DEGICHI QUIZ
FINAL SEMESTER EXAM PROJECT REPORT

(Prepared to fulfill the requirements for passing the Final Exam Semester 1)

By :

Dentar Muhammad Ababilla (25031554277)

Ghita Natasha Putri (25031554023)

Orchidea Savira Putri Arief Dhidin (25031554174)

Lecturer:

Hasanuddin Al-Habib, S.Si., M.Si

Siska Puspitaningsih, S.Kom., M.Kom

STUDY PROGRAM DATA SCIENCE
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
STATE UNIVERSITY OF SURABAYA
2025

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Degichi Quiz is a digital quiz platform developed to help students strengthen their abilities through a wide range of questions from various subject areas. By offering diverse, challenging, and informative question sets, this project aims to enhance students' understanding and support independent learning in a more structured way.

In addition to improving learning quality, *Degichi Quiz* is designed to minimize cheating during quiz sessions. Through monitoring and activity-restriction features similar to *Exambro*, the platform ensures a more honest, secure, and objective evaluation process.

With a modern, user-friendly interface, *Degichi Quiz* provides an interactive quiz experience that encourages a disciplined and responsible learning environment. This project is expected to serve as an effective evaluation tool while supporting students' overall skill development.

1.2 PROBLEM STATEMENT

Many students in Indonesia struggle to develop their academic abilities effectively due to limited access to interactive and secure digital evaluation tools. Traditional quiz methods often lack engaging features and are vulnerable to cheating, which reduces the accuracy of assessments and weakens students' learning outcomes. As a result, students may have low motivation, unreliable performance results, and limited opportunities to practice their knowledge in a disciplined environment.

To address this issue, we developed *Degichi Quiz,* a modern digital quiz platform designed to provide an interactive, structured, and secure learning experience. By offering diverse question sets across multiple subjects and integrating anti-cheating features similar to *Exambro*, this platform ensures a fair and focused evaluation process. *Degichi Quiz* aims to help students improve their understanding, strengthen their discipline, and build confidence while fostering a more honest, engaging, and effective approach to academic assessment.

1.3 OBJECTIVES

The purpose of making this *Degichi Quiz* includes:

1.3.1 To create a user-friendly digital quiz platform that is accessible and easy for students to use.

1.3.2 To implement secure and efficient systems that support question delivery and minimize cheating during quiz sessions.

1.3.3 To design a modern, intuitive, and visually appealing interface that improves students' quiz-taking experience.

1.3.4 To provide an interactive learning and evaluation tool that helps students strengthen their academic skills across various subjects.

1.3.5 To promote honest, disciplined, and independent learning through a controlled and well-structured digital assessment environment

CHAPTER 2

ANALYSIS AND DESIGN

2.1 APPLICATION NECESSITY ANALYSIS

The Degichi Quiz application is developed as a modern and secure digital evaluation platform that promotes independent learning. The necessity of this application can be summarized into three main needs as follows:

2.1.1 Interactive & Engaging Digital Assessment

- Digital and gamified quiz platforms have been proven to increase student engagement and motivation. A study published in the *International Journal of Educational Technology in Higher Education* (2018) found that adaptive gamified quizzes helped students stay more engaged and improved learning outcomes.
- Degichi Quiz supports step-by-step question presentation, instant feedback, and modern interface design. These features create an active learning environment that encourages students to reflect on their answers, understand mistakes, and remain focused throughout the quiz.
- Such interactive systems transform assessments from simply “answering questions” into a more meaningful learning experience, helping students strengthen conceptual understanding and develop higher-order thinking skills.

2.1.2 Secure, Anti-Cheating, and Reliable Evaluation System

- Research shows that academic dishonesty increases significantly in online examinations without proper monitoring or preventive measures. A systematic review in the *Journal of Academic Ethics* (2023) revealed that cheating during online exams can reach **30–50%**, depending on exam type and supervision levels.
- To reduce dishonesty, Degichi Quiz integrates essential security features such as session authentication, time-limited assessments, restricted navigation, and activity monitoring. These mechanisms help create fairer, more objective evaluations.

- With reliable system stability and secure data handling, teachers can trust the results presented by the platform, making the evaluation process more credible and accountable.

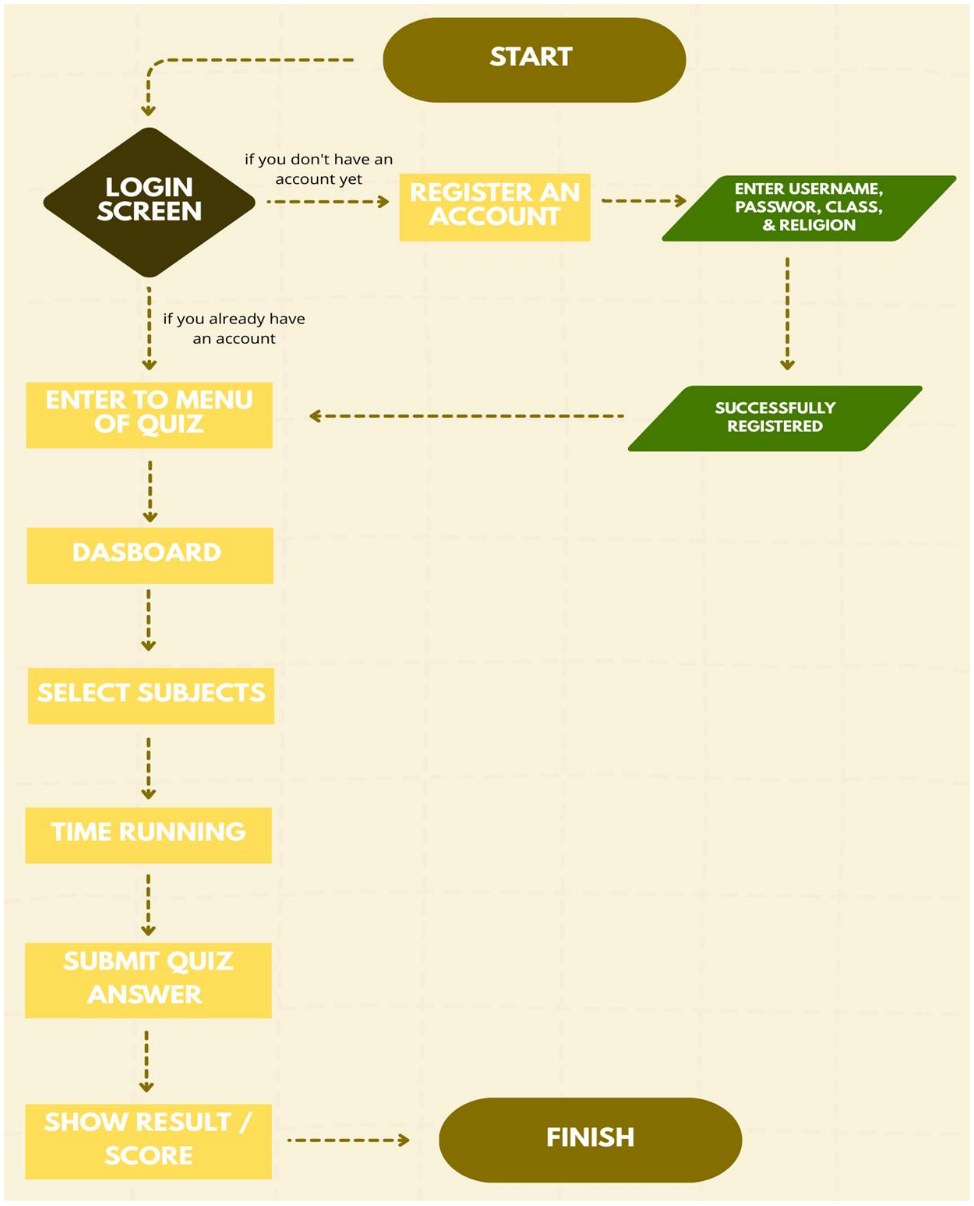
2.2.3 Data-Driven Learning & Independent Study Support

- According to the UNESCO Global Education Monitoring Report (2023), data-driven evaluation systems significantly improve the effectiveness of learning because they allow educators to track progress and identify learning gaps more accurately.
- Degichi Quiz stores comprehensive information such as quiz scores, learning progress, and individual performance statistics. These analytics help teachers design more targeted instruction and help students understand their strengths and weaknesses.
- The availability of progress data also supports self-paced and independent learning, enabling students to revisit topics, retry quizzes, and build confidence without external pressure.

2.2 FLOWCHART

As a digital evaluation medium, Degichi Quiz offers major advantages in terms of interactivity, security, and independent learning support. The system not only measures learning outcomes but also creates a more active, honest, and structured evaluation experience for students. To ensure that the application's process can be understood clearly, a workflow diagram is needed to illustrate how users interact with the system from beginning to end.

The flowchart below presents the main operational flow of the application in a structured manner.



2.3 USER INTERFACE DESIGN SKETCH

2.3.1 Login Page

Components:

- Dark academic background with formulas and doodles.
- Center title: “DEGICHI”
- Input Form:
  - Username
  - Password
  - Show Password Checkbox
- Buttons:
  - “Masuk”
  - “Daftar”

2.3.2 Registration Page

Components:

- Same background theme as login.
- Tittle: “Daftar Akun Baru”
- Input Form:
  - Username
  - Password
  - Konfirmasi Password
  - Kelas
  - Agama
- Button: “Daftar Sekarang”

2.3.4 Selection Category

Components:

- Title: “Pilih Mata Pelajaran”
- Six Subject Cards:
  - Matematika
  - Fisika
  - Kimia
  - Biologi
  - Bahasa Indonesia
  - Bahasa Inggris
  - PKN
  - Sosiologi
  - Ekonomi
  - Sejarah
  - Matematika Lanjut
  - Olahraga
  - Agama
- Buttons-right: “Keluar”

2.3.5 Main Quiz

Components:

- Top bar:
  - Timer
  - XP
  - Level
  - Menu icon.
- Header: Question X of 20
- Question text and four answer options (A–D).
- Penalties: –5 min, –10 min, –20 min
- Botton: “Soal Berikutnya“

2.3.6 Quiz Result

Components:

- Title: “Quiz Completed!”
- Score box:
  - Final Score
  - Time Left
  - XP Obtained
  - Level
  - Power-up used
- Buttons:
  - “Cetak PDF”

➓ “Kembali ke Dasboard”

2.3.7 Result in PDF

- Result like a raport

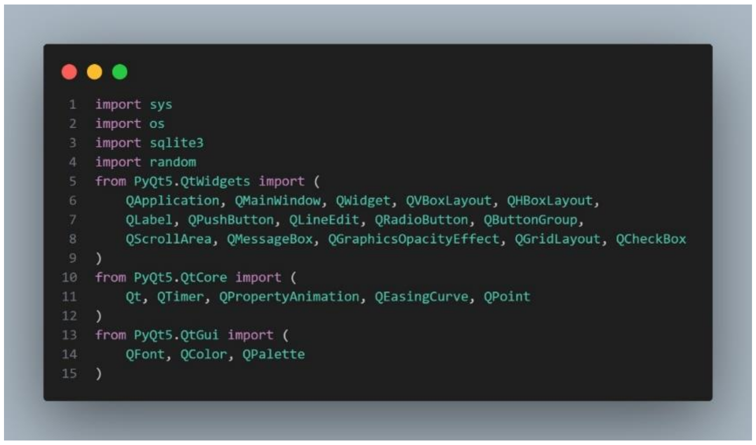
CHAPTER 3

IMPLEMENTATION

3.1 CODE EXPLANATION

Implementation of the **Degichi – Digital Exam & Gamified Cbt** application is carried out using the Python programming language, the PyQt5 framework for building the graphical user interface (GUI), and SQLite as the storage system for user data and XP progression. The code consists of several main components, including theme configuration, database system, UI components, question loader, power-up system, question navigation mechanism, time management, and exam result generation in PDF format.

The following section provides a systematic explanation of the code structure and its functions



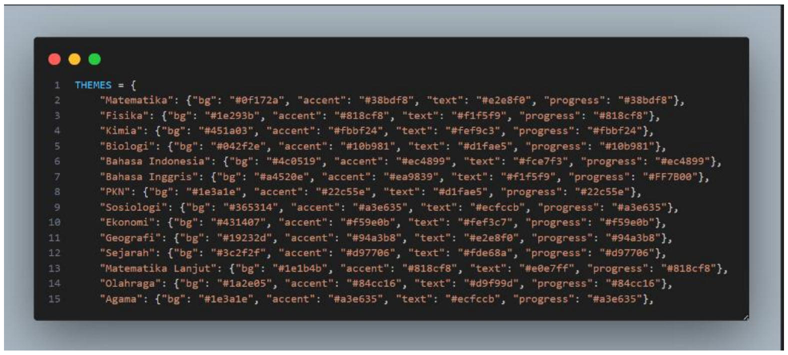
```python
import sys
import os
import sqlite3
import random
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QLabel, QPushButton, QLineEdit, QRadioButton, QButtonGroup,
    QScrollArea, QMessageBox, QGraphicsOpacityEffect, QGridLayout, QCheckBox
)
from PyQt5.QtCore import (
    Qt, QTimer, QPropertyAnimation, QEasingCurve, QPoint
)
from PyQt5.QtGui import (
    QFont, QColor, QPalette
)
```

Gambar 3. 1 Libraries for Degichi Quiz
Source: Personal documentation

Importing Libraries:

- **PyQt5** – used to build the application interface such as windows, buttons, layouts, radio buttons, message boxes, scroll areas, animations, timers, and visual effects.
- **sqlite3** – used to store and manage user data, including username, password, XP, class, and religion.
- **random** – used to shuffle questions and generate random visual effects such as confetti.
- **reportlab** – used to generate PDF files for downloadable exam reports.
- **datetime** – used to add timestamps to reports and activity logs.
- **os & sys** – used to manage file paths, access assets, read question files, and control application execution.

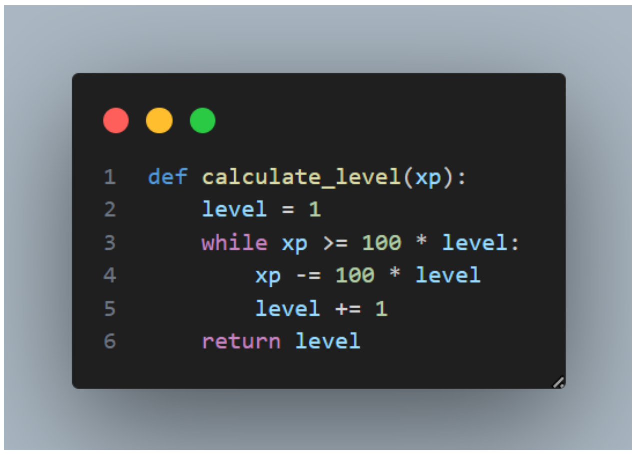
- **QTimer, QPropertyAnimation, QEasingCurve** – used to handle countdown timers and smooth UI animations.
- **QFont, QColor, QPalette** – used to style the application's visual theme, colors, and typography.



```python
THEMES = {
    "Matematika": {"bg": "#0f172a", "accent": "#38bdf8", "text": "#e2e8f0", "progress": "#38bdf8"},
    "Fisika": {"bg": "#1e293b", "accent": "#818cf8", "text": "#f1f5f9", "progress": "#818cf8"},
    "Kimia": {"bg": "#451a03", "accent": "#fbbf24", "text": "#fef9c3", "progress": "#fbbf24"},
    "Biologi": {"bg": "#042f2e", "accent": "#10b981", "text": "#d1fae5", "progress": "#10b981"},
    "Bahasa Indonesia": {"bg": "#4c0519", "accent": "#ec4899", "text": "#fce7f3", "progress": "#ec4899"},
    "Bahasa Inggris": {"bg": "#a4520e", "accent": "#ea9839", "text": "#f1f5f9", "progress": "#FF7B00"},
    "PKN": {"bg": "#1e3a1e", "accent": "#22c55e", "text": "#d1fae5", "progress": "#22c55e"},
    "Sosiologi": {"bg": "#365314", "accent": "#a3e635", "text": "#ecfccb", "progress": "#a3e635"},
    "Ekonomi": {"bg": "#431407", "accent": "#f59e0b", "text": "#fef3c7", "progress": "#f59e0b"},
    "Geografi": {"bg": "#19232d", "accent": "#94a3b8", "text": "#e2e8f0", "progress": "#94a3b8"},
    "Sejarah": {"bg": "#3c2f2f", "accent": "#d97706", "text": "#fde68a", "progress": "#d97706"},
    "Matematika Lanjut": {"bg": "#1e1b4b", "accent": "#818cf8", "text": "#e0e7ff", "progress": "#818cf8"},
    "Olahraga": {"bg": "#1a2e05", "accent": "#84cc16", "text": "#d9f99d", "progress": "#84cc16"},
    "Agama": {"bg": "#1e3a1e", "accent": "#a3e635", "text": "#ecfccb", "progress": "#a3e635"},
```

