
- **Saving Snapshots**: The whiteboard content can be saved as image files with the help of `saveSnapshot` utility method. When the user chooses to save a snapshot, a dialog prompts for a filename. The snapshot is then saved to the specified path, and its location is managed by the application.
- **Uploading Images**: Teachers can upload images to the whiteboard, allowing them to annotate or highlight specific parts of the image during a session. This feature uses a file dialog to select the image and the `whiteboardManager` to load it onto the whiteboard.

Quiz Marking

The **Quiz Marking** functionality automates the grading of quizzes using Optical Mark Recognition (OMR) technology. This involves capturing images of OMR sheets, processing them to detect marked answers, and storing the results in the database.

- **Video Stream Capture**: The `VideoStreamer` class is responsible for capturing images from a webcam. It uses the OpenCV library to access the webcam feed and capture images.
- **Image Processing**: The `OMRManager` class processes the captured images to detect and analyze marked answers. This class likely uses image processing techniques to identify the positions of marks on the OMR sheet and determine the selected answers.
- **Storing Quiz Results**: The `DatabaseHandler` class stores the quiz results in the `marks` table. The `uploadQuizMarks` method ensures that if a quiz result for a specific combination of quiz ID, class ID, and student ID already exists, it is overwritten with the new result.
- **Create and Grade Quizzes**: The application allows creating quizzes with details such as their name, marks, negative marking criteria and answer key. These are managed by `getQuizzes, addQuiz, uploadQuizMarks` methods in the `DatabaseHandler` class. The `uploadQuizMarks` method ensures that if a quiz result for a specific combination of quiz ID, class ID, and student ID already exists, it is overwritten with the new result.
- **Exporting Marks as CSV:** The application also allows exporting marks of any quiz in the form of CSV(Comma Separated Values) file using `exportMarksAsCSV` implemented in `DatabaseHandler` which makes quiz assessment and marks management even streamlined.

Implementation

Frontend Implementation

The welcome screen welcomes users with an aesthetic image on the left and a greeting message along with a get started image on the right. This design is achieved with the help of `RowLayout`

Welcome to Smart Teaching Assistant

Grade your students' tests using OMR and use Virtual Whiteboard to conduct live sessions.

