

main.py

```
import sys import os import random import hashlib import subprocess import json import asyncio import
shutil from PyQt6.QtWidgets import ( QApplication, QMainWindow, QWidget, QTabWidget, QVBoxLayout,
QHBoxLayout, QLabel, QPushButton, QLineEdit, QComboBox, QTableWidget, QTableWidgetItem, QSlider,
QFormLayout, QTextEdit, QFrame, QMessageBox, QFileDialog ) from PyQt6.QtCore import Qt, QUrl from
PyQt6.QtMultimedia import QMediaPlayer, QAudioOutput from PyQt6.QtGui import QFont
```

Third-Party Tooling

```
import genanki import edge_tts from PIL import Image, ImageDraw, ImageFont
```

Internal Project Module Imports

```
from database.connection import init_db, get_connection from core.bulk_importer import BulkImporter
from core.clean_glossary import GlossaryCleaner
```

```
class SpanishTrainerApp(QMainWindow): def init(self): super().init() self.setWindowTitle("Castilian Voice
Trainer Pro") self.setMinimumSize(1200, 800)
```

```
# 1. Initialize schema structures and check ingestion status
self.ensure_database_populated()

# Load system persistent settings from DB (including sleep-learning
fields)
self.load_system_settings()

# Audio Player Architecture Setup
self.media_player = QMediaPlayer()
self.audio_output = QAudioOutput()
self.media_player.setAudioOutput(self.audio_output)

self.current_flashcard_id = None
self.flashcard_ids_pool = [] # Tracks currently filtered study list
IDs

# Central Main Window Tabs Interface
self.tabs = QTabWidget()
self.setCentralWidget(self.tabs)

self.init_phrase_sandbox_tab()
self.init_flashcard_reviewer_tab()
self.init_settings_tab()

# 2. Populate table grids on initialization
self.refresh_crud_table()
self.refresh_review_table()
```

```

def ensure_database_populated(self):
    """Forces database configuration structure and triggers pipeline
    execution if empty."""
    print("🔍 Verification Pass: Running schema configuration scripts...")
    init_db()

    conn = get_connection()
    cursor = conn.cursor()

    # Ensure our settings table and key columns are structurally sound
    cursor.execute("CREATE TABLE IF NOT EXISTS settings (key TEXT PRIMARY
    KEY, value TEXT);")

    # Seamlessly inject duration column into phrases if it doesn't exist
    cursor.execute("PRAGMA table_info(phrases);")
    columns = [row[1] for row in cursor.fetchall()]
    if "duration" not in columns:
        print("🔄 Modifying phrases schema to support floating-point
    duration tracking...")
        cursor.execute("ALTER TABLE phrases ADD COLUMN duration REAL;")
        conn.commit()

    try:
        cursor.execute("SELECT COUNT(*) FROM translations")
        count = cursor.fetchone()[0]
        print(f"📊 Current Translation Pairs found in database: {count}")
    except Exception as e:
        print(f"⚠️ Table check encountered an issue (likely empty tables):
    {e}")
        count = 0
    finally:
        conn.close()

    if count == 0:
        print("📁 Database tables are empty. Triggering glossary reader
    pipeline...")
        pdf_file = "aula_int_plus_1_glos_en_alfa.pdf"

        if os.path.exists(pdf_file):
            importer = BulkImporter()
            importer.import_pdf_glossary(pdf_file, "Aula Internacional Plus
    1")

            cleaner = GlossaryCleaner()
            cleaner.process_database_clean()
            print("🌟 Ingestion pipeline processing sequence successfully
    completed.")
        else:
            print(f"❌ Error: Source document '{pdf_file}' is missing from
    the directory root.")

def load_system_settings(self):
    """Loads persistent variables from the key-value settings table."""