Gambar 3. 2 The code of Degichi Quiz
Source: Personal documentation

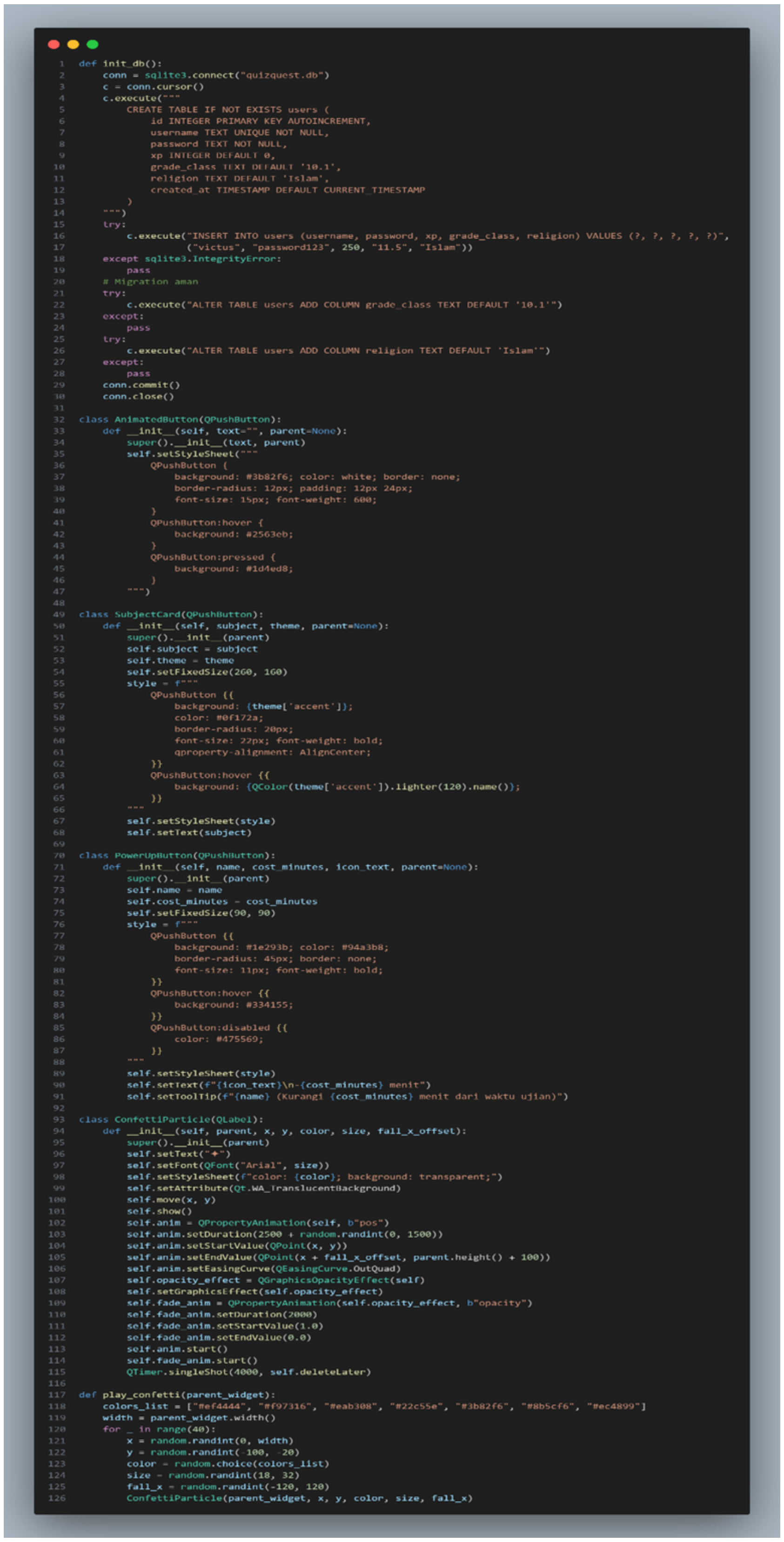
The code snippet contains a constant dictionary named **THEMES**. This dictionary is used to store theme configurations for different school subjects in the Degichi Quiz project. Each subject is assigned visual styling values including background color, accent color, text color, and progress indicator color. These color styles help ensure that each selected subject displays a unique and consistent UI appearance throughout the application.



```python
def calculate_level(xp):
    level = 1
    while xp >= 100 * level:
        xp -= 100 * level
        level += 1
    return level
```

Gambar 3. 3 The code of Degichi Quiz
Source: Personal documentation

This function calculates a player's level based on their XP. It starts from level 1 and keeps increasing the level as long as the XP is enough to pay the required amount (100 × current level). Each level-up reduces XP by that amount. When the XP is no longer sufficient, the function returns the final level.



```python
def init_db():
    conn = sqlite3.connect("quizquest.db")
    c = conn.cursor()
    c.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            xp INTEGER DEFAULT 0,
            grade_class TEXT DEFAULT '10.1',
            religion TEXT DEFAULT 'Islam',
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)
    try:
        c.execute("INSERT INTO users (username, password, xp, grade_class, religion) VALUES (?, ?, ?, ?, ?)",
                  ("victus", "password123", 250, "11.5", "Islam"))
    except sqlite3.IntegrityError:
        pass
    # Migration aman
    try:
        c.execute("ALTER TABLE users ADD COLUMN grade_class TEXT DEFAULT '10.1'")
    except:
        pass
    try:
        c.execute("ALTER TABLE users ADD COLUMN religion TEXT DEFAULT 'Islam'")
    except:
        pass
    conn.commit()
    conn.close()

class AnimatedButton(QPushButton):
    def __init__(self, text="", parent=None):
        super().__init__(text, parent)
        self.setStyleSheet("""
            QPushButton {
                background: #3b82f6; color: white; border: none;
                border-radius: 12px; padding: 12px 24px;
                font-size: 15px; font-weight: 600;
            }
            QPushButton:hover {
                background: #2563eb;
            }
            QPushButton:pressed {
                background: #1d4ed8;
            }
        """)

class SubjectCard(QPushButton):
    def __init__(self, subject, theme, parent=None):
        super().__init__(parent)
        self.subject = subject
        self.theme = theme
        self.setFixedSize(260, 160)
        style = f"""
            QPushButton {{
                background: {theme['accent']};
                color: #0f172a;
                border-radius: 20px;
                font-size: 22px; font-weight: bold;
                qproperty-alignment: AlignCenter;
            }}
            QPushButton:hover {{
                background: {QColor(theme['accent']).lighter(120).name()};
            }}
        """
        self.setStyleSheet(style)
        self.setText(subject)

class PowerUpButton(QPushButton):
    def __init__(self, name, cost_minutes, icon_text, parent=None):
        super().__init__(parent)
        self.name = name
        self.cost_minutes = cost_minutes
        self.setFixedSize(90, 90)
        style = f"""
            QPushButton {{
                background: #1e293b; color: #94a3b8;
                border-radius: 45px; border: none;
                font-size: 11px; font-weight: bold;
            }}
            QPushButton:hover {{
                background: #334155;
            }}
            QPushButton:disabled {{
                color: #475569;
            }}
        """
        self.setStyleSheet(style)
        self.setText(f"{icon_text}\n-{cost_minutes} menit")
        self.setToolTip(f"{name} (Kurangi {cost_minutes} menit dari waktu ujian)")

class ConfettiParticle(QLabel):
    def __init__(self, parent, x, y, color, size, fall_x_offset):
        super().__init__(parent)
        self.setText("✦")
        self.setFont(QFont("Arial", size))
        self.setStyleSheet(f"color: {color}; background: transparent;")
        self.setAttribute(Qt.WA_TranslucentBackground)
        self.move(x, y)
        self.show()
        self.anim = QPropertyAnimation(self, b"pos")
        self.anim.setDuration(2500 + random.randint(0, 1500))
        self.anim.setStartValue(QPoint(x, y))
        self.anim.setEndValue(QPoint(x + fall_x_offset, parent.height() + 100))
        self.anim.setEasingCurve(QEasingCurve.OutQuad)
        self.opacity_effect = QGraphicsOpacityEffect(self)
        self.setGraphicsEffect(self.opacity_effect)
        self.fade_anim = QPropertyAnimation(self.opacity_effect, b"opacity")
        self.fade_anim.setDuration(2000)
        self.fade_anim.setStartValue(1.0)
        self.fade_anim.setEndValue(0.0)
        self.anim.start()
        self.fade_anim.start()
        QTimer.singleShot(4000, self.deleteLater)

def play_confetti(parent_widget):
    colors_list = ["#ef4444", "#f97316", "#eab308", "#22c55e", "#3b82f6", "#8b5cf6", "#ec4899"]
    width = parent_widget.width()
    for _ in range(40):
        x = random.randint(0, width)
        y = random.randint(-100, -20)
        color = random.choice(colors_list)
        size = random.randint(18, 32)
        fall_x = random.randint(-120, 120)
        ConfettiParticle(parent_widget, x, y, color, size, fall_x)
```

Gambar 3. 4 The code of Degichi Quiz
Source: Personal do cumentation

init_db Function : Prepares the database and ensures all required fields exist before the app runs.

Key Components:

- sqlite3.connect("quizquest.db") : Connects to the main database file.
- CREATE TABLE IF NOT EXISTS users (...) : Creates the users table containing id, username, password, XP, class, religion, and timestamp.
- ALTER TABLE command : Adds missing columns (grade_class, religion) if they don't exist acts as a "migration".
- INSERT INTO users (...) : Inserts a default admin/testing account.
- conn.commit() : Saves all changes to the database.

This function ensures your quiz/exam app has a complete user table ready to store login accounts and student information, similar to login systems in exam browsers or Quizizz.

AnimatedButton Class : Creates a specially-styled button with hover and click animations.

Key Components:

- setStyleSheet(""" ... """) : Defines custom colors, padding, border radius, and font styling.
- QPushButton:hover : Changes button color when the user hovers.
- QPushButton:pressed : Changes style when clicked.

This class generates modern buttons to improve the UI experience so the app feels more interactive and game-like.

SubjectCard Class : Generates clickable subject cards (Math, Biology, etc.) with themed colors.

Key Components:

- self.subject : Stores the subject name displayed on the card.
- self.theme : Loads color themes from the THEMES dictionary.
- setFixedSize(260, 160) : Sets the card size.
- **Dynamic stylesheet** : Displays subject-specific colors and hover effects.

This class displays each subject as a stylish card that students can click similar to selecting quizzes by subject in Quizizz.

PowerUpButton Class : Provides power-up buttons (clue, 50:50, reveal) used during quizzes.

Key Components:

- self.cost_minutes : Deducts time when a power-up is used.
- **Circular button design** : Created through custom border-radius and colors.
- setToolTip()
  Shows descriptions like "Reduce 10 minutes to get a hint".

This class creates in-quiz helper buttons to mimic Quizizz-style power-ups, adding a game-like strategy element.

ConfettiParticle Class : Creates a floating confetti effect when a student finishes the quiz.

Key Components:

- QGraphicsOpacityEffect : Makes confetti fade out.
- QPropertyAnimation : Controls movement and fade animations.
- setText() : Confetti particles are generated as small colored text labels.

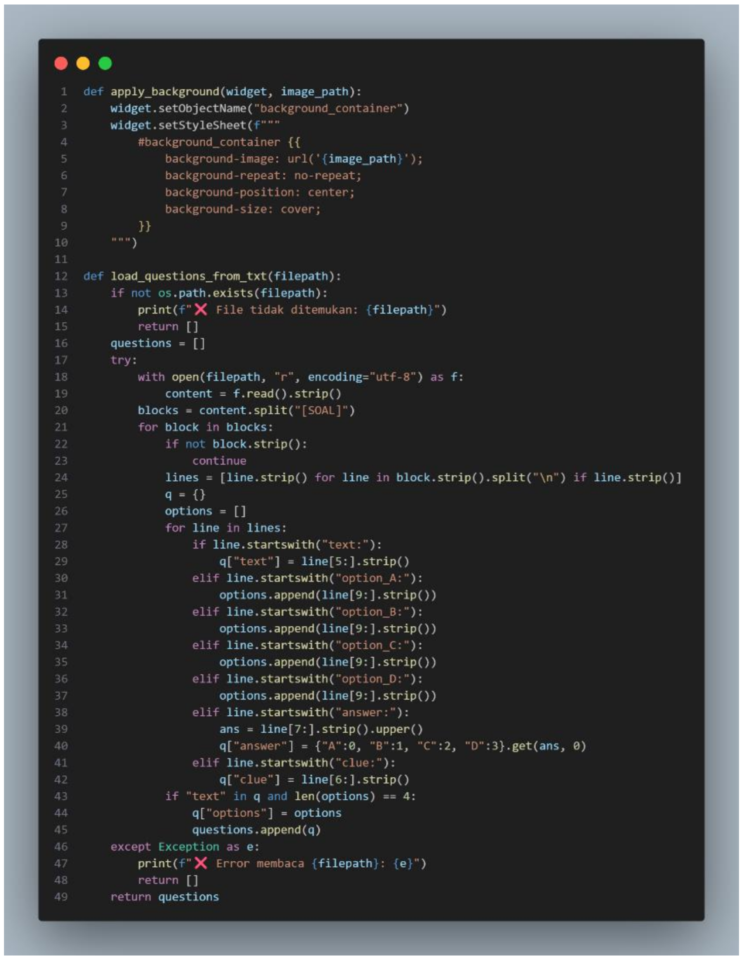
This class animates decorative confetti that falls and fades, used as celebration feedback to motivate students.

play_confetti Function : Spawns multiple ConfettiParticle objects at once for celebration.

Key Components:

- colors_list : Stores many confetti color options.
- **Randomized x-position, size, and fall direction** : Creates natural-looking confetti movement.
- Loop (for _ in range(40)) : Generates around 40 confetti particles.

This function produces the full party effect, dropping many animated confetti pieces when the quiz ends successfully.



```python
def apply_background(widget, image_path):
    widget.setObjectName("background_container")
    widget.setStyleSheet(f"""
        #background_container {{
            background-image: url('{image_path}');
            background-repeat: no-repeat;
            background-position: center;
            background-size: cover;
        }}
    """)

def load_questions_from_txt(filepath):
    if not os.path.exists(filepath):
        print(f"❌ File tidak ditemukan: {filepath}")
        return []
    questions = []
    try:
        with open(filepath, "r", encoding="utf-8") as f:
            content = f.read().strip()
        blocks = content.split("[SOAL]")
        for block in blocks:
            if not block.strip():
                continue
            lines = [line.strip() for line in block.strip().split("\n") if line.strip()]
            q = {}
            options = []
            for line in lines:
                if line.startswith("text:"):
                    q["text"] = line[5:].strip()
                elif line.startswith("option_A:"):
                    options.append(line[9:].strip())
                elif line.startswith("option_B:"):
                    options.append(line[9:].strip())
                elif line.startswith("option_C:"):
                    options.append(line[9:].strip())
                elif line.startswith("option_D:"):
                    options.append(line[9:].strip())
                elif line.startswith("answer:"):
                    ans = line[7:].strip().upper()
                    q["answer"] = {"A":0, "B":1, "C":2, "D":3}.get(ans, 0)
                elif line.startswith("clue:"):
                    q["clue"] = line[6:].strip()
            if "text" in q and len(options) == 4:
                q["options"] = options
                questions.append(q)
    except Exception as e:
        print(f"❌ Error membaca {filepath}: {e}")
        return []
    return questions
```

Gambar 3. 5 The code of Degichi Quiz
Source: Personal documentation

apply_background Function : Ensures each screen (subject page, quiz page, etc.) displays the correct themed background—similar to Quizizz's subject-based visuals.

Key Components

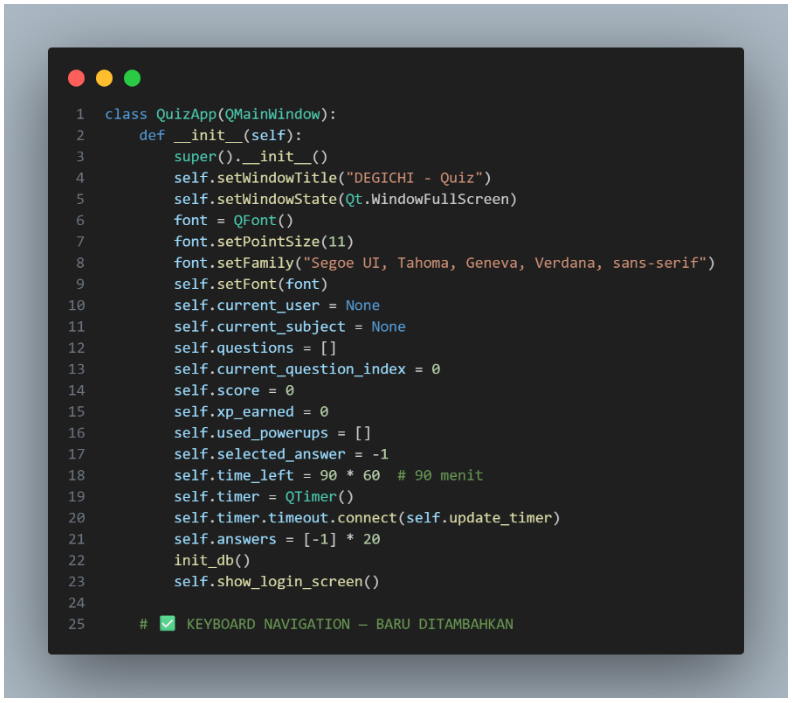
- widget.setObjectName("background_container") : Assigns an object name so the widget can be styled using QSS.
- widget.setStyleSheet(...) : Applies a CSS-style background using the provided image_path.

load_questions_from_txt Function : Converts teacher-made text files into usable quiz questions, supporting the exam system's flexibility (different subjects, classes, and formats). This allows your app to work like Quizizz but still controlled like an exam browser.

Key Components

- os.path.exists(filepath) : Checks if the question file exists.
- open(filepath, "r", encoding="utf-8") : Reads the entire text content from the file.