Get Started

Welcome Screen

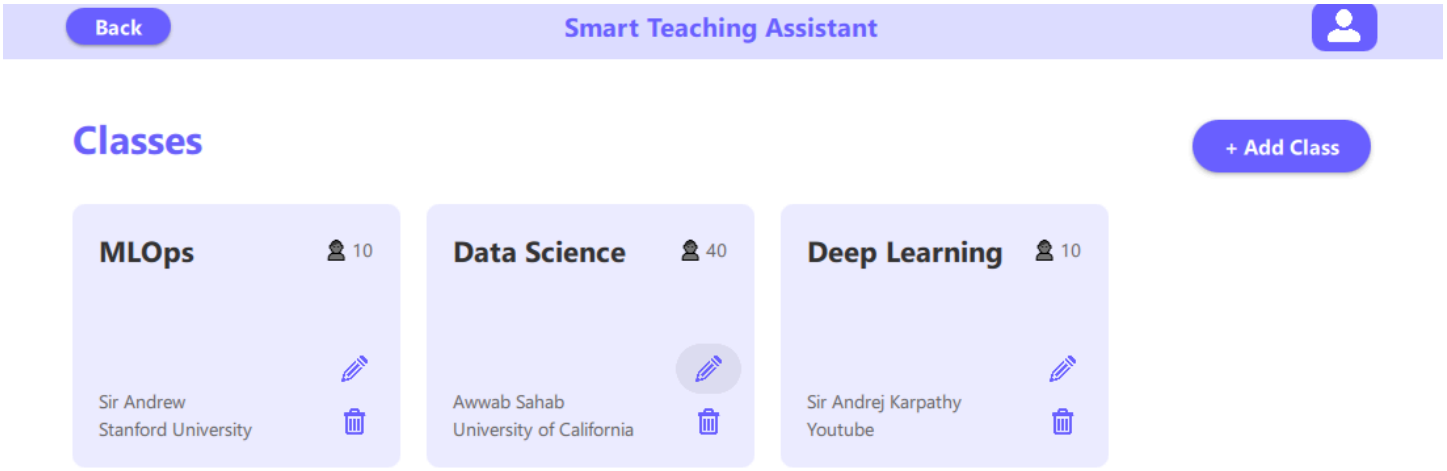
WelcomeScreen.qml

```
Rectangle {
    anchors.fill: parent
    color: "transparent"

    RowLayout {
        id: rowLayout
        anchors.fill: parent
        anchors.margins: 20

        Rectangle {
            Layout.fillHeight: true
            Layout.preferredWidth: parent.width / 2
            color: "transparent"

            Image {
                id: illustration
                anchors.centerIn: parent
                source: "welcome_illustration.svg"
                fillMode: Image.PreserveAspectFit
                width: parent.width * 0.4
                height: parent.height * 0.8
            }
        }

        ColumnLayout {
            Layout.fillHeight: true
            Layout.preferredWidth: parent.width / 2
            spacing: 20

            Text {
                text: "Welcome to Smart Teaching Assistant"
                font.pixelSize: 24
                Layout.fillWidth: true
                Layout.maximumWidth: parent.width
                wrapMode: Text.WordWrap
                Layout.alignment: Qt.AlignLeft
            }

            Text {
                text: qsTr("Grade your students' tests using OMR and use Virtual Whiteboard to conduct live sessions.")
                font.pixelSize: 16
                Layout.fillWidth: true
                Layout.maximumWidth: parent.width
                wrapMode: Text.WordWrap
                Layout.alignment: Qt.AlignLeft
            }

            Button {
                text: "Get Started"
                width: 150
                height: 40
                Material.background: "#6C63FF"

                contentItem: Text {
                    text: qsTr("Get Started")
                    color: "white"
                    font.pixelSize: 16
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                    verticalAlignment: Text.AlignVCenter
                }
                Layout.alignment: Qt.AlignVCenter | Qt.AlignLeft
                onClicked: stackView.push("classesScreen.qml")
            }
        }
    }