```

```

# Baseline internal fallback defaults
self.anki_export_dir = os.getcwd()
self.video_export_dir = os.getcwd()
self.video_first_lang = "English First (en -> es)"
self.video_repeats_count = "3"
self.video_pause_duration = "4.0"

conn = get_connection()
cursor = conn.cursor()
try:
    cursor.execute("SELECT key, value FROM settings")
    rows = cursor.fetchall()
    for row in rows:
        if row[0] == "anki_export_directory":
            self.anki_export_dir = row[1]
        elif row[0] == "video_export_directory":
            self.video_export_dir = row[1]
        elif row[0] == "video_first_language":
            self.video_first_lang = row[1]
        elif row[0] == "video_repeats_count":
            self.video_repeats_count = row[1]
        elif row[0] == "video_pause_duration":
            self.video_pause_duration = row[1]
except Exception as e:
    print(f"⚠️ Failed to read application settings from database:
{e}")
finally:
    conn.close()

def save_setting_to_db(self, key, value):
    """Updates or inserts a specific system runtime variable into the
    database."""
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("INSERT OR REPLACE INTO settings (key, value) VALUES
(?, ?)", (key, value))
        conn.commit()
    except Exception as e:
        print(f"❌ Critical: Failed to save setting '{key}': {e}")
    finally:
        conn.close()

def get_or_generate_audio_duration(self, phrase_id, text_str, lang):
    """
    Ensures a target audio track exists on disk, reads its run length
    via ffprobe, caches the duration field inside SQLite, and returns the
    float timing block.
    """
    safe_name = "".join([c for c in text_str if c.isalnum() or c in (" ",
"_")]).strip().replace(" ", "_").lower()
    os.makedirs("media", exist_ok=True)
    target_file = f"media/{safe_name}_{lang}_female.mp3"

```

```

# 1. Generate audio track dynamically if missing
if not os.path.exists(target_file):
    try:
        voice = "es-ES-ElviraNeural" if lang == "es" else "en-GB-
SoniaNeural"
        communicate = edge_tts.Communicate(text_str, voice)
        asyncio.run(communicate.save(target_file))
    except Exception as tts_err:
        print(f"❌ Core TTS System Exception: {tts_err}")
        return 2.5

# 2. Return cached value from DB if it exists
conn = get_connection()
cursor = conn.cursor()
cursor.execute("SELECT duration FROM phrases WHERE id = ?",
(phrase_id,))
cached_row = cursor.fetchone()

if cached_row and cached_row[0] is not None:
    conn.close()
    return float(cached_row[0])

# 3. Calculate audio duration via ffprobe and store it
try:
    cmd = [
        'ffprobe', '-v', 'quiet', '-print_format', 'json',
        '-show_entries', 'format=duration', target_file
    ]
    result = subprocess.run(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)
    data = json.loads(result.stdout)
    duration = float(data['format']['duration'])

    cursor.execute("UPDATE phrases SET duration = ? WHERE id = ?",
(duration, phrase_id))
    conn.commit()
    print(f"📁 Cached duration mapping: {duration}s -> ID
{phrase_id}")
except Exception as e:
    print(f"⚠️ Track structure analysis warning for {target_file}:
{e}")
    duration = 2.5
finally:
    conn.close()

return duration

# =====
# 📁 TAB 1: TRANSLATION-CENTRIC PHRASE SANDBOX (CRUD)
# =====
def init_phrase_sandbox_tab(self):
    tab = QWidget()
    layout = QHBoxLayout(tab)