- content.split("[SOAL]") : Splits questions using a custom block marker.
- for line in block.strip().split("\n") : Processes each line of a question block.
- line.startswith("text:") : Detects question text.
- **option_A / option_B / option_C / option_D** : Extracts answer options.
- line.startswith("answer:") : Retrieves the correct answer letter.
- line.startswith("clue:") ; Loads hint text for the power-up system.
- questions.append(q) : Stores the final parsed question object



```python
class QuizApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("DEGICHI - Quiz")
        self.setWindowState(Qt.WindowFullScreen)
        font = QFont()
        font.setPointSize(11)
        font.setFamily("Segoe UI, Tahoma, Geneva, Verdana, sans-serif")
        self.setFont(font)
        self.current_user = None
        self.current_subject = None
        self.questions = []
        self.current_question_index = 0
        self.score = 0
        self.xp_earned = 0
        self.used_powerups = []
        self.selected_answer = -1
        self.time_left = 90 * 60  # 90 menit
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_timer)
        self.answers = [-1] * 20
        init_db()
        self.show_login_screen()

    # ✅ KEYBOARD NAVIGATION – BARU DITAMBAHKAN
```

Gambar 3. 6 The code of Degichi Quiz
Source: Personal documentation

__init__(self) Function : method prepares everything the quiz system needs before a student can start the exam.

Key Components and Theis Roles in the Degichi Quiz code :

1. **Window Setup**

- super().init()
  Initializes the QMainWindow base class so the app works as a full PyQt window.
- self.setWindowTitle("DEGICHI - Quiz")
  Sets the quiz platform title.
- self.setWindowState(Qt.WindowFullScreen)
  Opens the app in fullscreen to mimic ExamBrowser (prevent tab-switching).

2. **Font Configuration**
   QFont(), setPointSize(11), setFamily(...)
   Applies a clean, readable global font for all text elements during the exam.

3. **User & Subject Tracking**

- self.current_user = None
  Stores the logged-in student's account.
- self.current_subject = None
  Tracks which subject is selected (Math/Bio/etc.), similar to Quizizz topic selection.

4. **Question Management**

- self.questions = []
  Holds all loaded questions from the .txt bank.
- self.current_question_index = 0
  Tracks which question is currently being answered.

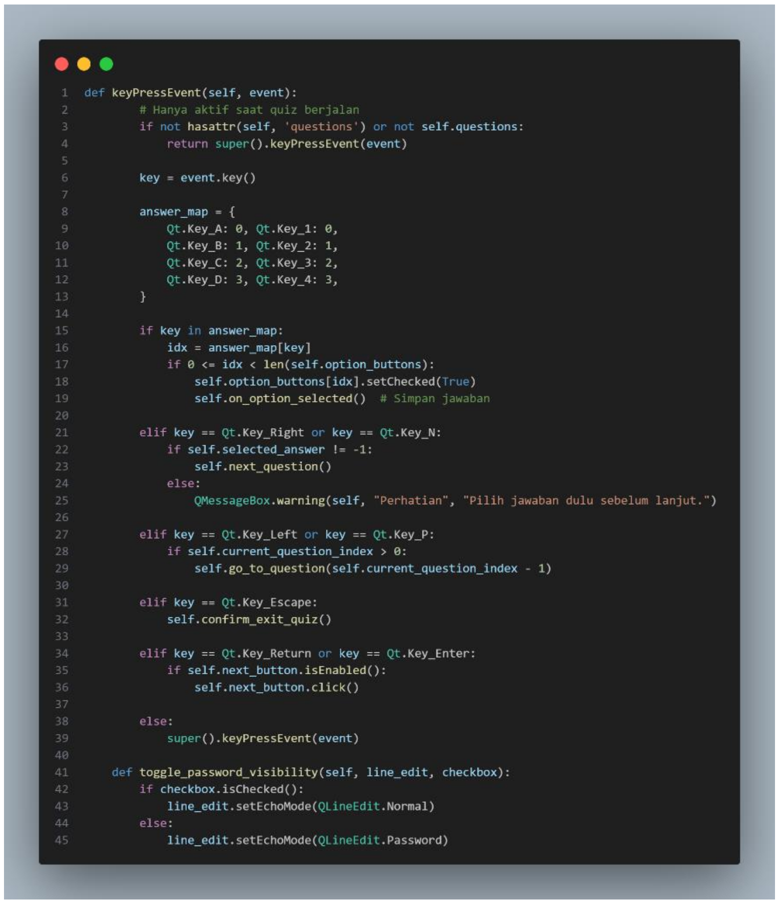
5. **Scoring & Gamification**

- self.score = 0
  Total correct answers.
- self.xp_earned = 0
  XP reward for leveling (Quizizz-style progress).
- self.used_powerups = []
  Logs used power-ups (Clue, 50:50, Reveal).
- self.selected_answer = -1
  Stores the chosen answer for the active question

6. **Timer System**

- self.time_left = 90 * 60
  Sets a 90-minute exam duration.
- self.timer = QTimer()
  Creates the countdown timer.
- self.timer.timeout.connect(self.update_timer)
  Updates the time every second + reacts to time penalties from power-ups.

7. **Answer Storage**
   self.answers = [-1] * 20
   Holds all 20 answers for scoring, review, and generating results.
8. **Database Initialization**
   init_db()
   Prepares the user database (accounts, XP, classes, religion).
   Ensures persistent login like a real learning platform.
9. **Launch Login Screen**
   self.show_login_screen()
   Opens the login interface as the first screen for students.



```python
def keyPressEvent(self, event):
        # Hanya aktif saat quiz berjalan
        if not hasattr(self, 'questions') or not self.questions:
            return super().keyPressEvent(event)

        key = event.key()

        answer_map = {
            Qt.Key_A: 0, Qt.Key_1: 0,
            Qt.Key_B: 1, Qt.Key_2: 1,
            Qt.Key_C: 2, Qt.Key_3: 2,
            Qt.Key_D: 3, Qt.Key_4: 3,
        }

        if key in answer_map:
            idx = answer_map[key]
            if 0 <= idx < len(self.option_buttons):
                self.option_buttons[idx].setChecked(True)
                self.on_option_selected()  # Simpan jawaban

        elif key == Qt.Key_Right or key == Qt.Key_N:
            if self.selected_answer != -1:
                self.next_question()
            else:
                QMessageBox.warning(self, "Perhatian", "Pilih jawaban dulu sebelum lanjut.")

        elif key == Qt.Key_Left or key == Qt.Key_P:
            if self.current_question_index > 0:
                self.go_to_question(self.current_question_index - 1)

        elif key == Qt.Key_Escape:
            self.confirm_exit_quiz()

        elif key == Qt.Key_Return or key == Qt.Key_Enter:
            if self.next_button.isEnabled():
                self.next_button.click()

        else:
            super().keyPressEvent(event)

    def toggle_password_visibility(self, line_edit, checkbox):
        if checkbox.isChecked():
            line_edit.setEchoMode(QLineEdit.Normal)
        else:
            line_edit.setEchoMode(QLineEdit.Password)
```

Gambar 3. 7 The code of Degichi Quiz
Source: Personal documentation

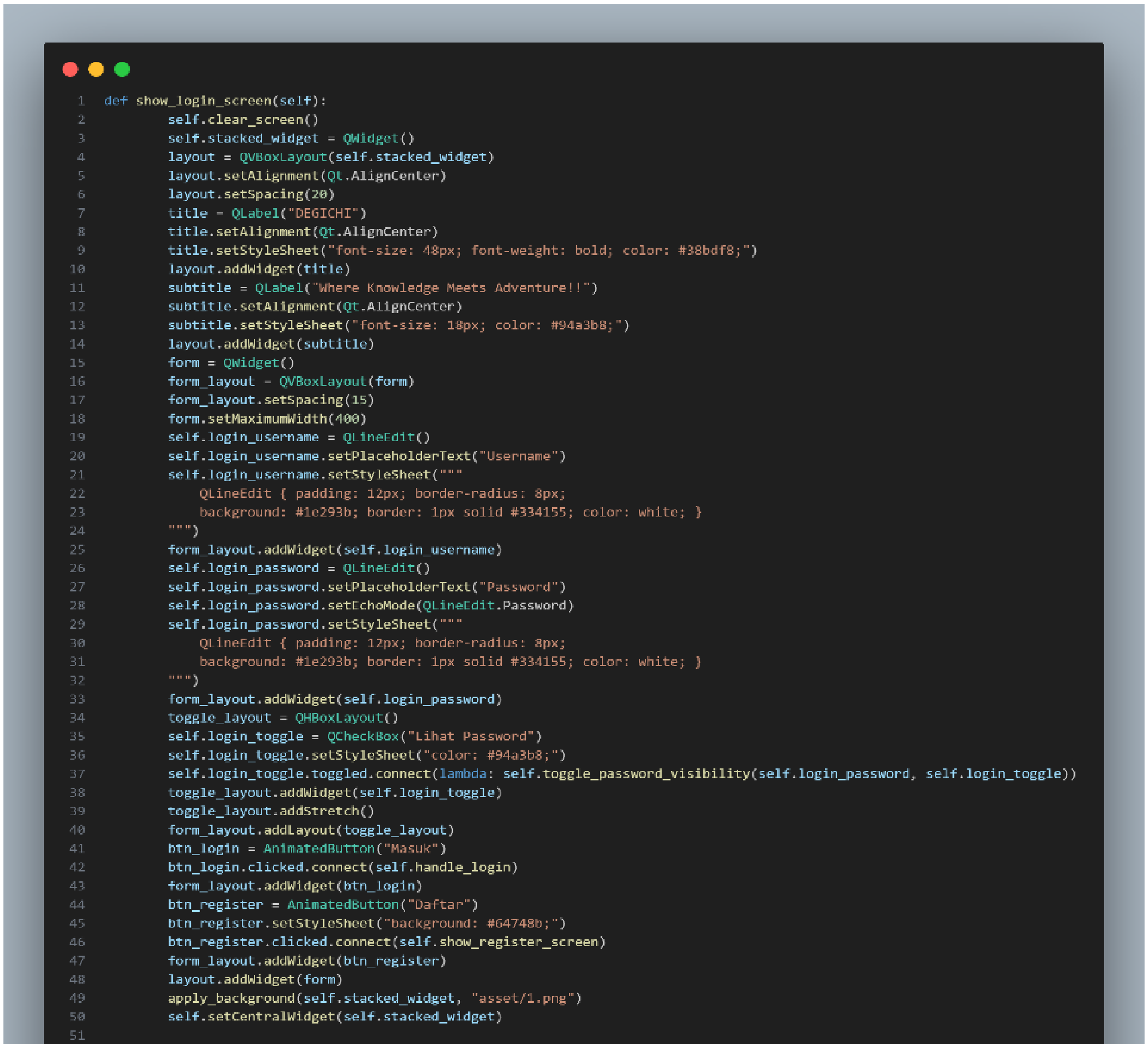
This code is part of the Degichi Quiz application, a learning quiz system that combines features similar to Quizizz (game-based learning) and Exambro (anti-cheating system). The main purpose of this code is to control the quiz using the keyboard, so students can answer quickly and stay focused without relying on the mouse.

keyPressEvent() Function : This function handles what happens when a key is pressed:

- **Press A / B / C / D or 1 / 2 / 3 / 4** → selects an answer instantly
- **Press Right Arrow / N** → moves to the next question (only if an answer is selected)
- **Press Left Arrow / P** → goes back to the previous question
- **Press Enter** → automatically triggers the “Next” button
- **Press Escape** → opens a confirmation dialog before exiting the quiz

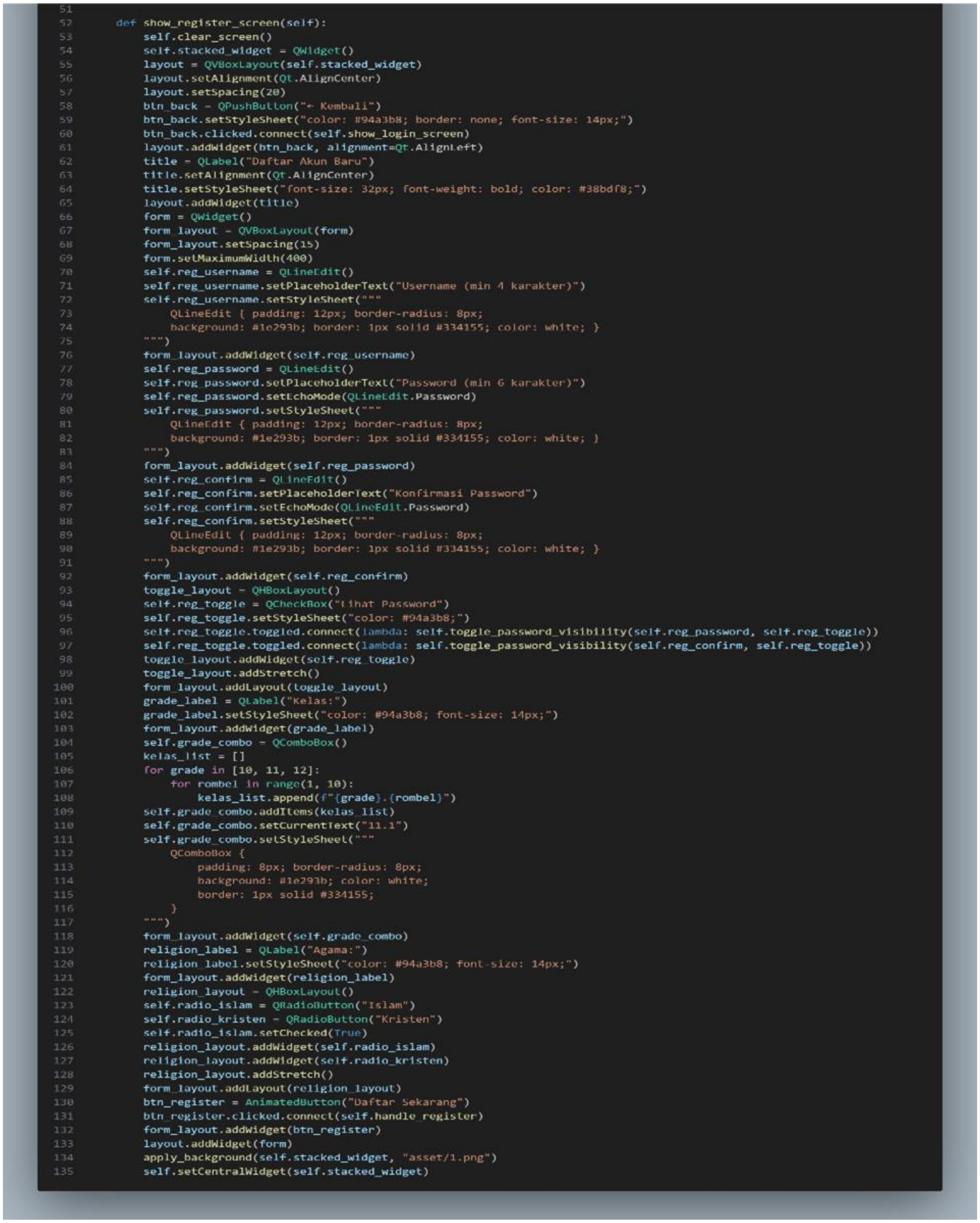
These controls help increase answering speed, create a game-like experience, prevent students from skipping questions, and keep them focused in the quiz environment to reduce cheating behavior.

toggle_password_visibility() Function : This function is used on the login screen, It allows the password to be shown or hidden depending on the checkbox condition. This feature helps students ensure they type the correct password while still maintaining login security.



```python
def show_login_screen(self):
        self.clear_screen()
        self.stacked_widget = QWidget()
        layout = QVBoxLayout(self.stacked_widget)
        layout.setAlignment(Qt.AlignCenter)
        layout.setSpacing(20)
        title = QLabel("DEGICHI")
        title.setAlignment(Qt.AlignCenter)
        title.setStyleSheet("font-size: 48px; font-weight: bold; color: #38bdf8;")
        layout.addWidget(title)
        subtitle = QLabel("Where Knowledge Meets Adventure!!")
        subtitle.setAlignment(Qt.AlignCenter)
        subtitle.setStyleSheet("font-size: 18px; color: #94a3b8;")
        layout.addWidget(subtitle)
        form = QWidget()
        form_layout = QVBoxLayout(form)
        form_layout.setSpacing(15)
        form.setMaximumWidth(400)
        self.login_username = QLineEdit()
        self.login_username.setPlaceholderText("Username")
        self.login_username.setStyleSheet("""
            QLineEdit { padding: 12px; border-radius: 8px;
            background: #1e293b; border: 1px solid #334155; color: white; }
        """)
        form_layout.addWidget(self.login_username)
        self.login_password = QLineEdit()
        self.login_password.setPlaceholderText("Password")
        self.login_password.setEchoMode(QLineEdit.Password)
        self.login_password.setStyleSheet("""
            QLineEdit { padding: 12px; border-radius: 8px;
            background: #1e293b; border: 1px solid #334155; color: white; }
        """)
        form_layout.addWidget(self.login_password)
        toggle_layout = QHBoxLayout()
        self.login_toggle = QCheckBox("Lihat Password")
        self.login_toggle.setStyleSheet("color: #94a3b8;")
        self.login_toggle.toggled.connect(lambda: self.toggle_password_visibility(self.login_password, self.login_toggle))
        toggle_layout.addWidget(self.login_toggle)
        toggle_layout.addStretch()
        form_layout.addLayout(toggle_layout)
        btn_login = AnimatedButton("Masuk")
        btn_login.clicked.connect(self.handle_login)
        form_layout.addWidget(btn_login)
        btn_register = AnimatedButton("Daftar")
        btn_register.setStyleSheet("background: #64748b;")
        btn_register.clicked.connect(self.show_register_screen)
        form_layout.addWidget(btn_register)
        layout.addWidget(form)
        apply_background(self.stacked_widget, "asset/1.png")
        self.setCentralWidget(self.stacked_widget)