```

TitleBar.qml

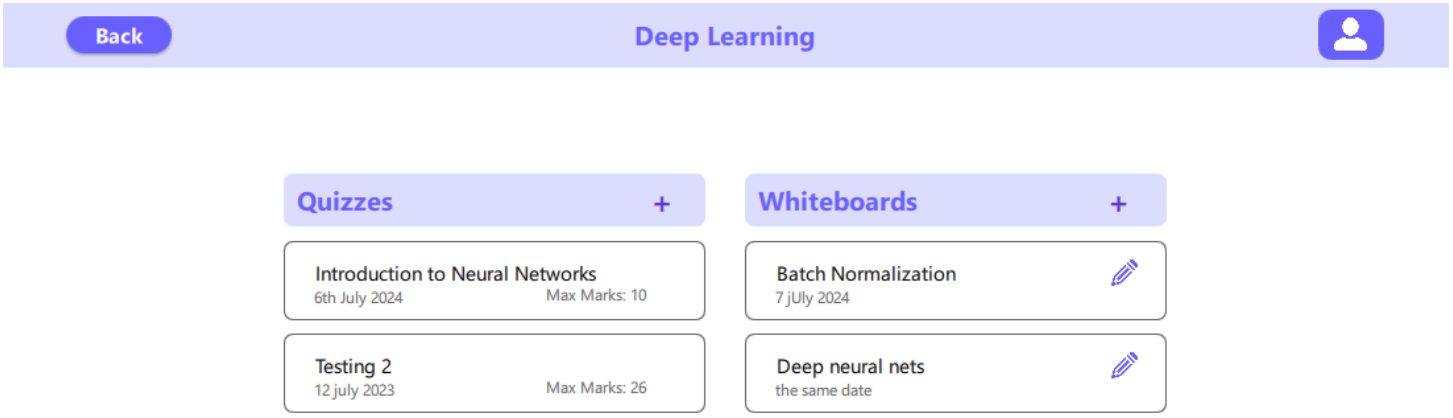
```qml
import QtQuick
import QtQuick.Controls.Material
import QtQuick.Controls
Rectangle {
    property string title : ""
    id : topBar
    height : 50
    color : "#E0E0FF"
    anchors {
        left : parent.left
        top : parent.top
        right : parent.right
    } // back button
    Button {
        text : qsTr("← Back")
        Material.background : "#6C63FF"
        width : 80
        height : 40
        contentItem : Text {
            text : qsTr("Back")
            color : "white"
            font.pixelSize : 16
            font.bold : true
            horizontalAlignment : Text.AlignHCenter
            verticalAlignment : Text.AlignVCenter
        }
        onClicked : {
            stackView.pop()
        }
        anchors.left : parent.left
        anchors.leftMargin : 50
        anchors.verticalCenter : parent.verticalCenter
    }
    Label {
        text : title
        color : "#6C63FF"
        font.pixelSize : 20
        font.bold : true
        anchors.horizontalCenter : parent.horizontalCenter
        anchors.verticalCenter : parent.verticalCenter
    } // profile icon
    Button {
        background : Rectangle {
            color : "#6C63FF"
            radius : 10
        }
        height : 50
        width : 50
        Image {
            source : "https://static-00.iconduck.com/assets.00/avatar-icon-256x256-lc2hm878.png"
            width : 25
            height : 25
            fillMode : Image.PreserveAspectFit
            anchors.centerIn : parent
        }
        anchors.right : parent.right
        anchors.rightMargin : 50
    }
}
```

Classes Screen gives details about classes the teacher has created. It provides buttons to edit and delete classes. The title bar above is totally dynamic which is coded once and adapts itself to match every screen.



Classes Screen

Clicking on any class redirects users to a new screen which displays the quizzes and whiteboards associated with that class in a nice dynamic way. All these quizzes and whiteboards are dynamically loaded from the database.