```

```
left_panel = QVBoxLayout()

filter_layout = QHBoxLayout()
filter_layout.addWidget(QLabel("🔍 Text Filter:"))
self.search_text_input = QLineEdit()
self.search_text_input.setPlaceholderText("Search Spanish or English text blocks...")
self.search_text_input.textChanged.connect(self.refresh_crud_table)
filter_layout.addWidget(self.search_text_input)

filter_layout.addWidget(QLabel("📁 Context:"))
self.search_context_input = QLineEdit()
self.search_context_input.setPlaceholderText("e.g. U2")
self.search_context_input.setMaximumWidth(130)
self.search_context_input.textChanged.connect(self.refresh_crud_table)
filter_layout.addWidget(self.search_context_input)

left_panel.addLayout(filter_layout)

self.translation_table = QTableWidgetItem()
self.translation_table.setColumnCount(6)
self.translation_table.setHorizontalHeaderLabels([
    "TX ID", "Spanish Phrase", "English Translation", "Type", "Source Context", "Deck Assignment"
])

self.translation_table.itemSelectionChanged.connect(self.handle_table_row_select)
left_panel.addWidget(self.translation_table)

nav_layout = QHBoxLayout()
self.btn_row_up = QPushButton("⬆️ Previous Pair")
self.btn_row_down = QPushButton("Next Pair ⬆️")
self.btn_row_up.clicked.connect(lambda: self.step_table_row(-1))
self.btn_row_down.clicked.connect(lambda: self.step_table_row(1))
nav_layout.addWidget(self.btn_row_up)
nav_layout.addWidget(self.btn_row_down)
left_panel.addLayout(nav_layout)

right_panel = QVBoxLayout()
form_frame = QFrame()
form_frame.setFrameShape(QFrame.Shape.StyledPanel)
form_layout = QFormLayout(form_frame)

self.input_tx_id = QLineEdit()
self.input_tx_id.setReadOnly(True)
self.input_tx_id.setPlaceholderText("Auto-Increment ID")

self.input_text_es = QTextEdit()
self.input_text_es.setMaximumHeight(75)

self.input_text_en = QTextEdit()
self.input_text_en.setMaximumHeight(75)
```

```

self.combo_type = QComboBox()
self.combo_type.addItem("phrase", "sentence", "noun", "verb",
"adjective"])

self.input_context = QLineEdit()
self.input_context.setPlaceholderText("e.g., U8_5A")

self.input_tags = QLineEdit()
self.input_tags.setPlaceholderText("e.g., irregular_er boots_verb")

self.input_deck_tag = QLineEdit()
self.input_deck_tag.setPlaceholderText("Anki Sub-deck Hierarchy")

button_qss = """
    QPushButton {
        background-color: #f0f0f0;
        border: 1px solid #c0c0c0;
        border-radius: 4px;
        font-size: 11px;
        font-weight: bold;
        color: #333333;
    }
    QPushButton:hover {
        background-color: #e0e0e0;
        border: 1px solid #a0a0a0;
    }
    QPushButton:pressed {
        background-color: #d0d0d0;
    }
    """

es_header_layout = QHBoxLayout()
es_header_layout.setContentsMargins(0, 5, 0, 5)
es_header_layout.addWidget(QLabel("<b>🇪🇸 Castilian Spanish Text
Element:</b>"))

self.btn_play_sandbox_es = QPushButton("Play 🎮 ")
self.btn_play_sandbox_es.setFixedWidth(75)
self.btn_play_sandbox_es.setFixedHeight(24)
self.btn_play_sandbox_es.setStyleSheet(button_qss)
self.btn_play_sandbox_es.clicked.connect(self.handle_sandbox_play_es)
es_header_layout.addWidget(self.btn_play_sandbox_es)

self.combo_speed_es = QComboBox()
self.combo_speed_es.addItem("0.50x", "0.75x", "1.00x", "1.25x",
"1.50x")
self.combo_speed_es.setCurrentText("1.00x")
self.combo_speed_es.setFixedWidth(70)
self.combo_speed_es.setFixedHeight(24)
es_header_layout.addWidget(self.combo_speed_es)
es_header_layout.addStretch()

en_header_layout = QHBoxLayout()
en_header_layout.setContentsMargins(0, 5, 0, 5)