```



```python

    def show_register_screen(self):
        self.clear_screen()
        self.stacked_widget = QWidget()
        layout = QVBoxLayout(self.stacked_widget)
        layout.setAlignment(Qt.AlignCenter)
        layout.setSpacing(20)
        btn_back = QPushButton("← Kembali")
        btn_back.setStyleSheet("color: #94a3b8; border: none; font-size: 14px;")
        btn_back.clicked.connect(self.show_login_screen)
        layout.addWidget(btn_back, alignment=Qt.AlignLeft)
        title = QLabel("Daftar Akun Baru")
        title.setAlignment(Qt.AlignCenter)
        title.setStyleSheet("font-size: 32px; font-weight: bold; color: #38bdf8;")
        layout.addWidget(title)
        form = QWidget()
        form_layout = QVBoxLayout(form)
        form_layout.setSpacing(15)
        form.setMaximumWidth(400)
        self.reg_username = QLineEdit()
        self.reg_username.setPlaceholderText("Username (min 4 karakter)")
        self.reg_username.setStyleSheet("""
            QLineEdit { padding: 12px; border-radius: 8px;
            background: #1e293b; border: 1px solid #334155; color: white; }
        """)
        form_layout.addWidget(self.reg_username)
        self.reg_password = QLineEdit()
        self.reg_password.setPlaceholderText("Password (min 6 karakter)")
        self.reg_password.setEchoMode(QLineEdit.Password)
        self.reg_password.setStyleSheet("""
            QLineEdit { padding: 12px; border-radius: 8px;
            background: #1e293b; border: 1px solid #334155; color: white; }
        """)
        form_layout.addWidget(self.reg_password)
        self.reg_confirm = QLineEdit()
        self.reg_confirm.setPlaceholderText("Konfirmasi Password")
        self.reg_confirm.setEchoMode(QLineEdit.Password)
        self.reg_confirm.setStyleSheet("""
            QLineEdit { padding: 12px; border-radius: 8px;
            background: #1e293b; border: 1px solid #334155; color: white; }
        """)
        form_layout.addWidget(self.reg_confirm)
        toggle_layout = QHBoxLayout()
        self.reg_toggle = QCheckBox("Lihat Password")
        self.reg_toggle.setStyleSheet("color: #94a3b8;")
        self.reg_toggle.toggled.connect(lambda: self.toggle_password_visibility(self.reg_password, self.reg_toggle))
        self.reg_toggle.toggled.connect(lambda: self.toggle_password_visibility(self.reg_confirm, self.reg_toggle))
        toggle_layout.addWidget(self.reg_toggle)
        toggle_layout.addStretch()
        form_layout.addLayout(toggle_layout)
        grade_label = QLabel("Kelas:")
        grade_label.setStyleSheet("color: #94a3b8; font-size: 14px;")
        form_layout.addWidget(grade_label)
        self.grade_combo = QComboBox()
        kelas_list = []
        for grade in [10, 11, 12]:
            for rombel in range(1, 10):
                kelas_list.append(f"{grade}.{rombel}")
        self.grade_combo.addItems(kelas_list)
        self.grade_combo.setCurrentText("11.1")
        self.grade_combo.setStyleSheet("""
            QComboBox {
                padding: 8px; border-radius: 8px;
                background: #1e293b; color: white;
                border: 1px solid #334155;
            }
        """)
        form_layout.addWidget(self.grade_combo)
        religion_label = QLabel("Agama:")
        religion_label.setStyleSheet("color: #94a3b8; font-size: 14px;")
        form_layout.addWidget(religion_label)
        religion_layout = QHBoxLayout()
        self.radio_islam = QRadioButton("Islam")
        self.radio_kristen = QRadioButton("Kristen")
        self.radio_islam.setChecked(True)
        religion_layout.addWidget(self.radio_islam)
        religion_layout.addWidget(self.radio_kristen)
        religion_layout.addStretch()
        form_layout.addLayout(religion_layout)
        btn_register = AnimatedButton("Daftar Sekarang")
        btn_register.clicked.connect(self.handle_register)
        form_layout.addWidget(btn_register)
        layout.addWidget(form)
        apply_background(self.stacked_widget, "asset/1.png")
        self.setCentralWidget(self.stacked_widget)
```

Gambar 3. 8 The code of Degichi Quiz
Source : Personal documentation

This part of the Degichi Quiz program is responsible for displaying the Login Screen and Register Screen. Both screens are created using PyQt widgets and layouts to build a clean, modern interface that students can easily interact with.

show_login_screen(self) Function :
This function displays the main login interface:

1. **Main objectives:**

- Allow users (students) to enter their **username** and **password**
- Provide a button for **Login** and a **Register** option to create a new account
- Display a **title and tagline** to make the UI look more like a learning platform

2. **Important UI components:**

- QLabel → displays title and subtitle text
- QLineEdit → input fields for username & password
- AnimatedButton → styled interactive buttons
- QCheckBox → **"Show Password"** feature
- VBoxLayout & HBoxLayout → create a structured and centered layout

The password field uses:

setEchoMode(QLineEdit.Password)→ to hide characters for security.

There's also a connected function:

self.togle_password_visibility(self.login_password, self.login_toggle)→ lets the user toggle between showing and hiding the password.

Buttons behavior:

- **Login button** → calls self.handle_login()
- **Register button** → switches to the register screen using self.show_register_screen()

The background image is applied to make the login page visually appealing.

show_register_screen(self)Function :
This function displays the account creation screen:

1. **Main objectives:**

- Let users register before joining the quiz platform
- Ensure valid and complete data is provided

2. **Fields provided:**

- Username (with minimum length requirement)
- Password (minimum 6 characters)
- Confirm Password (must match)
- Grade selection (dropdown: class 10, 11, 12)
- Religion selection using radio buttons

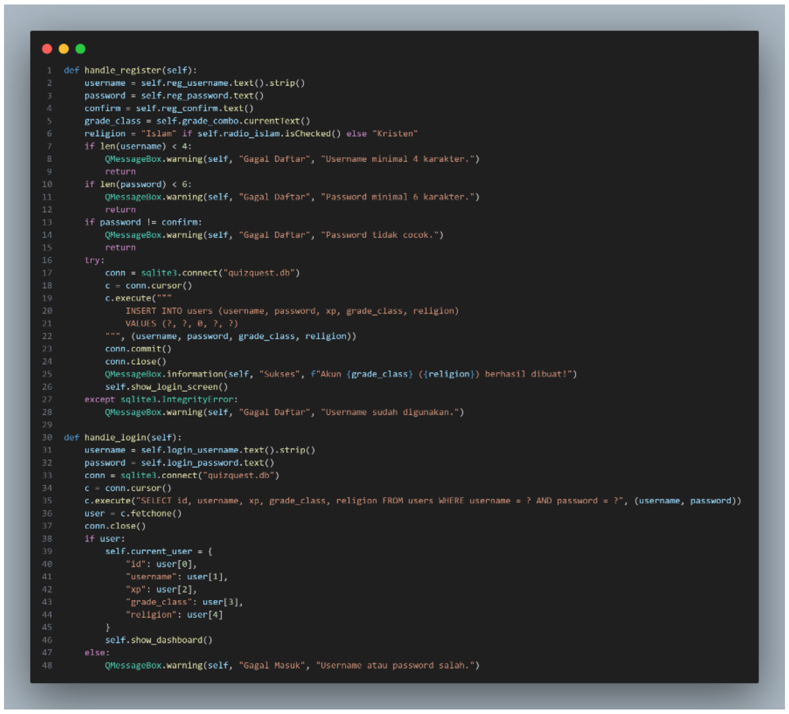
The UI structure is similar to the login screen so the design stays consistent. Each input field has custom styling to match the theme of Degichi Quiz.

3. **Important functional behavior:**

- The toggle checkbox also shows/hides both password fields to prevent typing mistakes
- The register button calls self.handle_register() to process the account data
- A "Back" button allows returning to the login screen easily

This design helps ensure:

- Students enter correct login information
- Smooth navigation between screens
- Clean and accessible UI supporting learning environment



```python
def handle_register(self):
    username = self.reg_username.text().strip()
    password = self.reg_password.text()
    confirm = self.reg_confirm.text()
    grade_class = self.grade_combo.currentText()
    religion = "Islam" if self.radio_islam.isChecked() else "Kristen"
    if len(username) < 4:
        QMessageBox.warning(self, "Gagal Daftar", "Username minimal 4 karakter.")
        return
    if len(password) < 6:
        QMessageBox.warning(self, "Gagal Daftar", "Password minimal 6 karakter.")
        return
    if password != confirm:
        QMessageBox.warning(self, "Gagal Daftar", "Password tidak cocok.")
        return
    try:
        conn = sqlite3.connect("quizquest.db")
        c = conn.cursor()
        c.execute("""
            INSERT INTO users (username, password, xp, grade_class, religion)
            VALUES (?, ?, 0, ?, ?)
        """, (username, password, grade_class, religion))
        conn.commit()
        conn.close()
        QMessageBox.information(self, "Sukses", f"Akun {grade_class} ({religion}) berhasil dibuat!")
        self.show_login_screen()
    except sqlite3.IntegrityError:
        QMessageBox.warning(self, "Gagal Daftar", "Username sudah digunakan.")

def handle_login(self):
    username = self.login_username.text().strip()
    password = self.login_password.text()
    conn = sqlite3.connect("quizquest.db")
    c = conn.cursor()
    c.execute("SELECT id, username, xp, grade_class, religion FROM users WHERE username = ? AND password = ?", (username, password))
    user = c.fetchone()
    conn.close()
    if user:
        self.current_user = {
            "id": user[0],
            "username": user[1],
            "xp": user[2],
            "grade_class": user[3],
            "religion": user[4]
        }
        self.show_dashboard()
    else:
        QMessageBox.warning(self, "Gagal Masuk", "Username atau password salah.")
```

Gambar 3. 9 The code of Degichi Quiz
Source: Personal documentation

Degichi Quiz is a learning game where each student needs a personal profile so their progress, XP, and school data can be tracked. These two functions work as the entry point into the whole system.

handle_register(self) Function : This function is responsible for creating new student accounts. It takes all the information that the student entered in the registration form (username, password, class, religion) and saves it as their identity in the game.

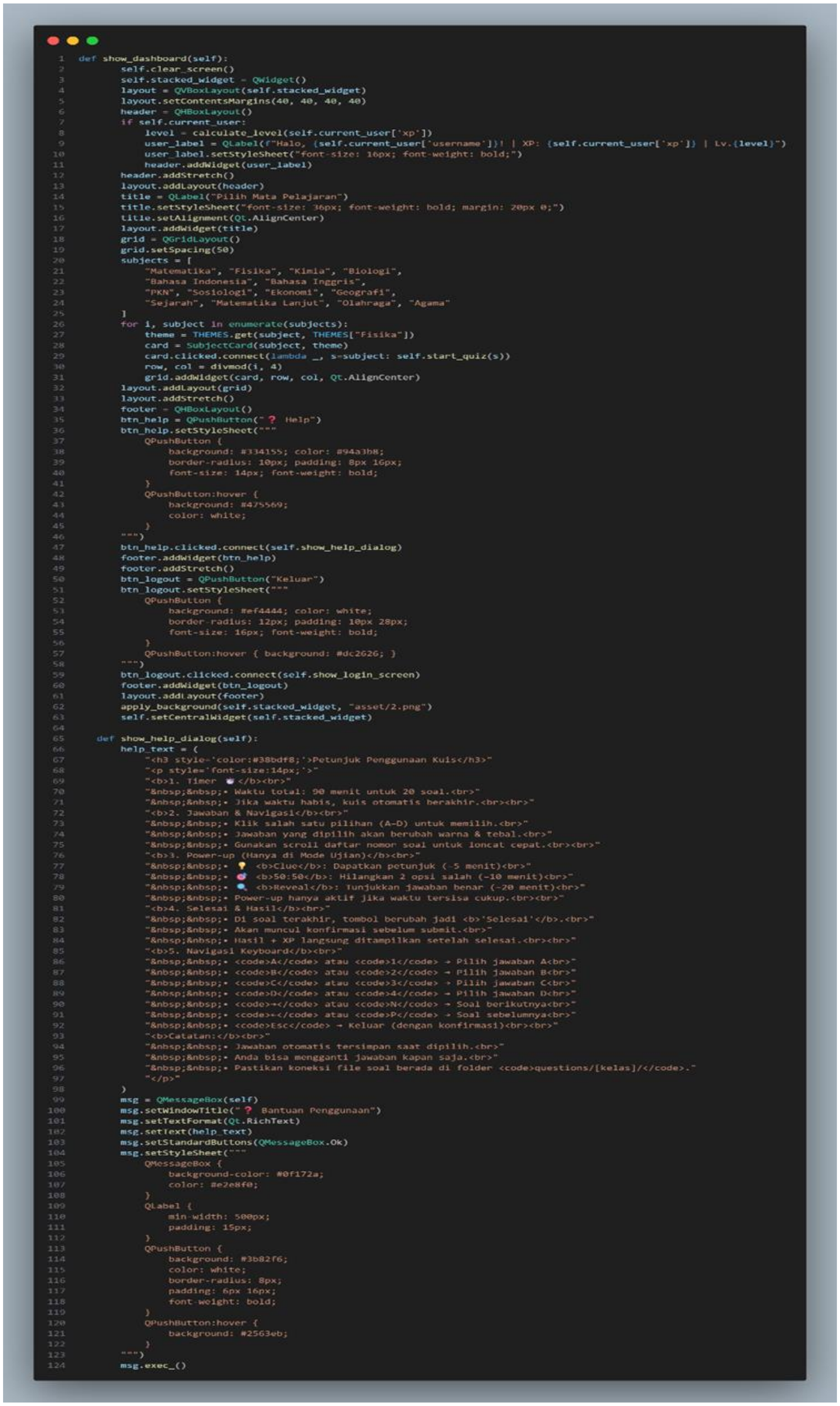
- It ensures every player has a unique profile
- The game can store XP and update progress every time they answer quizzes
- School-required data (grade & religion) are recorded properly
- The student can return later and continue learning from where they left off

If the username is already used, the function stops and asks the student to choose another one.

handle_login(self) Function : This function is used when a student returns and wants to access their existing profile. It checks whether the username and password match a registered student.

- The player's stored data (including XP and class) is loaded
- The game sends them to the dashboard
- They continue their quiz journey seamlessly

If the account does not match, an error message appears and login is denied



```python
def show_dashboard(self):
        self.clear_screen()
        self.stacked_widget = QWidget()
        layout = QVBoxLayout(self.stacked_widget)
        layout.setContentsMargins(40, 40, 40, 40)
        header = QHBoxLayout()
        if self.current_user:
            level = calculate_level(self.current_user['xp'])
            user_label = QLabel(f"Halo, {self.current_user['username']}! | XP: {self.current_user['xp']} | Lv.{level}")
            user_label.setStyleSheet("font-size: 16px; font-weight: bold;")
            header.addWidget(user_label)
        header.addStretch()
        layout.addLayout(header)
        title = QLabel("Pilih Mata Pelajaran")
        title.setStyleSheet("font-size: 36px; font-weight: bold; margin: 20px 0;")
        title.setAlignment(Qt.AlignCenter)
        layout.addWidget(title)
        grid = QGridLayout()
        grid.setSpacing(50)
        subjects = [
            "Matematika", "Fisika", "Kimia", "Biologi",
            "Bahasa Indonesia", "Bahasa Inggris",
            "PKN", "Sosiologi", "Ekonomi", "Geografi",
            "Sejarah", "Matematika Lanjut", "Olahraga", "Agama"
        ]
        for i, subject in enumerate(subjects):
            theme = THEMES.get(subject, THEMES["Fisika"])
            card = SubjectCard(subject, theme)
            card.clicked.connect(lambda _, s=subject: self.start_quiz(s))
            row, col = divmod(i, 4)
            grid.addWidget(card, row, col, Qt.AlignCenter)
        layout.addLayout(grid)
        layout.addStretch()
        footer = QHBoxLayout()
        btn_help = QPushButton(" ❓ Help")
        btn_help.setStyleSheet("""
            QPushButton {
                background: #334155; color: #94a3b8;
                border-radius: 10px; padding: 8px 16px;
                font-size: 14px; font-weight: bold;
            }
            QPushButton:hover {
                background: #475569;
                color: white;
            }
        """)
        btn_help.clicked.connect(self.show_help_dialog)
        footer.addWidget(btn_help)
        footer.addStretch()
        btn_logout = QPushButton("Keluar")
        btn_logout.setStyleSheet("""
            QPushButton {
                background: #ef4444; color: white;
                border-radius: 12px; padding: 10px 28px;
                font-size: 16px; font-weight: bold;
            }
            QPushButton:hover { background: #dc2626; }
        """)
        btn_logout.clicked.connect(self.show_login_screen)
        footer.addWidget(btn_logout)
        layout.addLayout(footer)
        apply_background(self.stacked_widget, "asset/2.png")
        self.setCentralWidget(self.stacked_widget)

    def show_help_dialog(self):
        help_text = (
            "<h3 style='color:#38bdf8;'>Petunjuk Penggunaan Kuis</h3>"
            "<p style='font-size:14px;'>"
            "<b>1. Timer ⏰</b><br>"
            "&nbsp;&nbsp;• Waktu total: 90 menit untuk 20 soal.<br>"
            "&nbsp;&nbsp;• Jika waktu habis, kuis otomatis berakhir.<br><br>"
            "<b>2. Jawaban & Navigasi</b><br>"
            "&nbsp;&nbsp;• Klik salah satu pilihan (A-D) untuk memilih.<br>"
            "&nbsp;&nbsp;• Jawaban yang dipilih akan berubah warna & tebal.<br>"
            "&nbsp;&nbsp;• Gunakan scroll daftar nomor soal untuk loncat cepat.<br><br>"
            "<b>3. Power-up (Hanya di Mode Ujian)</b><br>"
            "&nbsp;&nbsp;• 💡 <b>Clue</b>: Dapatkan petunjuk (-5 menit)<br>"
            "&nbsp;&nbsp;• 🎯 <b>50:50</b>: Hilangkan 2 opsi salah (-10 menit)<br>"
            "&nbsp;&nbsp;• 🔍 <b>Reveal</b>: Tunjukkan jawaban benar (-20 menit)<br>"
            "&nbsp;&nbsp;• Power-up hanya aktif jika waktu tersisa cukup.<br><br>"
            "<b>4. Selesai & Hasil</b><br>"
            "&nbsp;&nbsp;• Di soal terakhir, tombol berubah jadi <b>'Selesai'</b>.<br>"
            "&nbsp;&nbsp;• Akan muncul konfirmasi sebelum submit.<br>"
            "&nbsp;&nbsp;• Hasil + XP langsung ditampilkan setelah selesai.<br><br>"
            "<b>5. Navigasi Keyboard</b><br>"
            "&nbsp;&nbsp;• <code>A</code> atau <code>1</code> → Pilih jawaban A<br>"
            "&nbsp;&nbsp;• <code>B</code> atau <code>2</code> → Pilih jawaban B<br>"
            "&nbsp;&nbsp;• <code>C</code> atau <code>3</code> → Pilih jawaban C<br>"
            "&nbsp;&nbsp;• <code>D</code> atau <code>4</code> → Pilih jawaban D<br>"
            "&nbsp;&nbsp;• <code>→</code> atau <code>N</code> → Soal berikutnya<br>"
            "&nbsp;&nbsp;• <code>←</code> atau <code>P</code> → Soal sebelumnya<br>"
            "&nbsp;&nbsp;• <code>Esc</code> → Keluar (dengan konfirmasi)<br><br>"
            "<b>Catatan:</b><br>"
            "&nbsp;&nbsp;• Jawaban otomatis tersimpan saat dipilih.<br>"
            "&nbsp;&nbsp;• Anda bisa mengganti jawaban kapan saja.<br>"
            "&nbsp;&nbsp;• Pastikan koneksi file soal berada di folder <code>questions/[kelas]/</code>."
            "</p>"
        )
        msg = QMessageBox(self)
        msg.setWindowTitle(" ❓ Bantuan Penggunaan")
        msg.setTextFormat(Qt.RichText)
        msg.setText(help_text)
        msg.setStandardButtons(QMessageBox.Ok)
        msg.setStyleSheet("""
            QMessageBox {
                background-color: #0f172a;
                color: #e2e8f0;
            }
            QLabel {
                min-width: 500px;
                padding: 15px;
            }
            QPushButton {
                background: #3b82f6;
                color: white;
                border-radius: 8px;
                padding: 6px 16px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: #2563eb;
            }
        """)
        msg.exec_()
```