Quizzes & Whiteboards Screen

Back

Deep Learning

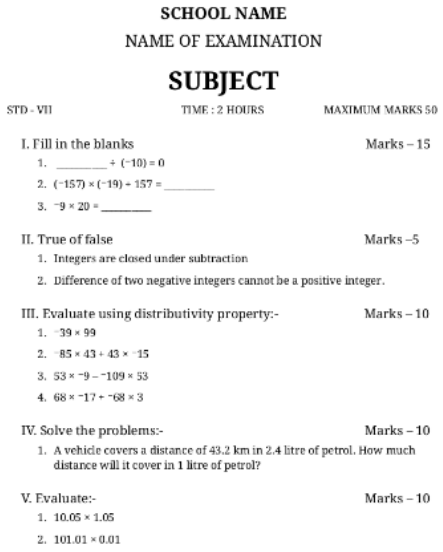
Introduction to Neural Networks

Max Marks: 10

Negative Marking: -1

Number of Questions: 10

Grade



SCHOOL NAME

NAME OF EXAMINATION

SUBJECT

STD - VII TIME : 2 HOURS MAXIMUM MARKS 50

I. Fill in the blanks Marks – 15

1. _____ ÷ (−10) = 0
2. (−157) × (−19) ÷ 157 = _____
3. −9 × 20 = _____

II. True of false Marks –5

1. Integers are closed under subtraction
2. Difference of two negative integers cannot be a positive integer.

III. Evaluate using distributivity property:- Marks – 10

1. −39 × 99
2. −85 × 43 + 43 × −15
3. 53 × −9 − −109 × 53
4. 68 × −17 + −68 × 3

IV. Solve the problems:- Marks – 10

1. A vehicle covers a distance of 43.2 km in 2.4 litre of petrol. How much distance will it cover in 1 litre of petrol?

V. Evaluate:- Marks – 10

1. 10.05 × 1.05
2. 101.01 × 0.01

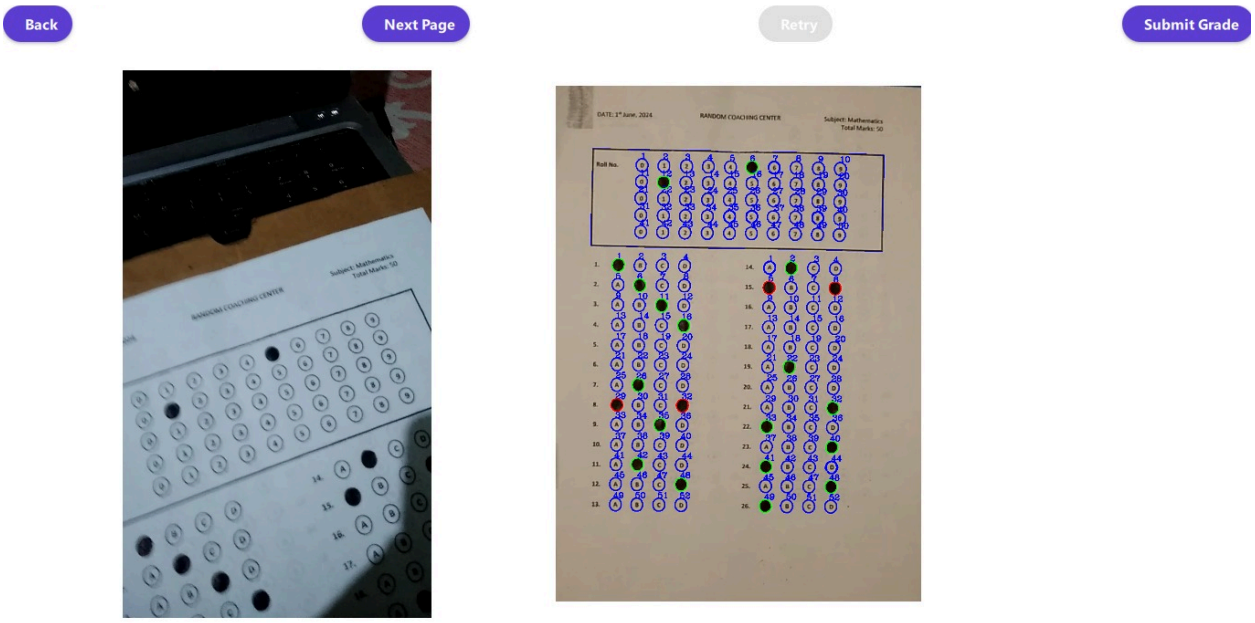
Marks Table

Export As CSV

Roll No	Correct	Wrong	Not Attempted	Obtained Marks
0	9	1	0	8
10	9	1	0	8
51	0	7	3	0
123	9	1	0	8
513	0	7	3	0

Individual Quiz Screen

The individual quiz screen provides details about the quiz clicked, its name, total marks, negative marking criteria, and other information. Along with that, the page also provides insights about the marks of each student. How many have they got correct, wrong, not attempted, and obtained overall.



Quiz Grading Screen

The Quiz Grading Screen lets users scan the omr sheet while showing their camera feed on the left side and the scanned image on the right side. This is also achieved using dynamic layouting techniques provided by QML. At the top, there are 4 buttons for going back, next page, retry and grade functions which call relevant functions when clicked.