```

```

en_header_layout.addWidget(QLabel("<b>🇬🇧 English Target Translation:
</b>"))

self.btn_play_sandbox_en = QPushButton("Play 🎮 ")
self.btn_play_sandbox_en.setFixedWidth(75)
self.btn_play_sandbox_en.setFixedHeight(24)
self.btn_play_sandbox_en.setStyleSheet(button_qss)
self.btn_play_sandbox_en.clicked.connect(self.handle_sandbox_play_en)
en_header_layout.addWidget(self.btn_play_sandbox_en)

self.combo_speed_en = QComboBox()
self.combo_speed_en.addItem(["0.50x", "0.75x", "1.00x", "1.25x",
"1.50x"])
self.combo_speed_en.setCurrentText("1.00x")
self.combo_speed_en.setFixedWidth(70)
self.combo_speed_en.setFixedHeight(24)
en_header_layout.addWidget(self.combo_speed_en)
en_header_layout.addStretch()

grammar_header_layout = QHBoxLayout()
grammar_header_layout.setContentsMargins(0, 5, 0, 5)
grammar_header_layout.addWidget(QLabel("<b>📖 Grammar / Usage Note:
</b>"))
grammar_header_layout.addStretch()

self.input_grammar_note = QTextEdit()
self.input_grammar_note.setMaximumHeight(75)
self.input_grammar_note.setPlaceholderText("e.g., feminine variant...")

form_layout.addRow("<b>Translation Link ID:</b>", self.input_tx_id)
form_layout.addRow(es_header_layout)
form_layout.addRow(self.input_text_es)
form_layout.addRow(en_header_layout)
form_layout.addRow(self.input_text_en)
form_layout.addRow(grammar_header_layout)
form_layout.addRow(self.input_grammar_note)
form_layout.addRow("Classification Profile:", self.combo_type)
form_layout.addRow("Source Context ID (Raw):", self.input_context)
form_layout.addRow("Anki Note Tags:", self.input_tags)
form_layout.addRow("<b>Target Deck Scope:</b>", self.input_deck_tag)

crud_buttons = QHBoxLayout()
self.btn_save = QPushButton("➕ Create Pair")
self.btn_update = QPushButton("📄 Update Node")
self.btn_delete = QPushButton("🗑 Sever Link")

self.btn_save.clicked.connect(self.crud_create_pair)
self.btn_update.clicked.connect(self.crud_update_pair)
self.btn_delete.clicked.connect(self.crud_delete_pair)

crud_buttons.addWidget(self.btn_save)
crud_buttons.addWidget(self.btn_update)
crud_buttons.addWidget(self.btn_delete)

```

```

        right_panel.addWidget(QLabel("<h3>Translation Node Management
Matrix</h3>"))
        right_panel.addWidget(form_frame)
        right_panel.addLayout(crud_buttons)
        right_panel.addStretch()

        layout.addLayout(left_panel, stretch=4)
        layout.addLayout(right_panel, stretch=3)

        self.tabs.addTab(tab, "📝 Phrase Sandbox (CRUD)")

# =====
# 📇 TAB 2: FLASHCARD STUDY MODULE
# =====
def init_flashcard_reviewer_tab(self):
    tab = QWidget()
    layout = QHBoxLayout(tab)

    left_panel = QVBoxLayout()

    filter_layout = QHBoxLayout()
    filter_layout.addWidget(QLabel("📁 Context:"))
    self.review_context_filter = QLineEdit()
    self.review_context_filter.setPlaceholderText("Filter Context...")

self.review_context_filter.textChanged.connect(self.refresh_review_table)
    filter_layout.addWidget(self.review_context_filter)

    filter_layout.addWidget(QLabel("🏷️ Tag:"))
    self.review_tag_filter = QLineEdit()
    self.review_tag_filter.setPlaceholderText("Filter Tag...")
    self.review_tag_filter.textChanged.connect(self.refresh_review_table)
    filter_layout.addWidget(self.review_tag_filter)

    left_panel.addLayout(filter_layout)

    self.review_table = QTableWidgetItem()
    self.review_table.setColumnCount(4)
    self.review_table.setHorizontalHeaderLabels(["Tx ID", "Spanish Phrase",
"Context", "Tags"])

self.review_table.itemSelectionChanged.connect(self.handle_review_table_sel
ect)
    left_panel.addWidget(self.review_table)

    layout.addLayout(left_panel, stretch=4)

    right_panel = QVBoxLayout()

    card_frame = QFrame()
    card_frame.setStyleSheet("background-color: #ffffff; border: 2px solid
#bdc3c7; border-radius: 12px;")
    card_layout = QVBoxLayout(card_frame)
    card_frame.setMinimumHeight(280)