Gambar 3. 10 The code of Degichi Quiz
Source: Personal documentation

These two functions (show_dashboard and show_help_dialog) are the main part of the user interface after login. Their purpose is to guide students into the learning experience and ensure they understand how the quiz system works before entering the exam environment. Degichi Quiz is designed as a modern educational quiz platform that combines:

- Exambro-style discipline (focused exam, anti-cheating behavior)
- Quizizz-style gamification (XP, levels, power-ups, subjects selection)

And these functions are responsible for delivering exactly that.

1. **The Main Home Screen**

show_dashboard(self) Function : This function builds the primary learning interface where students start their quiz journey. It welcomes the student, shows their progress (XP & Level), and presents different subjects like in Quizizz.

- Motivates students by showing XP growth → improves learning engagement
- Allows students to choose subjects independently
- Controls navigation, ensuring they stay inside the learning platform (Exambro-style)
- Prevents distraction — they cannot wander outside during the quiz

Key Components & Roles

- **User Info Display** : Shows username + XP → helps build identity and learning motivation

```
self.username_label = QLabel(f"Welcome, {self.current_user['username']}!")
self.xp_label = QLabel(f"XP: {self.current_user['xp']}")
```

- **Level Calculation** : The more they play → the stronger they become → adds game mechanics

```
level = self.current_user['xp'] // 100
self.level_label = QLabel(f"Level: {level}")
```

- **Subject Grid** : Gives access to different school subjects (Math, Physics, English, etc.)

```
for subject in subjects:
    button = QPushButton(subject)
    button.clicked.connect(lambda checked, s=subject: self.start_quiz(s))
```

- **Click-to-Start Quiz Button** : Every subject becomes an entry point to the quiz engine → fast & direct

```
button.clicked.connect(lambda checked, s=subject: self.start_quiz(s))
```

- **Help Button :** Leads to user guidance → reduces confusion before exam self.help_button.clicked.connect(self.show_help_dialog)

- **Logout Button :** Allowed only outside the quiz → prevents cheating or escape during tests

  self.logout_button.clicked.connect(self.open_logout_popup)

2. **Rule & Guide System**

   show_help_dialog(self) Function : This function informs students about how to interact with the quiz system properly. It prepares them for exam mechanics, preventing errors during answering.

   Key Components & Roles

   - **Timer Explanation** : Time limits enforce discipline like real examinations
   - **Answering Instructions** : Clicking A/B/C/D ensures students make clear selections

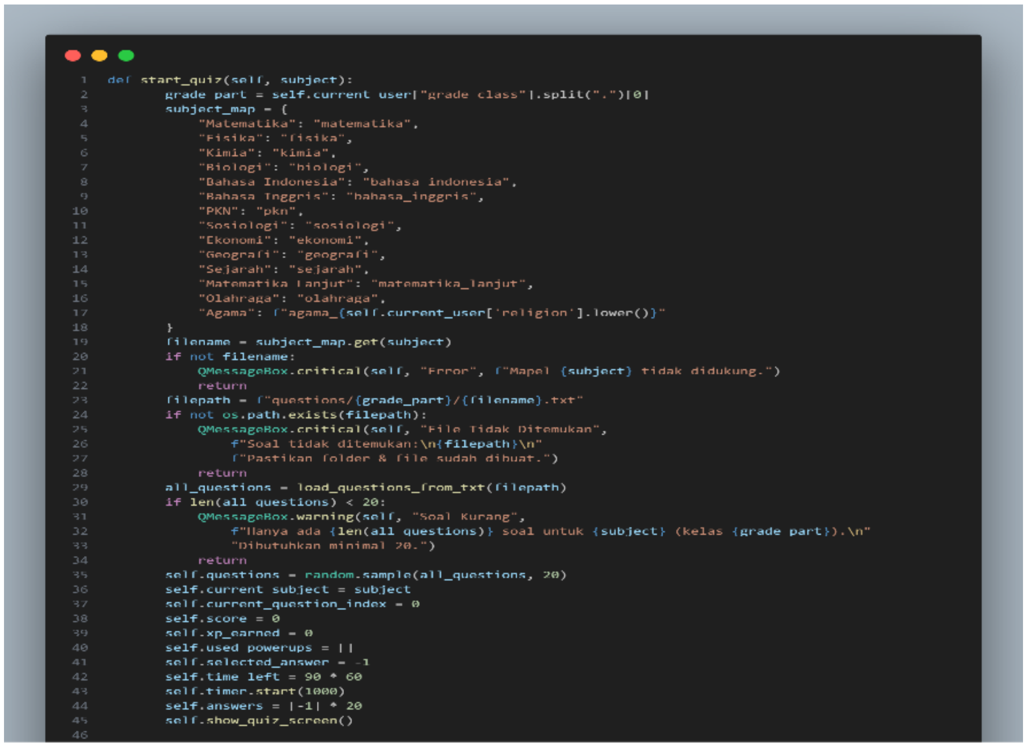
     help_text = ("Select answers by clicking A/B/C/D.\n"

   - **Navigation Rules** : Students can move questions smoothly without losing focus
   - Power-Up Instructions (Quizizz mechanics)
     Example:
     - 50:50 remove two wrong answers
     - Reveal correct answer but reduce remaining time
       → Fun but still learning-focused consequence

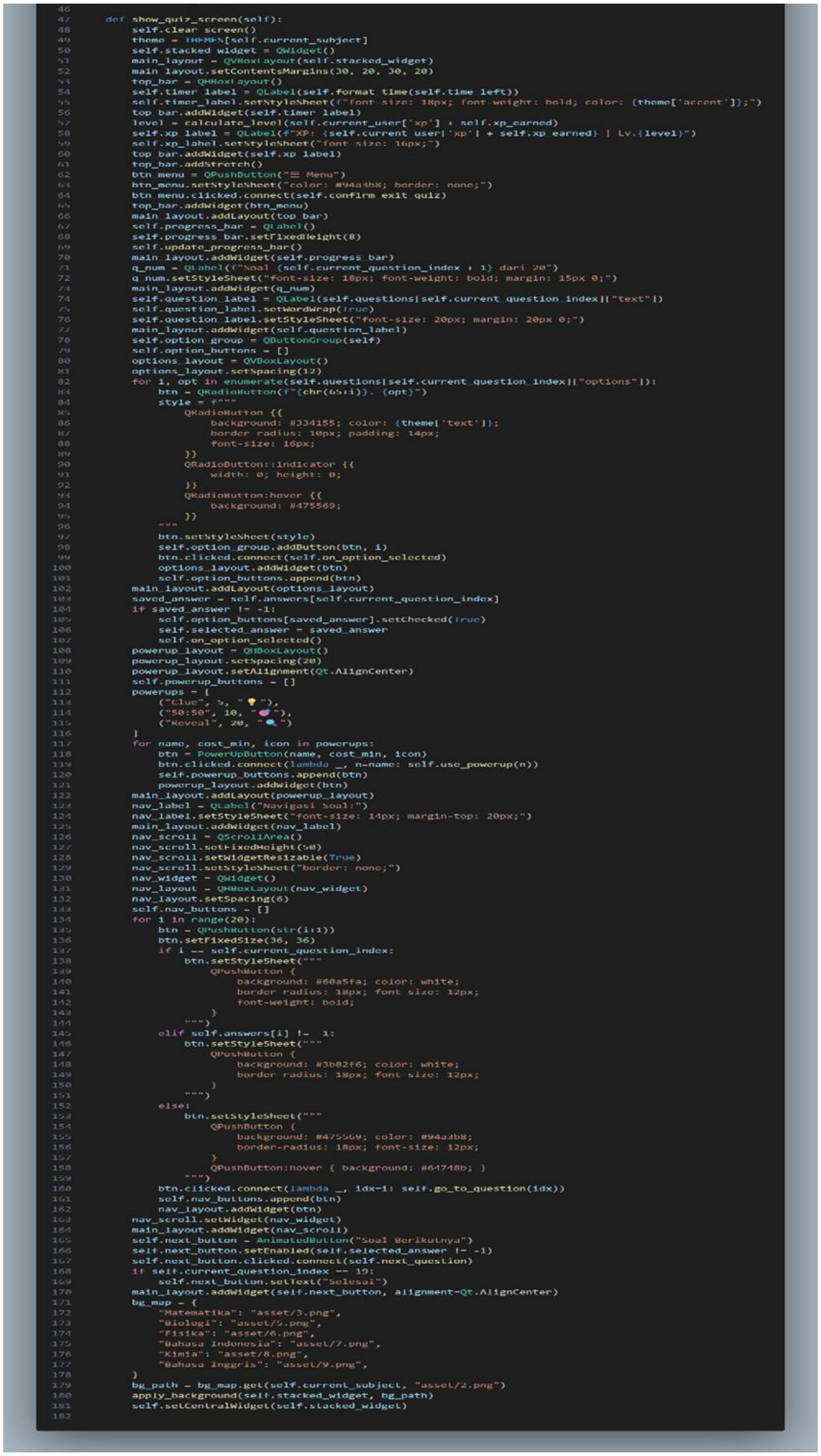
     help_text = ("You may use hints like 50:50 or Reveal Answer (time penalty).\n"

   - **Warning Notes** : Answers saved automatically → prevents cheating like refreshing screen

     help_text = ("Your answers are saved automatically. Do not close the quiz.\n"



```python
def start_quiz(self, subject):
        grade_part = self.current_user["grade_class"].split(".")[0]
        subject_map = {
            "Matematika": "matematika",
            "Fisika": "fisika",
            "Kimia": "kimia",
            "Biologi": "biologi",
            "Bahasa Indonesia": "bahasa_indonesia",
            "Bahasa Inggris": "bahasa_inggris",
            "PKN": "pkn",
            "Sosiologi": "sosiologi",
            "Ekonomi": "ekonomi",
            "Geografi": "geografi",
            "Sejarah": "sejarah",
            "Matematika Lanjut": "matematika_lanjut",
            "Olahraga": "olahraga",
            "Agama": f"agama_{self.current_user['religion'].lower()}"
        }
        filename = subject_map.get(subject)
        if not filename:
            QMessageBox.critical(self, "Error", f"Mapel {subject} tidak didukung.")
            return
        filepath = f"questions/{grade_part}/{filename}.txt"
        if not os.path.exists(filepath):
            QMessageBox.critical(self, "File Tidak Ditemukan",
                f"Soal tidak ditemukan:\n{filepath}\n"
                f"Pastikan folder & file sudah dibuat.")
            return
        all_questions = load_questions_from_txt(filepath)
        if len(all_questions) < 20:
            QMessageBox.warning(self, "Soal Kurang",
                f"Hanya ada {len(all_questions)} soal untuk {subject} (kelas {grade_part}).\n"
                "Dibutuhkan minimal 20.")
            return
        self.questions = random.sample(all_questions, 20)
        self.current_subject = subject
        self.current_question_index = 0
        self.score = 0
        self.xp_earned = 0
        self.used_powerups = []
        self.selected_answer = -1
        self.time_left = 90 * 60
        self.timer.start(1000)
        self.answers = [-1] * 20
        self.show_quiz_screen()
```



```python
    def show_quiz_screen(self):
        self.clear_screen()
        theme = THEMES[self.current_subject]
        self.stacked_widget = QWidget()
        main_layout = QVBoxLayout(self.stacked_widget)
        main_layout.setContentsMargins(30, 20, 30, 20)
        top_bar = QHBoxLayout()
        self.timer_label = QLabel(self.format_time(self.time_left))
        self.timer_label.setStyleSheet(f"font-size: 18px; font-weight: bold; color: {theme['accent']};")
        top_bar.addWidget(self.timer_label)
        level = calculate_level(self.current_user['xp'] + self.xp_earned)
        self.xp_label = QLabel(f"XP: {self.current_user['xp'] + self.xp_earned} | Lv.{level}")
        self.xp_label.setStyleSheet("font-size: 16px;")
        top_bar.addWidget(self.xp_label)
        top_bar.addStretch()
        btn_menu = QPushButton("☰ Menu")
        btn_menu.setStyleSheet("color: #94a3b8; border: none;")
        btn_menu.clicked.connect(self.confirm_exit_quiz)
        top_bar.addWidget(btn_menu)
        main_layout.addLayout(top_bar)
        self.progress_bar = QLabel()
        self.progress_bar.setFixedHeight(8)
        self.update_progress_bar()
        main_layout.addWidget(self.progress_bar)
        q_num = QLabel(f"Soal {self.current_question_index + 1} dari 20")
        q_num.setStyleSheet("font-size: 18px; font-weight: bold; margin: 15px 0;")
        main_layout.addWidget(q_num)
        self.question_label = QLabel(self.questions[self.current_question_index]["text"])
        self.question_label.setWordWrap(True)
        self.question_label.setStyleSheet("font-size: 20px; margin: 20px 0;")
        main_layout.addWidget(self.question_label)
        self.option_group = QButtonGroup(self)
        self.option_buttons = []
        options_layout = QVBoxLayout()
        options_layout.setSpacing(12)
        for i, opt in enumerate(self.questions[self.current_question_index]["options"]):
            btn = QRadioButton(f"{chr(65+i)}. {opt}")
            style = f"""
                QRadioButton {{
                    background: #334155; color: {theme['text']};
                    border-radius: 10px; padding: 14px;
                    font-size: 16px;
                }}
                QRadioButton::indicator {{
                    width: 0; height: 0;
                }}
                QRadioButton:hover {{
                    background: #475569;
                }}
            """
            btn.setStyleSheet(style)
            self.option_group.addButton(btn, i)
            btn.clicked.connect(self.on_option_selected)
            options_layout.addWidget(btn)
            self.option_buttons.append(btn)
        main_layout.addLayout(options_layout)
        saved_answer = self.answers[self.current_question_index]
        if saved_answer != -1:
            self.option_buttons[saved_answer].setChecked(True)
            self.selected_answer = saved_answer
            self.on_option_selected()
        powerup_layout = QHBoxLayout()
        powerup_layout.setSpacing(20)
        powerup_layout.setAlignment(Qt.AlignCenter)
        self.powerup_buttons = []
        powerups = [
            ("Clue", 5, "💡"),
            ("50:50", 10, "🎯"),
            ("Reveal", 20, "🔍")
        ]
        for name, cost_min, icon in powerups:
            btn = PowerUpButton(name, cost_min, icon)
            btn.clicked.connect(lambda _, n=name: self.use_powerup(n))
            self.powerup_buttons.append(btn)
            powerup_layout.addWidget(btn)
        main_layout.addLayout(powerup_layout)
        nav_label = QLabel("Navigasi Soal:")
        nav_label.setStyleSheet("font-size: 14px; margin-top: 20px;")
        main_layout.addWidget(nav_label)
        nav_scroll = QScrollArea()
        nav_scroll.setFixedHeight(50)
        nav_scroll.setWidgetResizable(True)
        nav_scroll.setStyleSheet("border: none;")
        nav_widget = QWidget()
        nav_layout = QHBoxLayout(nav_widget)
        nav_layout.setSpacing(6)
        self.nav_buttons = []
        for i in range(20):
            btn = QPushButton(str(i+1))
            btn.setFixedSize(36, 36)
            if i == self.current_question_index:
                btn.setStyleSheet("""
                    QPushButton {
                        background: #60a5fa; color: white;
                        border-radius: 18px; font-size: 12px;
                        font-weight: bold;
                    }
                """)
            elif self.answers[i] != -1:
                btn.setStyleSheet("""
                    QPushButton {
                        background: #3b82f6; color: white;
                        border-radius: 18px; font-size: 12px;
                    }
                """)
            else:
                btn.setStyleSheet("""
                    QPushButton {
                        background: #475569; color: #94a3b8;
                        border-radius: 18px; font-size: 12px;
                    }
                    QPushButton:hover { background: #64748b; }
                """)
            btn.clicked.connect(lambda _, idx=i: self.go_to_question(idx))
            self.nav_buttons.append(btn)
            nav_layout.addWidget(btn)
        nav_scroll.setWidget(nav_widget)
        main_layout.addWidget(nav_scroll)
        self.next_button = AnimatedButton("Soal Berikutnya")
        self.next_button.setEnabled(self.selected_answer != -1)
        self.next_button.clicked.connect(self.next_question)
        if self.current_question_index == 19:
            self.next_button.setText("Selesai")
        main_layout.addWidget(self.next_button, alignment=Qt.AlignCenter)
        bg_map = {
            "Matematika": "asset/3.png",
            "Biologi": "asset/5.png",
            "Fisika": "asset/6.png",
            "Bahasa Indonesia": "asset/7.png",
            "Kimia": "asset/8.png",
            "Bahasa Inggris": "asset/9.png",
        }
        bg_path = bg_map.get(self.current_subject, "asset/2.png")
        apply_background(self.stacked_widget, bg_path)
        self.setCentralWidget(self.stacked_widget)
```

Gambar 3. 11 The code of Degichi Quiz
Source: Personal documentation

start_quiz(self, subject) Function : Prepare everything before the quiz starts and role in the progam acts as the initializer or setup engine for the quiz system.