main.qml

```qml
import QtQuick
import QtQuick.Layouts
import QtQuick.Controls
import QtQuick.Controls.Material

Window {
    id: root
    width: 1100
    height: 600
    visible: true
    title: qsTr("Smart Teaching Assistant")

    minimumHeight: 430
    minimumWidth: 750

    StackView {
        id: stackView
        initialItem: "welcomeScreen.qml"
        anchors.fill: parent

        onCurrentItemChanged: {
            omrManager.connectOMRPage(currentItem)
        }
    }
}
```

gradeQuiz.qml

```
Item {
    id: omrpage
    property string pageId: "omrpage"
    property int pNo: 1
    property int quizId
    property int classId
    property string ansKey
    property double negative_marking

    width: parent.width
    height: parent.height

    signal imageCaptured(var img, bool firstPage, string ansKey, double ne

    Component.onCompleted: {
        videoStreamerOMR.startStream(1)
    }

    Component.onDestruction: {
        videoStreamerOMR.stopStream()
    }

    DatabaseHandler {
        id: dbhandler
    }

    ColumnLayout {
        width: parent.width
        RowLayout {
            Layout.fillWidth: true
            spacing: (parent.width - 100) / 4

            Button {
                Material.background: "#5D3FD3"
                width: 80
                height: 40
                anchors.top: omrSpace.bottom
                contentItem: Text {
                    text: qsTr("Back")
                    color: "white"
                    font.pixelSize: 16
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                    verticalAlignment: Text.AlignVCenter
                }
                onClicked: {
                    console.log("Back Clicked!")
                    omrManager.retry()
                    stackView.pop()
                }
            }

            Button {
                Material.background: "#5D3FD3"
                width: 80
                height: 40
                contentItem: Text {
                    text: qsTr("Next Page")
                    color: "white"
                    font.pixelSize: 16
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                    verticalAlignment: Text.AlignVCenter
                }
                onClicked: {
                    console.log("Next Page Clicked!")
                    pNo += 1
                }
            }

            Button {
                id: retryButton
                Material.background: "#5D3FD3"
                width: 80
                height: 40
                contentItem: Text {
                    text: qsTr("Retry")
                    color: "white"
                    font.pixelSize: 16
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                    verticalAlignment: Text.AlignVCenter
                }
                onClicked: {
                    pNo = 1
                    console.log("Retry Clicked!")
                    omrManager.retry()
                }
            }
```

gradeQuiz.qml

```
            Button {
                Material.background: "#5D3FD3"
                width: 80
                height: 40
                contentItem: Text {
                    text: qsTr("Submit Grade")
                    color: "white"
                    font.pixelSize: 16
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                    verticalAlignment: Text.AlignVCenter
                }
                onClicked: {
                    console.log("Submit Grade Clicked!")
                    var result = omrManager.returnGrade()
                    var response = dbhandler.uploadQuizMarks(
                                quizId, classId, result["rollNo"],
                                result["correct"], result["wrong"],
                                result["unattempted"], result["obtained"])
                    console.log("Obtained marks", result["obtained"])
                    if (!response) {
                        console.log("Failed to upload quiz marks")
                    } else {
                        console.log("Successfully uploaded quiz marks")
                    }

                    pNo = 1
                }
            }
        }

        RowLayout {
            Layout.fillWidth: true

            Image {
                id: omrSpace
                fillMode: Image.PreserveAspectFit
                source: "image://omr/image"
                cache: false
                Layout.preferredWidth: 600
                Layout.preferredHeight: 900
                transform: Rotation {
                    id: rotateImagePhoto
                    angle: 90
                    origin.x: omrSpace.width / 2
                    origin.y: omrSpace.height / 2
                }

                Layout.topMargin: -130
                Layout.alignment: Qt.AlignLeft

                function reload() {
                    source = "image://omr/image?id=" + Date.now()
                }
            }

            Image {
                id: scannedSpace
                fillMode: Image.PreserveAspectFit
                source: "image://scanned/image"
                cache: false
                Layout.preferredWidth: 400
                Layout.preferredHeight: 600

                Layout.topMargin: -130
                Layout.alignment: Qt.AlignRight

                function reload() {
                    source = "image://scanned/image?id=" + Date.now()
                }
            }
        }
    }

    Connections {
        target: OMRImageProvider
        function onImageChanged() {
            omrSpace.reload()
        }
    }

    Connections {
        target: scannedImageProvider
        function onImageChanged() {
            scannedSpace.reload()
        }
    }

    Keys.onPressed: {
        if (event.key === Qt.Key_S) {
            console.log(ansKey)
            var img = videoStreamerOMR.getCurrentFrame()
            if (pNo == 1) {
                imageCaptured(img, true, ansKey, negative_marking)
            } else {
                imageCaptured(img, false, ansKey, negative_marking)
            }
        }
    }
}
```