- **Map subject → filename**
  filename = subject_map.get(subject) : Matches the selected subject (Math, Biology, etc.) to the correct **question file**, so the quiz loads the proper content.
- **Load questions**
  all_questions = load_questions_from_txt(filepath) : Reads all questions from the text file and prepares them for use in the quiz.
- **Check if enough questions exist**
  if len(all_questions) < 20 : Makes sure the quiz has enough questions. Prevents errors or incomplete quizzes.
- **Select 20 random questions**
  self.questions = random.sample(all_questions, 20) : Randomizes the quiz each time, so students don't get the same question order and reduces cheating.
- **Initialize quiz state variables**
  self.current_question_index = 0
  self.answers = [-1] * 20
  self.xp_earned = 0
  self.score = 0

show_quiz_screen(self) Function : Render the main quiz interface on the screen and role in the program serves as the UI controller and interaction layer.

- **Clear old UI**
  self.clear_screen() : for the role in the program to removes whatever screen was shown before (menu, home, etc.) so the quiz screen can load cleanly without overlapping elements.
- **Create a new container**
  self.stacked_widget = QWidget() : the **main layout container** that holds all quiz elements (timer, question, options). It is the "canvas" for the quiz screen.
- **Display timer**
  self.timer_label = QLabel(self.format_time(self.time_left)) : Shows the countdown for the quiz. This helps control time and creates a challenge (similar to exam or Quizizz timer).
- **Display XP**
  self.xp_label = QLabel(f"XP: {self.current_user['xp']}") : Shows user progress/experience points. This adds a **gamification element** to motivate the student.
- **Render question**
  self.question_label = QLabel(self.questions[self.current_question_index]["question"]) : Displays the current question text. This is the **core content** of the quiz.
- **Render answer options**
  btn = QRadioButton(f"{chr(65+i)}. {opt}") : Creates each answer button (A, B, C, D). This lets the student choose an answer in a clear, standard quiz format.
- **Restore saved answer**
  saved = self.answers[self.current_question_index] : If the user goes back to a previous question, their old answer is shown again. Prevents losing work and makes navigation smooth.

- **Handle option selection**
  btn.clicked.connect(self.on_option_selected) : Link the button to the function that record the answer. This is how the progam **knows** which option the student picked,

Power-ups Section – To Add optional in-game tools like clue, 50:50, reveal and the rule of this program acts as the game mechanic / gamification engine that makes the quiz feel like Quizizz.

```
powerups = {
    'Clue': ('🔍', amount), → give hints
    '50:50': ('🎲', amount), → reduce choices
    'Reveal': ('💡', amount), → show correct answer
}
```

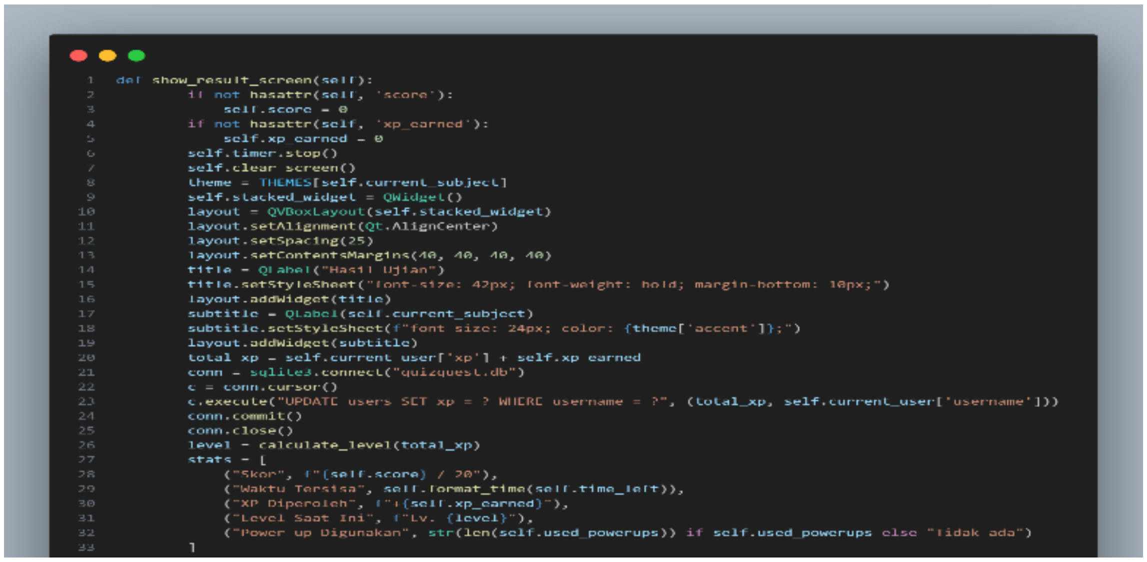
Navigation Controls – In the program for move to next question and acts as the flow controller for quiz progression. Keeps the quiz progressing smoothly and decides when the quiz ends.

```
self.next_button.clicked.connect(self.next_question)
```

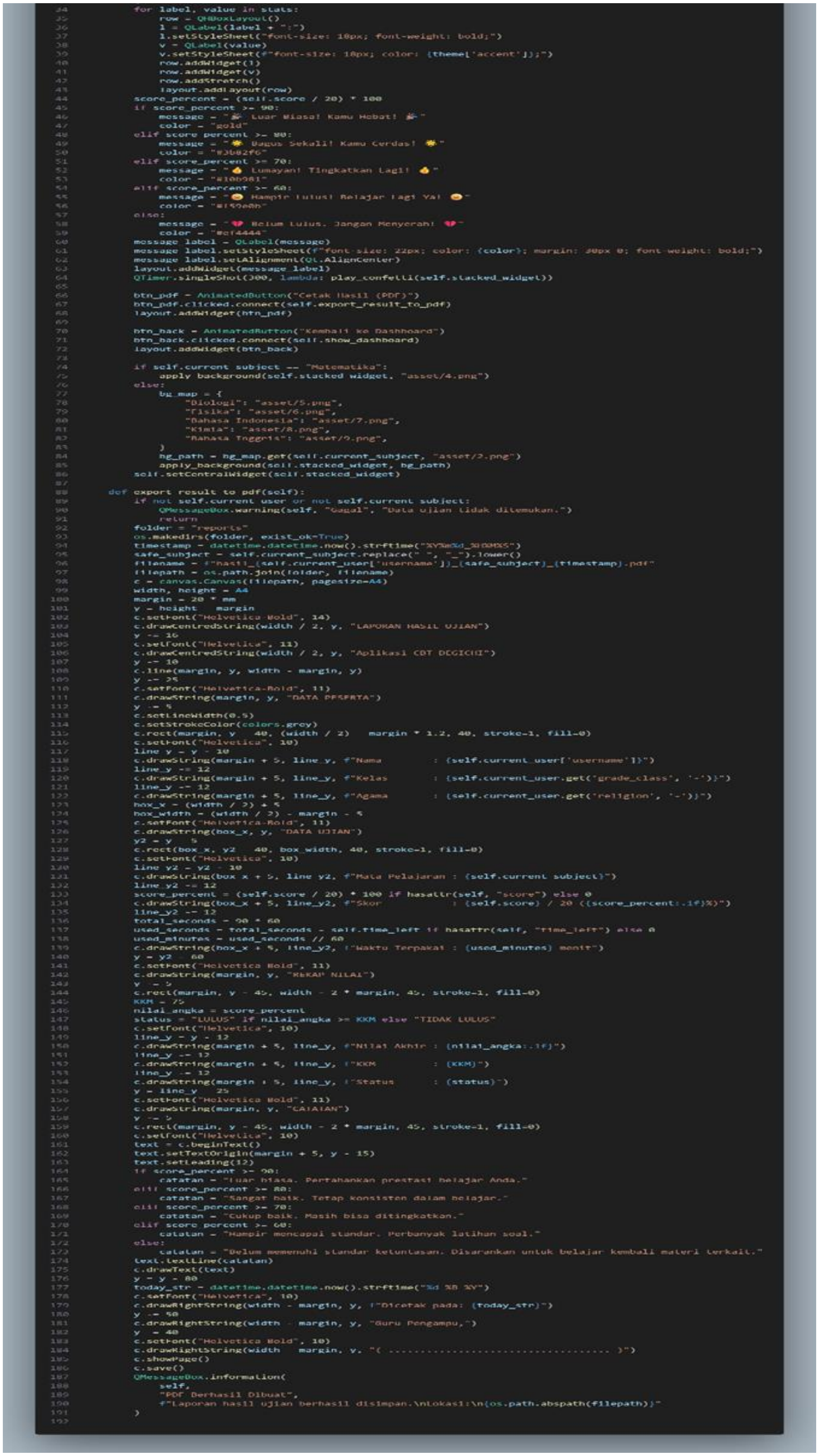
- Saves current answer
- Moves to next question
- Detects if last question
- Changes button text to “Finish”

Background Mapping - Show a background image based on subject and Improves the visual experience and links each subject to a theme.

```
bg_map = {
    "Matematika": "asset/3.png",
    "Biologi": "asset/5.png",
}
```



```
def show_result_screen(self):
        if not hasattr(self, 'score'):
            self.score = 0
        if not hasattr(self, 'xp_earned'):
            self.xp_earned = 0
        self.timer.stop()
        self.clear_screen()
        theme = THEMES[self.current_subject]
        self.stacked_widget = QWidget()
        layout = QVBoxLayout(self.stacked_widget)
        layout.setAlignment(Qt.AlignCenter)
        layout.setSpacing(25)
        layout.setContentsMargins(40, 40, 40, 40)
        title = QLabel("Hasil Ujian")
        title.setStyleSheet("font-size: 42px; font-weight: bold; margin-bottom: 10px;")
        layout.addWidget(title)
        subtitle = QLabel(self.current_subject)
        subtitle.setStyleSheet(f"font size: 24px; color: {theme['accent']};")
        layout.addWidget(subtitle)
        total xp = self.current user['xp'] + self.xp earned
        conn = sqlite3.connect("quizquest.db")
        c = conn.cursor()
        c.execute("UPDATE users SET xp = ? WHERE username = ?", (total_xp, self.current_user['username']))
        conn.commit()
        conn.close()
        level = calculate_level(total_xp)
        stats = [
            ("Skor", f"{self.score} / 20"),
            ("Waktu Tersisa", self.format_time(self.time_left)),
            ("XP Diperoleh", f"+{self.xp_earned}"),
            ("Level Saat Ini", f"Lv. {level}"),
            ("Power up Digunakan", str(len(self.used_powerups)) if self.used_powerups else "Tidak ada")
        ]
```



```python
        for label, value in stats:
            row = QHBoxLayout()
            l = QLabel(label + ":")
            l.setStyleSheet("font-size: 18px; font-weight: bold;")
            v = QLabel(value)
            v.setStyleSheet(f"font-size: 18px; color: {theme['accent']};")
            row.addWidget(l)
            row.addWidget(v)
            row.addStretch()
            layout.addLayout(row)
        score_percent = (self.score / 20) * 100
        if score_percent >= 90:
            message = "🎉 Luar Biasa! Kamu Hebat! 🎉"
            color = "gold"
        elif score_percent >= 80:
            message = "🌟 Bagus Sekali! Kamu Cerdas! 🌟"
            color = "#3b82f6"
        elif score_percent >= 70:
            message = "👍 Lumayan! Tingkatkan Lagi! 👍"
            color = "#10b981"
        elif score_percent >= 60:
            message = "😅 Hampir Lulus! Belajar Lagi Ya! 😅"
            color = "#f59e0b"
        else:
            message = "💔 Belum Lulus. Jangan Menyerah! 💔"
            color = "#ef4444"
        message_label = QLabel(message)
        message_label.setStyleSheet(f"font-size: 22px; color: {color}; margin: 30px 0; font-weight: bold;")
        message_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(message_label)
        QTimer.singleShot(300, lambda: play_confetti(self.stacked_widget))

        btn_pdf = AnimatedButton("Cetak Hasil (PDF)")
        btn_pdf.clicked.connect(self.export_result_to_pdf)
        layout.addWidget(btn_pdf)

        btn_back = AnimatedButton("Kembali ke Dashboard")
        btn_back.clicked.connect(self.show_dashboard)
        layout.addWidget(btn_back)

        if self.current_subject == "Matematika":
            apply_background(self.stacked_widget, "asset/4.png")
        else:
            bg_map = {
                "Biologi": "asset/5.png",
                "Fisika": "asset/6.png",
                "Bahasa Indonesia": "asset/7.png",
                "Kimia": "asset/8.png",
                "Bahasa Inggris": "asset/9.png",
            }
            bg_path = bg_map.get(self.current_subject, "asset/2.png")
            apply_background(self.stacked_widget, bg_path)
        self.setCentralWidget(self.stacked_widget)

    def export_result_to_pdf(self):
        if not self.current_user or not self.current_subject:
            QMessageBox.warning(self, "Gagal", "Data ujian tidak ditemukan.")
            return
        folder = "reports"
        os.makedirs(folder, exist_ok=True)
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        safe_subject = self.current_subject.replace(" ", "_").lower()
        filename = f"hasil_{self.current_user['username']}_{safe_subject}_{timestamp}.pdf"
        filepath = os.path.join(folder, filename)
        c = canvas.Canvas(filepath, pagesize=A4)
        width, height = A4
        margin = 20 * mm
        y = height - margin
        c.setFont("Helvetica-Bold", 14)
        c.drawCentredString(width / 2, y, "LAPORAN HASIL UJIAN")
        y -= 16
        c.setFont("Helvetica", 11)
        c.drawCentredString(width / 2, y, "Aplikasi CBT DEGICHI")
        y -= 10
        c.line(margin, y, width - margin, y)
        y -= 25
        c.setFont("Helvetica-Bold", 11)
        c.drawString(margin, y, "DATA PESERTA")
        y -= 5
        c.setLineWidth(0.5)
        c.setStrokeColor(colors.grey)
        c.rect(margin, y - 40, (width / 2) - margin * 1.2, 40, stroke=1, fill=0)
        c.setFont("Helvetica", 10)
        line_y = y - 10
        c.drawString(margin + 5, line_y, f"Nama          : {self.current_user['username']}")
        line_y -= 12
        c.drawString(margin + 5, line_y, f"Kelas         : {self.current_user.get('grade_class', '-')}")
        line_y -= 12
        c.drawString(margin + 5, line_y, f"Agama         : {self.current_user.get('religion', '-')}")
        box_x = (width / 2) + 5
        box_width = (width / 2) - margin - 5
        c.setFont("Helvetica-Bold", 11)
        c.drawString(box_x, y, "DATA UJIAN")
        y2 = y - 5
        c.rect(box_x, y2 - 40, box_width, 40, stroke=1, fill=0)
        c.setFont("Helvetica", 10)
        line_y2 = y2 - 10
        c.drawString(box_x + 5, line_y2, f"Mata Pelajaran : {self.current_subject}")
        line_y2 -= 12
        score_percent = (self.score / 20) * 100 if hasattr(self, "score") else 0
        c.drawString(box_x + 5, line_y2, f"Skor           : {self.score} / 20 ({score_percent:.1f}%)")
        line_y2 -= 12
        total_seconds = 90 * 60
        used_seconds = total_seconds - self.time_left if hasattr(self, "time_left") else 0
        used_minutes = used_seconds // 60
        c.drawString(box_x + 5, line_y2, f"Waktu Terpakai : {used_minutes} menit")
        y = y2 - 60
        c.setFont("Helvetica-Bold", 11)
        c.drawString(margin, y, "REKAP NILAI")
        y -= 5
        c.rect(margin, y - 45, width - 2 * margin, 45, stroke=1, fill=0)
        KKM = 75
        nilai_angka = score_percent
        status = "LULUS" if nilai_angka >= KKM else "TIDAK LULUS"
        c.setFont("Helvetica", 10)
        line_y = y - 12
        c.drawString(margin + 5, line_y, f"Nilai Akhir : {nilai_angka:.1f}")
        line_y -= 12
        c.drawString(margin + 5, line_y, f"KKM         : {KKM}")
        line_y -= 12
        c.drawString(margin + 5, line_y, f"Status      : {status}")
        y = line_y - 25
        c.setFont("Helvetica-Bold", 11)
        c.drawString(margin, y, "CATATAN")
        y -= 5
        c.rect(margin, y - 45, width - 2 * margin, 45, stroke=1, fill=0)
        c.setFont("Helvetica", 10)
        text = c.beginText()
        text.setTextOrigin(margin + 5, y - 15)
        text.setLeading(12)
        if score_percent >= 90:
            catatan = "Luar biasa. Pertahankan prestasi belajar Anda."
        elif score_percent >= 80:
            catatan = "Sangat baik. Tetap konsisten dalam belajar."
        elif score_percent >= 70:
            catatan = "Cukup baik. Masih bisa ditingkatkan."
        elif score_percent >= 60:
            catatan = "Hampir mencapai standar. Perbanyak latihan soal."
        else:
            catatan = "Belum memenuhi standar ketuntasan. Disarankan untuk belajar kembali materi terkait."
        text.textLine(catatan)
        c.drawText(text)
        y = y - 80
        today_str = datetime.datetime.now().strftime("%d %B %Y")
        c.setFont("Helvetica", 10)
        c.drawRightString(width - margin, y, f"Dicetak pada: {today_str}")
        y -= 50
        c.drawRightString(width - margin, y, "Guru Pengampu,")
        y -= 40
        c.setFont("Helvetica-Bold", 10)
        c.drawRightString(width - margin, y, "( ..................................... )")
        c.showPage()
        c.save()
        QMessageBox.information(
            self,
            "PDF Berhasil Dibuat",
            f"Laporan hasil ujian berhasil disimpan.\nLokasi:\n{os.path.abspath(filepath)}"
        )
```

Gambar 3. 12 The code of Degichi Quiz
Source: Personal documentation

This code acts as the "**Result & Reporting System**" of the DEGICHI Quiz application. It determines the student's final score, shows pass/fail status, and generates an official report similar to a mix of Exam Browser (secure environment) and Quizizz (animations, feedback, power-ups, and XP).

show_result_screen(self) Function : To display the results page after the student finishes the quiz.