Backend Implementation

1. `OMRManager` Class

The `OMRManager` class is responsible for processing Optical Mark Recognition (OMR) sheets to detect and analyze marked answers. This class typically interacts with OpenCV for image processing and handles the logic for interpreting the marked answers.

Key Responsibilities:

- Image Processing: Uses OpenCV functions to preprocess the image, such as converting to grayscale, thresholding, and detecting contours.
- Mark Detection: Identifies marked answers by analyzing the processed image. This involves detecting the location of the marks and determining which options are selected.
- Result Extraction: Extracts the answers from the detected marks and prepares the data for storage or further analysis.

OMRmanager.h

```h
#ifndef OMRMANAGER_H
#define OMRMANAGER_H

#include <QObject>
#include <QVariant>
#include <QImage>
#include <QDebug>
#include <opencv2/opencv.hpp>
# include <QVariantMap>
# include <vector>
# include "imutils.h"

class OMRmanager : public QObject
{
    Q_OBJECT
public:
    explicit OMRmanager(QObject *parent = nullptr);
    cv::Mat detectPaper(cv::Mat& img);
    std::vector<std::vector<std::vector<cv::Point>>> getCircles(cv::Mat img, cv::Mat& imgCopy, int x, int y, bool rollNo = false);
    std::vector<int> getSelectedCircles(cv::Mat& img, std::vector<std::vector<std::vector<cv::Point>>>& circles, cv::Mat& imgCopy,
int x, int y, bool rollNo = false);
    int getRollNo(cv::Mat& img, cv::Mat& imgCopy);
    std::vector<int> getSelectedOptions(cv::Mat& img, cv::Mat& imgCopy, bool firstPage = false);
    Q_INVOKABLE QVariantMap returnGrade();
    Q_INVOKABLE void retry();

public slots:
    void startOMR(const QVariant &imgVar, const bool firstPage, const QString ansKey, const double negative_marking);
    void connectOMRPage(QObject* currentItem);

signals:
    void newScannedImage(const QImage img);

private:
    QImage scannedImage;
    QVariantMap result;
    std::vector<int> resultVector;
    QString ansKey;
    int rollNo;
    double negative_marking;
};

#endif // OMRMANAGER_H

```

2. `DatabaseHandler` Class

The `DatabaseHandler` class manages all interactions with the SQLite database. It provides methods to perform CRUD (Create, Read, Update, Delete) operations on the database tables, ensuring that data is stored and retrieved efficiently.

Key Responsibilities:

- **Connection Management:** Establishes and manages the connection to the SQLite database.
- **Table Management:** Creates necessary tables such as `students`, `classes`, `quizzes`, and `marks`.
- **Data Operations:** Provides methods to insert, update, delete, and query data. For example:
    - **uploadQuizMarks:** Inserts or updates quiz marks in the `marks` table, ensuring that existing records are overwritten if the `quizId`, `classId`, and `studentId` already exist.

DatabaseHandler.h

```h
#ifndef DATABASEHANDLER_H
#define DATABASEHANDLER_H

#include <QObject> // mandatory for every custom class
#include <QSqlDatabase> // for database operations
#include <QSqlQuery> // for running sql queries
#include <QVariantList> // its like a list
#include <string>
#include <QFile>

class DatabaseHandler : public QObject
{
    Q_OBJECT
public:
    explicit DatabaseHandler(QObject *parent = nullptr);
    Q_INVOKABLE QVariantList getClasses();
    Q_INVOKABLE bool addClass(const QString &className, const int &studentCount, const QString &teacherName, const QString &centerName);
    Q_INVOKABLE bool editClass(const int iD, const QString &className, const int &studentCount, const QString &teacherName, const QString &centerName);
    Q_INVOKABLE bool deleteClass(const int id);
    Q_INVOKABLE QVariantList getWhiteboards(const int &classId);
    Q_INVOKABLE bool addQuiz(const int &classId, const QString &title, const QString &date, const int &questionsCount, const int &totalMarks, const double &negativeMarks, const QString &answerKey, const QString &imagePath);
    Q_INVOKABLE QVariantList getQuizzes(const int &classId);
    Q_INVOKABLE bool addWhiteboard(const int &classId, const QString &whiteboardName, const QString &created_at, const QString &folderPath);
    Q_INVOKABLE bool uploadQuizMarks(const int &quizId, const int &classId, const int &studentId, const int &correct, const int &wrong, const int &unattempted, const double &obtained);
    Q_INVOKABLE QVariantList getMarks(const int& quizId, const int &classId);
    Q_INVOKABLE void exportMarksAsCSV(const QString &filePath, const int &quizId, const int &classId);

signals:
    void classDeleted(int id);

private:
    QSqlDatabase db;

};

#endif // DATABASEHANDLER_H

```

3. `VideoStreamer` Class

The `VideoStreamer` class handles video capture from a webcam using OpenCV and streams the captured frames as `QImage` objects. This class ensures real-time video streaming for various purposes, such as capturing images for OMR processing.

Key Responsibilities:

- **Video Capture:** Opens the webcam and captures frames using OpenCV.
- **Frame Processing:** Converts captured frames from `cv::Mat` to `QImage`.
- **Streaming:** Emits signals with the current frame for real-time display or further processing.

4. `OpenCVImageProvider` Class

The `OpenCVImageProvider` class is a custom image provider for QML, which supplies images processed by OpenCV. This class allows the QML frontend to request and display images processed in the C++ backend.

Key Responsibilities:

- **Image Management:** Stores the current image to be provided to the QML layer.
- **Image Update:** Provides a method to update the stored image and emits a signal when the image changes.
- **Image Request:** Implements the `requestImage` method to supply the current image to the QML layer.

```cpp
#ifndef VIDEOSTREAMER_H
#define VIDEOSTREAMER_H

#include <QObject>
#include <QTimer>
#include <opencv2/opencv.hpp>
#include <opencv2/videoio.hpp>
#include <QImage>
#include <iostream>

class VideoStreamer : public QObject
{
    Q_OBJECT
public:
    VideoStreamer();
    ~VideoStreamer();

public:
    void streamVideo();
    Q_INVOKABLE void startStream(int  camera);
    Q_INVOKABLE void stopStream();
    Q_INVOKABLE QImage getCurrentFrame() const;

private:
    cv::Mat frame;
    cv::VideoCapture cap;
    QTimer tUpdate;

signals:
    void newImage(QImage img);
};

#endif // VIDEOSTREAMER_H

```

```cpp
#ifndef OPENCVIMAGEPROVIDER_H
#define OPENCVIMAGEPROVIDER_H

#include <QImage>
#include <QQuickImageProvider>


class OpenCVImageProvider : public QQuickImageProvider
{
    Q_OBJECT

public:
    OpenCVImageProvider();
    QImage requestImage(const QString &id, QSize *size,
   const QSize &requestedSize) override;

public slots:
    void updateImage(const QImage &img);

signals:
    void imageChanged();

private:
    QImage image;
};

#endif // OPENCVIMAGEPROVIDER_H

```

Data Flow and Signal-Slot Implementation

In the Smart Teaching Assistant application, the data flow is managed efficiently through Qt's signal-slot mechanism.