Key Components & Their Roles

- self.score and self.xp_earned : Calculate the final score and XP earned.
- self.timer.stop() : Stops the exam timer once the quiz is finished.
- UI Builder (QWidget, QVBoxLayout, QLabel) : Constructs the visual layout: title, score, subtitle, feedback messages.
- **Database update** : Uses SQL to update the student's total XP.
- **Status feedback** : Shows motivational messages such as "Great job! 😄🔥" or "Keep trying!"
- **Confetti Animation** : Triggers play_confetti() if the student achieves a high score.
- **Navigation Buttons** : PDF export button + back-to-dashboard button.
- **Background Theme** : Changes background based on subject—similar to Quizizz styling.

Role in the System:

- the final score
- pass/fail status
- earned XP
- motivational feedback
- a button to export the PDF report

export_result_to_pdf(self) Function : To generate an official PDF exam report (clean and formatted like a digital report card).

Key Components & Their Roles

- **Folder and filename builder**
  Creates the "report/" folder and generates filenames automatically:
  username_subject_date.pdf
- **ReportLab Canvas**
  Writes all exam data into a PDF.
- **Data Rendering** :
  - Student name
  - Class

    - Religion
    - Subject
    - Score
    - Pass/Fail status
    - Teacher notes
- **KKM checking**
  Determines “PASS” or “FAIL” based on final score.
- **Text formatting**
  Controls margins, fonts, and layout.
- **Final save**
  Saves the file and shows a message: *“PDF Successfully Created.”*

Role in the System:

Creates a formal proof that the student has completed the quiz useful for teachers and score documentation.

Interaction with the DEGICHI Quiz System : This section supports key features of the DEGICHI Quiz ecosystem:

1. **ExamBrowser-like Behavior**

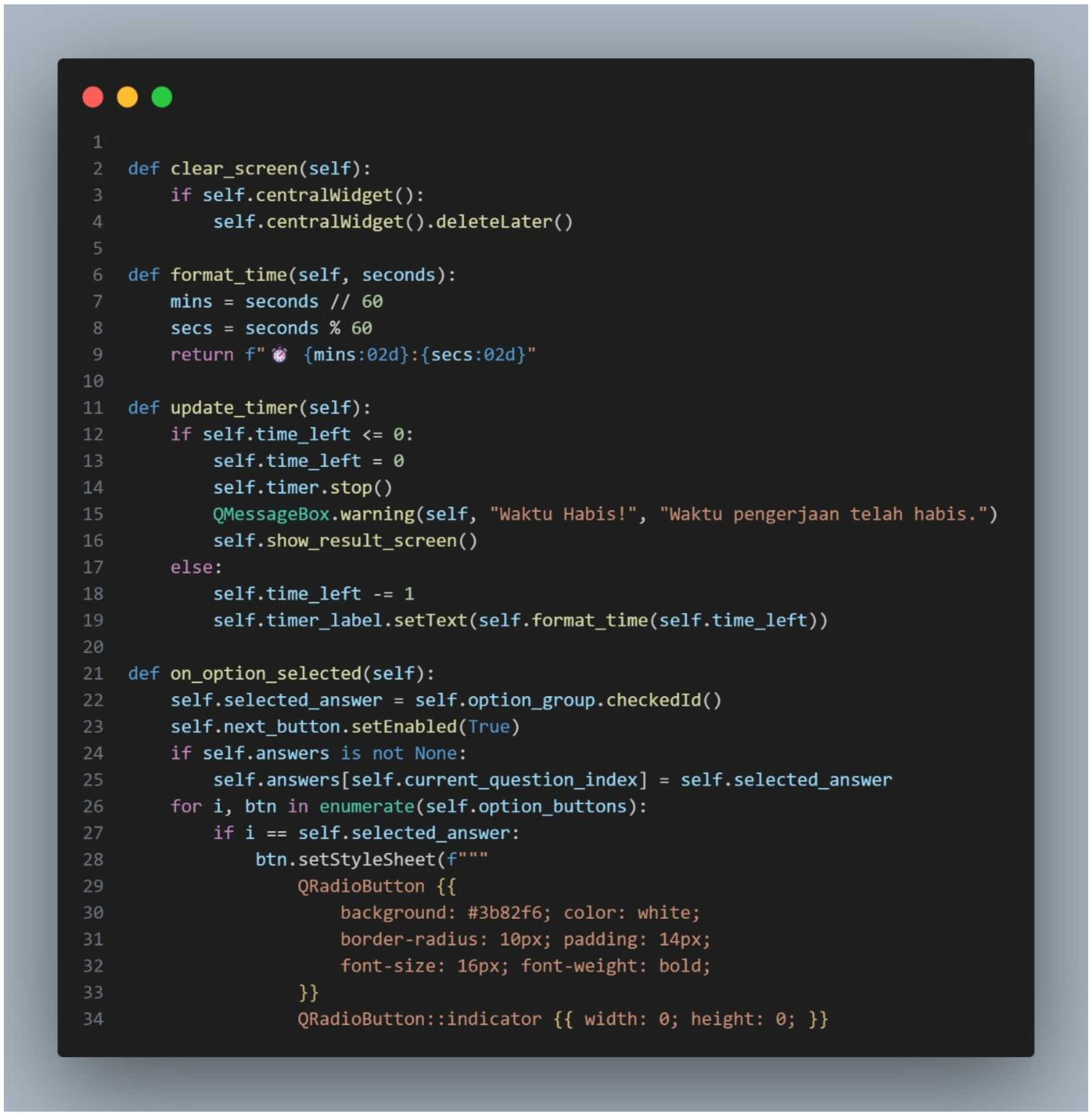
- Timer is strictly controlled
- UI switches to result mode with no escape
- Secure end-of-test handling

2. **Quizizz-like Behavior**

- XP rewards
- Motivational messages
- Confetti animations
- Subject-based themes
- Power-ups that consume time

3. **CBT / Report System Features**

- Digital PDF report
- Pass/fail evaluation
- KKM comparison



```python

def clear_screen(self):
    if self.centralWidget():
        self.centralWidget().deleteLater()

def format_time(self, seconds):
    mins = seconds // 60
    secs = seconds % 60
    return f"⏰ {mins:02d}:{secs:02d}"

def update_timer(self):
    if self.time_left <= 0:
        self.time_left = 0
        self.timer.stop()
        QMessageBox.warning(self, "Waktu Habis!", "Waktu pengerjaan telah habis.")
        self.show_result_screen()
    else:
        self.time_left -= 1
        self.timer_label.setText(self.format_time(self.time_left))

def on_option_selected(self):
    self.selected_answer = self.option_group.checkedId()
    self.next_button.setEnabled(True)
    if self.answers is not None:
        self.answers[self.current_question_index] = self.selected_answer
    for i, btn in enumerate(self.option_buttons):
        if i == self.selected_answer:
            btn.setStyleSheet(f"""
                QRadioButton {{
                    background: #3b82f6; color: white;
                    border-radius: 10px; padding: 14px;
                    font-size: 16px; font-weight: bold;
                }}
                QRadioButton::indicator {{ width: 0; height: 0; }}
```

Gambar 3. 13 The code of Degichi Quiz
Source: Personal documentation

This part of the DEGICHI Quiz program handles the screen display, countdown timer, progress bar, and answer selection. These features make the quiz run smoothly, help students track their progress, and keep the exam system secure, just like a mix of ExamBrowser and Quizizz.

clear_screen(self) Function : Removes the current page/screen before loading a new one. Ensures each screen (login, dashboard, quiz, result) loads cleanly without UI elements overlapping.
This behaves like an ExamBrowser page transition always clean and isolated.

Key Components

- self.centralWidget(): the current visible page

- deleteLater(): removes the widget from memory

format_time(self, seconds) Function : Converts total seconds into MM:SS format and can be makes the countdown timer readable for students during the exam.

Key Components

- mins = seconds // 60
- secs = seconds % 60
- Returns "{mins:02d}:{secs:02d}"

update_timer(self) Function : Updates the exam timer every second.

Key Components

- self.time_left: remaining time
- self.timer.stop(): stops timer when time is up
- QMessageBox.warning(...): shows "Time's Up!"
- self.show_result_screen(): goes to results automatically

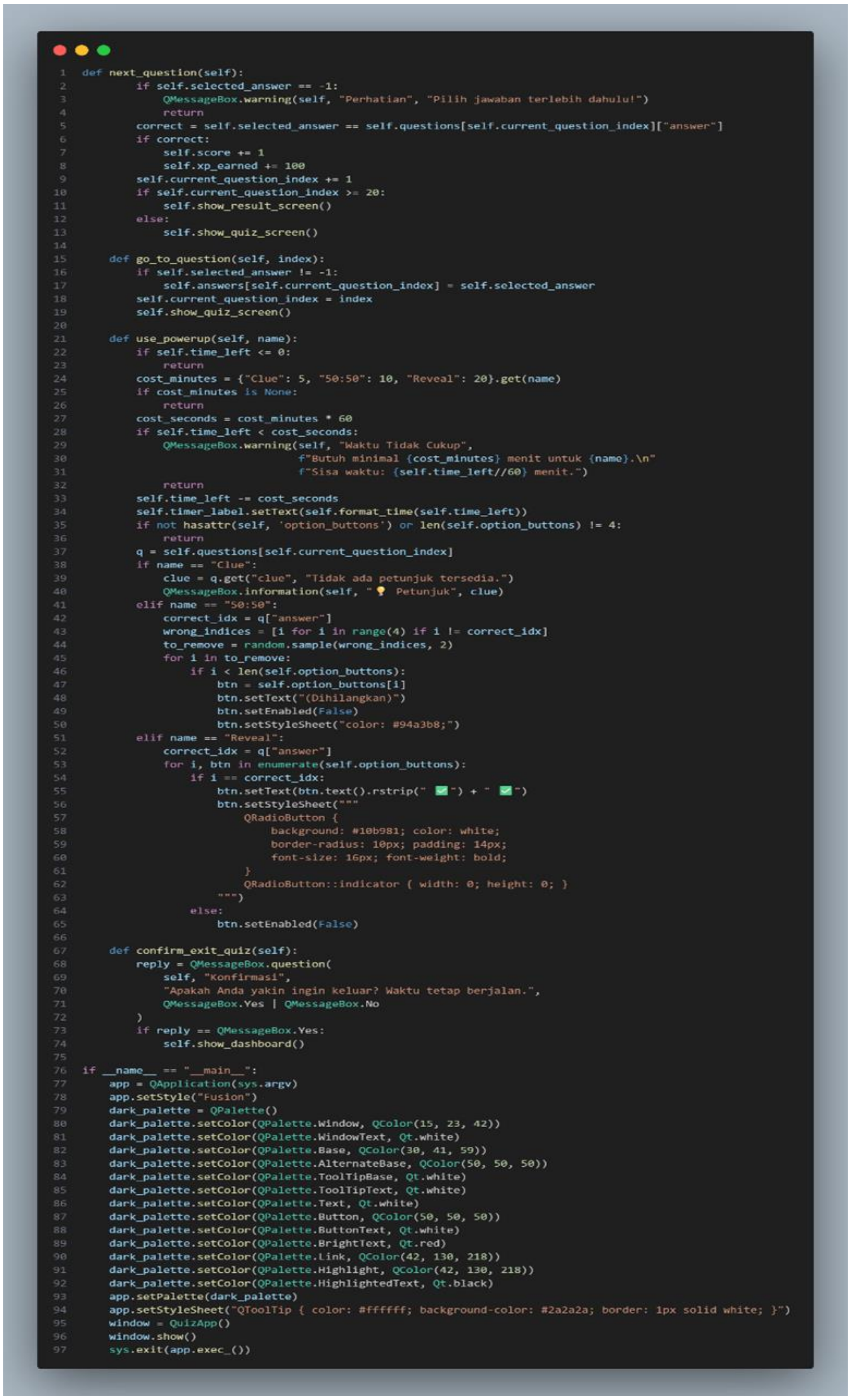
Role in the System

- automatically ends the exam
- prevents extra time
- forces result submission

Power-ups that reduce time also affect self.time_left, so this connects Quizizz-style gameplay with exam rules.

Key Components

- self.selected_answer : stores which option is clicked
- self.option_group.checkedId() : detects the selected radio button
- **Updates visual style:**
  - selected = bold white button
  - unselected = theme-colored button
- self.answers[self.current_question_index] : saves answer permanently
- self.next_button.setEnabled(True) : unlocks "Next Question"



```python
def next_question(self):
        if self.selected_answer == -1:
            QMessageBox.warning(self, "Perhatian", "Pilih jawaban terlebih dahulu!")
            return
        correct = self.selected_answer == self.questions[self.current_question_index]["answer"]
        if correct:
            self.score += 1
            self.xp_earned += 100
        self.current_question_index += 1
        if self.current_question_index >= 20:
            self.show_result_screen()
        else:
            self.show_quiz_screen()

    def go_to_question(self, index):
        if self.selected_answer != -1:
            self.answers[self.current_question_index] = self.selected_answer
        self.current_question_index = index
        self.show_quiz_screen()

    def use_powerup(self, name):
        if self.time_left <= 0:
            return
        cost_minutes = {"Clue": 5, "50:50": 10, "Reveal": 20}.get(name)
        if cost_minutes is None:
            return
        cost_seconds = cost_minutes * 60
        if self.time_left < cost_seconds:
            QMessageBox.warning(self, "Waktu Tidak Cukup",
                                f"Butuh minimal {cost_minutes} menit untuk {name}.\n"
                                f"Sisa waktu: {self.time_left//60} menit.")
            return
        self.time_left -= cost_seconds
        self.timer_label.setText(self.format_time(self.time_left))
        if not hasattr(self, 'option_buttons') or len(self.option_buttons) != 4:
            return
        q = self.questions[self.current_question_index]
        if name == "Clue":
            clue = q.get("clue", "Tidak ada petunjuk tersedia.")
            QMessageBox.information(self, "💡 Petunjuk", clue)
        elif name == "50:50":
            correct_idx = q["answer"]
            wrong_indices = [i for i in range(4) if i != correct_idx]
            to_remove = random.sample(wrong_indices, 2)
            for i in to_remove:
                if i < len(self.option_buttons):
                    btn = self.option_buttons[i]
                    btn.setText("(Dihilangkan)")
                    btn.setEnabled(False)
                    btn.setStyleSheet("color: #94a3b8;")
        elif name == "Reveal":
            correct_idx = q["answer"]
            for i, btn in enumerate(self.option_buttons):
                if i == correct_idx:
                    btn.setText(btn.text().rstrip(" ✅") + " ✅")
                    btn.setStyleSheet("""
                        QRadioButton {
                            background: #10b981; color: white;
                            border-radius: 10px; padding: 14px;
                            font-size: 16px; font-weight: bold;
                        }
                        QRadioButton::indicator { width: 0; height: 0; }
                    """)
                else:
                    btn.setEnabled(False)

    def confirm_exit_quiz(self):
        reply = QMessageBox.question(
            self, "Konfirmasi",
            "Apakah Anda yakin ingin keluar? Waktu tetap berjalan.",
            QMessageBox.Yes | QMessageBox.No
        )
        if reply == QMessageBox.Yes:
            self.show_dashboard()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setStyle("Fusion")
    dark_palette = QPalette()
    dark_palette.setColor(QPalette.Window, QColor(15, 23, 42))
    dark_palette.setColor(QPalette.WindowText, Qt.white)
    dark_palette.setColor(QPalette.Base, QColor(30, 41, 59))
    dark_palette.setColor(QPalette.AlternateBase, QColor(50, 50, 50))
    dark_palette.setColor(QPalette.ToolTipBase, Qt.white)
    dark_palette.setColor(QPalette.ToolTipText, Qt.white)
    dark_palette.setColor(QPalette.Text, Qt.white)
    dark_palette.setColor(QPalette.Button, QColor(50, 50, 50))
    dark_palette.setColor(QPalette.ButtonText, Qt.white)
    dark_palette.setColor(QPalette.BrightText, Qt.red)
    dark_palette.setColor(QPalette.Link, QColor(42, 130, 218))
    dark_palette.setColor(QPalette.Highlight, QColor(42, 130, 218))
    dark_palette.setColor(QPalette.HighlightedText, Qt.black)
    app.setPalette(dark_palette)
    app.setStyleSheet("QToolTip { color: #ffffff; background-color: #2a2a2a; border: 1px solid white; }")
    window = QuizApp()
    window.show()
    sys.exit(app.exec_())
```

Gambar 3. 14 The code of Degichi Quiz
Source: Personal documentation

This part of the code is responsible for running the quiz process from answering questions, moving between items, using power-ups, keeping the timer active, and ensuring that the user cannot exit the quiz carelessly.

next_question(self) Function **:** Moves the student to the next question after selecting an answer and so role in DEGICHI Quiz is a controls progression, prevents skipping without answering, and calculates score like a CBT or exambro and Quizizz system.