1. Video Streaming and Whiteboard Management

- **VideoStreamer Class:**
  - **Start Streaming:** The `startStream` method initializes the webcam and begins capturing frames.
  - **Signal Emission:** The `streamVideo` method captures frames continuously and emits the `newImage(QImage)` signal with the current frame as a `QImage`.
  - **Frame Processing:** The `newImage` signal is connected to the `WhiteboardManager` class for frame processing.
- **WhiteboardManager Class:**
  - **Process Frame:** The `processFrame` method in the `WhiteboardManager` processes the incoming frames and emits the `newWeightedImage` signal with the processed image.
  - **Update Image Provider:** This signal is connected to the `OpenCVImageProvider` to update the image displayed on the whiteboard.

2. OMR Processing

- **VideoStreamer Class:**
  - **OMR Image Capture:** A separate `VideoStreamer` instance captures images specifically for OMR processing.
  - **Signal Emission:** The `newImage` signal is emitted with the captured frame.
- **OMRManager Class:**
  - **Process Image:** The `OMRManager` processes the captured image to detect marked answers.
  - **Update Scanned Image:** The `newScannedImage` signal is emitted with the processed image.

3. Image Providers

- **OpenCVImageProvider Class:**
  - **Update Image:** The `updateImage` method updates the internal image and emits the `imageChanged` signal.
  - **Request Image:** QML components request the image through the `requestImage` method, which returns the current image.

4. QML Connections

- **Whiteboard Image Reload:**
  - **Image Changed:** The `onImageChanged` function in QML listens to the `imageChanged` signal from the `whiteboardImageProvider` and reloads the whiteboard image.
- **OMR Image Reload:**
  - **Image Changed:** The `onImageChanged` function in QML listens to the `imageChanged` signal from the `OMRImageProvider` and reloads the OMR image.
- **Scanned Image Reload:**
  - **Image Changed:** The `onImageChanged` function in QML listens to the `imageChanged` signal from the `scannedImageProvider` and reloads the scanned image.

Results

Outcomes

The Smart Teaching Assistant application was successfully developed and implemented, meeting all the initially defined requirements. The key functionalities, including student and class management, whiteboard management, and quiz marking, were tested and validated for robustness and performance. The application demonstrated high accuracy in OMR processing and efficient data management.

Testing and Validation

- **Unit Testing:** Individual components were tested to ensure correct functionality.
- **Integration Testing:** Integrated components were tested to verify interactions and data flow.

Discussion

Challenges and Solutions

1. **Challenge:** Ensuring real-time performance for the whiteboard feature.
    - **Solution:** Optimized the image processing pipeline and utilized efficient data structures.
2. **Challenge:** Handling varying image qualities in the OMR process.
    - **Solution:** Implemented preprocessing techniques such as image normalization and thresholding to improve OMR accuracy.

Limitations

- The current implementation requires a high-resolution camera for accurate OMR processing.
- The user interface could be further optimized for better responsiveness on lower-end devices.

Conclusion

The Smart Teaching Assistant application provides an effective solution for teachers, integrating class management, interactive whiteboards, and automated quiz marking into a single platform. The project demonstrates the potential of combining modern technologies such as Qt, OpenCV, and SQL databases to streamline educational tasks. This application not only enhances productivity but also improves the accuracy and efficiency of educational processes.

Future Work

- **Enhanced User Interface:** Further refinement of the UI for better user experience on various devices.
- **Mobile Application:** Development of a mobile version to increase accessibility for teachers on the go.
- **Advanced Analytics:** Integration of advanced data analytics to provide insights into student performance and class trends.
- **Cloud Integration:** Implementing cloud storage and synchronization features to enable seamless data access across different devices.

References

1. https://doc.qt.io/
2. https://docs.opencv.org/4.x/
3. https://www.researchgate.net/publication/330977246_An_Image_Processing_Oriented_Optical_Mark_Recognition_and_Evaluation_System
4. https://www.researchgate.net/publication/376376773_Virtual_white_board
5. https://www.scribd.com/document/673065048/Batch-13-Air-Canvas-Using-Python-Opencv-1
6. https://pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/