Key Components:

- self.selected_answer → ensures the student chooses an answer first.
- **Compares selected answer with the correct one** → updates score & XP.
- self.current_question_index += 1 → moves forward.
- **If question number exceeds 20** → show_result_screen().
- **Else** → open the next question screen.

go_to_question(self, index) Function : Allows jumping to a specific question (used in review mode or navigation UI). Role in the code to enables controlled navigation between questions while still tracking answers securely.

Key Components :

- Saves selected answer for current question.
- Updates current_question_index to the selected one.
- Reloads the quiz screen.

use_powerup(self, name) Function : Activates special features like Clue, 50:50, or Reveal each costing time.

Key Components:

- cost_minutes → determines how much time is removed.
- Subtracts time from self.time_left.
- For “Clue”: shows a hint.
- For “50:50”: hides two wrong answers.
- For “Reveal”: highlights the correct answer.
- Updates button UI (disabled state, color change)

confirm_exit_quiz(self) Function : Asks for confirmation before leaving the quiz while the timer still runs.

Key Components :

- QMessageBox.question() → confirmation pop-up.
- If "Yes" → go back to dashboard.

The main block (if __name__ == "__main__":) : Starts the application, applies theme, and opens the QuizApp window.

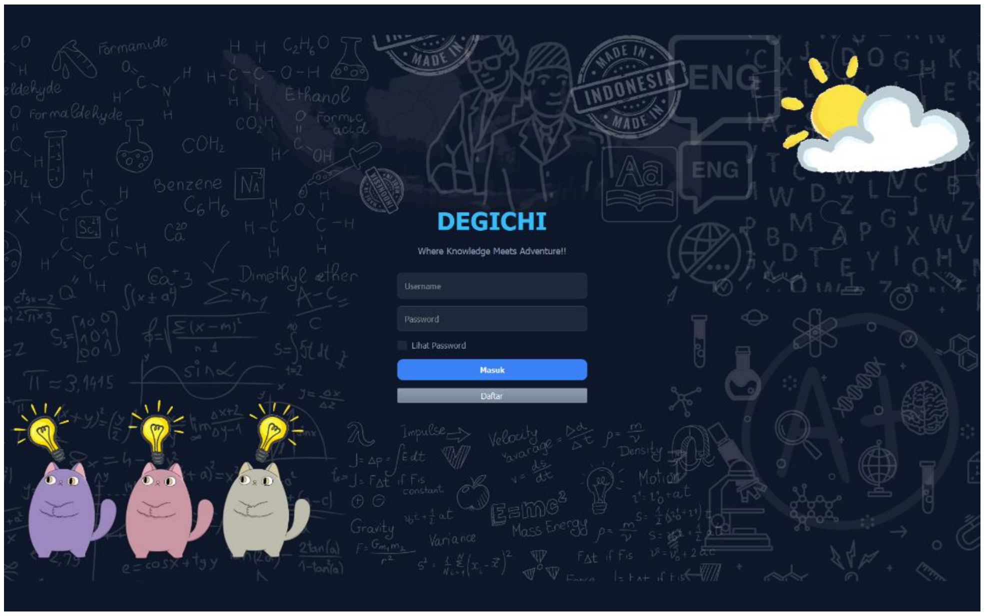
Key Components :

- Sets Fusion style.
- Applies dark color palette.
- Loads custom tooltip style.
- Creates and shows the main window.

3.2 APPLICATION SCREENSHOTS

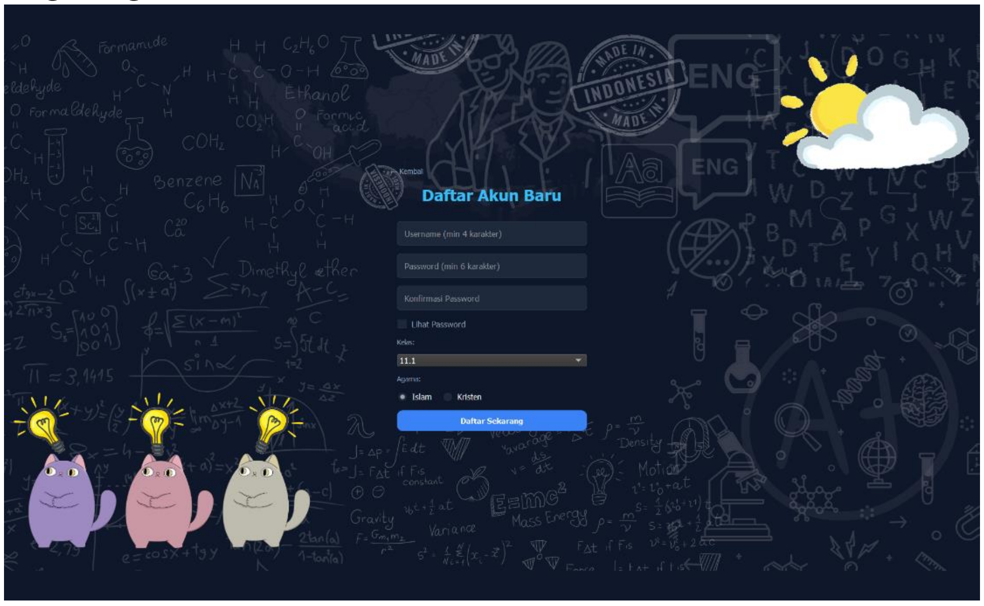
Some things require visualization of this DEGICHI QUIZ which is to support the reader's visualization to be more interpreted, here are screenshots of each slide of this game:

3.2.1 Main Screen



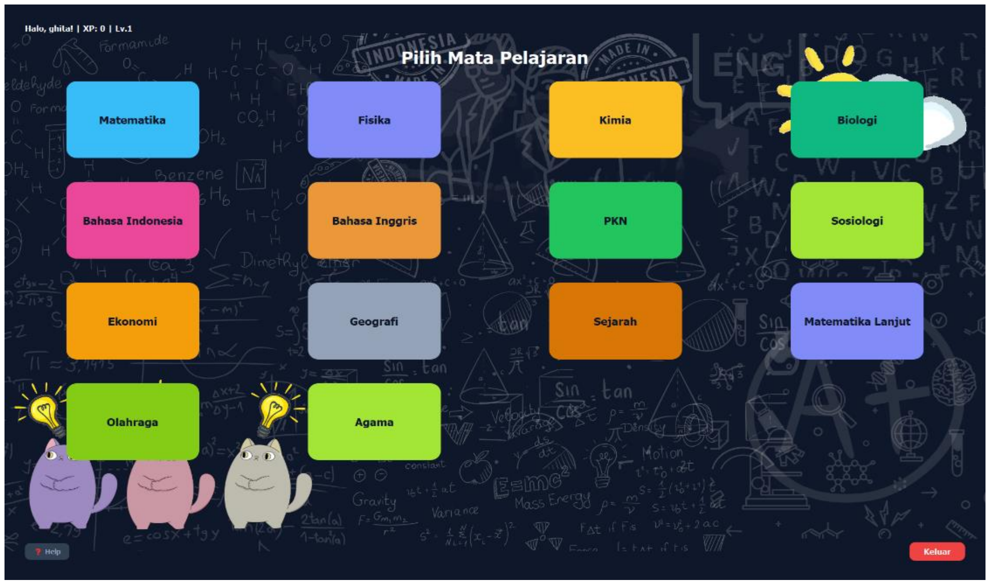
Gambar 3. 15 Main Screen of Degichi Quiz
Source: Personal documentation

3.2.2 Log in register username



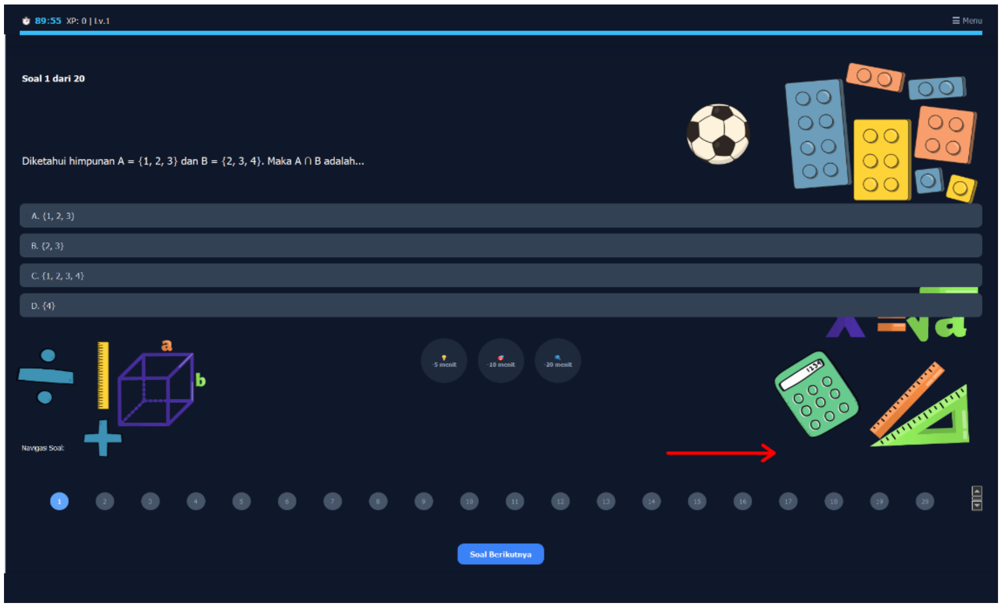
Gambar 3. 16 Log in save in
Source: Personal documentation

3.2.3 Select category



Gambar 3. 17 Category Selection
Source: Personal documentation

3.2.4 Main Quiz in one of the options



89:55 XP: 0 | Lv.1

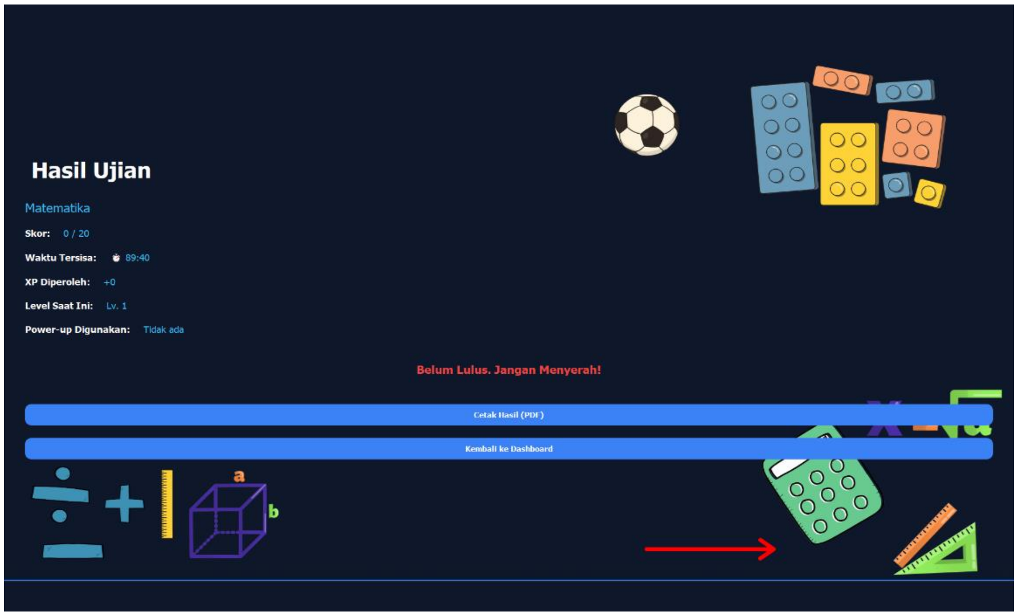
Soal 1 dari 20

Diketahui himpunan A = {1, 2, 3} dan B = {2, 3, 4}. Maka A ∩ B adalah...

A. {1, 2, 3}
B. {2, 3}
C. {1, 2, 3, 4}
D. {4}

Gambar 3. 18 Main Quiz
Source: Personal documentation

3.2.5 Quiz Result in one of the options



Hasil Ujian

Matematika

Skor: 0 / 20

Waktu Tersisa: 89:40

XP Diperoleh: +0

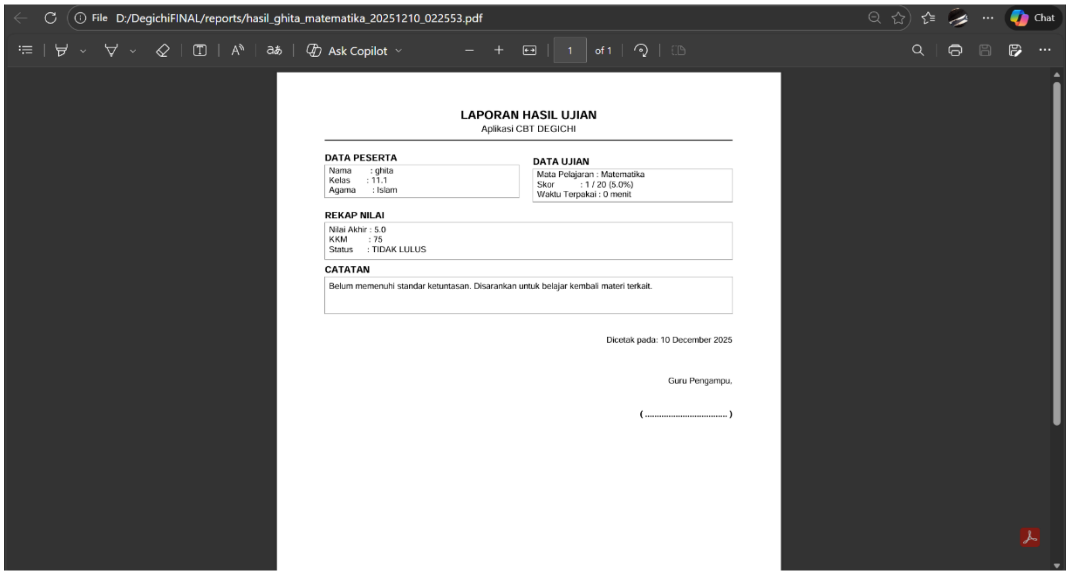
Level Saat Ini: Lv. 1

Power-up Digunakan: Tidak ada

Belum Lulus. Jangan Menyerah!

Gambar 3. 19 Quiz Result
Source: Personal documentation

3.2.6 Result in PDF



File D:/DegichiFINAL/reports/hasil_ghita_matematika_20251210_022553.pdf

LAPORAN HASIL UJIAN
Aplikasi CBT DEGICHI

DATA PESERTA

Nama	: ghita
Kelas	: 11.1
Agama	: Islam

DATA UJIAN

Mata Pelajaran	: Matematika
Skor	: 1 / 20 (5.0%)
Waktu Terpakai	: 0 menit

REKAP NILAI

Nilai Akhir	: 5.0
KKM	: 75
Status	: TIDAK LULUS

CATATAN

Belum memenuhi standar ketuntasan. Disarankan untuk belajar kembali materi terkait.

Dicetak pada: 10 December 2025

Guru Pengampu,

(......................................)

Gambar 3. 20 Result in PDF
Source: Personal documentation

CHAPTER 4
ATTACHMENT

4.1 Embed File

Embed below is the complete folder of assets and the full code of *DEGICHI QUIZ*

DEGICHI QUIZ DRIVE LINK

REFERENCES

Citra, C. A., & Rosy, B. (2020). Keefektifan penggunaan media pembelajaran berbasis game edukasi Quizizz terhadap hasil belajar teknologi perkantoran siswa kelas X SMK Ketintang Surabaya. *Jurnal Pendidikan Administrasi Perkantoran (JPAP),* 8(2).

Marlina., & Juanda, M. F. (2025). Efektivitas penggunaan aplikasi Exambro pada evaluasi pembelajaran geografi di Sekolah Menengah Atas Provinsi Sulawesi Selatan. *Geosfera: Jurnal Penelitian Geografi (GeoJPG),* 4**(**1), 125–134.

Dawson, P., Van der Meer, J., Skalicky, J., & Cowley, K. (2018). *Adaptive quizzes to increase motivation, engagement and learning outcomes in a first year accounting unit*. International Journal of Educational Technology in Higher Education, 15(30).

Awdry, R., & Newton, P. (2023). *How common is cheating in online exams, and did it increase during the COVID-19 pandemic? A systematic review.* Journal of Academic Ethics.

UNESCO. (2023). *Global Education Monitoring Report 2023*. UNESCO Publishing.