```



```
self.lbl_card_text = QLabel("Select a row or click 'Next Card' to  
initiate...")  
self.lbl_card_text.setAlignment(Qt.AlignmentFlag.AlignCenter)  
self.lbl_card_text.setFont(QFont("Arial", 20, QFont.Weight.Bold))  
self.lbl_card_text.setWordWrap(True)  
self.lbl_card_text.setStyleSheet("color: #2c3e50; border: none;  
padding: 20px;")  
  
self.lbl_card_meta = QLabel("")  
self.lbl_card_meta.setAlignment(Qt.AlignmentFlag.AlignCenter)  
self.lbl_card_meta.setFont(QFont("Arial", 11))  
self.lbl_card_meta.setStyleSheet("color: #7f8c8d; border: none;")  
  
card_layout.addStretch()  
card_layout.addWidget(self.lbl_card_text)  
card_layout.addWidget(self.lbl_card_meta)  
card_layout.addStretch()  
right_panel.addWidget(card_frame, stretch=4)  
  
playback_layout = QHBoxLayout()  
playback_layout.addWidget(QLabel("🔊 Voice Speed:"))  
self.slider_review_speed = QSlider(Qt.Orientation.Horizontal)  
self.slider_review_speed.setMinimum(50)  
self.slider_review_speed.setMaximum(150)  
self.slider_review_speed.setValue(100)  
self.lbl_review_speed = QLabel("1.00x")  
  
self.slider_review_speed.valueChanged.connect(self.handle_live_speed_change  
)  
  
playback_layout.addWidget(self.slider_review_speed)  
playback_layout.addWidget(self.lbl_review_speed)  
right_panel.addLayout(playback_layout)  
  
action_buttons = QHBoxLayout()  
self.btn_play_voice = QPushButton("🔊 Play Voice Track")  
self.btn_flip_card = QPushButton("👁️ Reveal English Partner")  
  
self.btn_play_voice.clicked.connect(self.handle_play_voice)  
self.btn_flip_card.clicked.connect(self.handle_flip_card)  
  
action_buttons.addWidget(self.btn_play_voice)  
action_buttons.addWidget(self.btn_flip_card)  
right_panel.addLayout(action_buttons)  
  
right_panel.addSpacing(15)  
  
bottom_utility_layout = QHBoxLayout()  
self.btn_export_anki = QPushButton("📦 Export Anki Deck")  
self.btn_export_video = QPushButton("🎬 Export Video")  
self.btn_load_next = QPushButton("➡️ Next Card")  
  
self.btn_export_anki.clicked.connect(self.handle_export_anki_deck)  
self.btn_export_video.clicked.connect(self.handle_export_video_assets)
```

```

self.btn_load_next.clicked.connect(self.handle_load_next_card)

utility_qss = "QPushButton { font-weight: bold; background-color:
#eaf2f8; padding: 6px; border-radius: 4px; }"
self.btn_export_anki.setStyleSheet(utility_qss)
self.btn_export_video.setStyleSheet(utility_qss)
self.btn_load_next.setStyleSheet("QPushButton { font-weight: bold;
background-color: #d5f5e3; padding: 6px; border-radius: 4px; }")

bottom_utility_layout.addWidget(self.btn_export_anki)
bottom_utility_layout.addWidget(self.btn_export_video)
bottom_utility_layout.addStretch()
bottom_utility_layout.addWidget(self.btn_load_next)
right_panel.addLayout(bottom_utility_layout)

layout.addLayout(right_panel, stretch=3)
self.tabs.addTab(tab, "📇 Flashcard Review")

# =====
# ⚙️ TAB 3: SYSTEM HARDWARE & EXPORT SETTINGS
# =====
def init_settings_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)

    settings_frame = QFrame()
    settings_frame.setFrameShape(QFrame.Shape.StyledPanel)
    form_layout = QFormLayout(settings_frame)

    anki_layout = QHBoxLayout()
    self.line_anki_dir = QLineEdit(self.anki_export_dir)
    self.line_anki_dir.setReadOnly(True)
    btn_browse_anki = QPushButton("Browse 📁")
    btn_browse_anki.clicked.connect(self.handle_browse_anki_directory)
    anki_layout.addWidget(self.line_anki_dir)
    anki_layout.addWidget(btn_browse_anki)

    video_layout = QHBoxLayout()
    self.line_video_dir = QLineEdit(self.video_export_dir)
    self.line_video_dir.setReadOnly(True)
    btn_browse_video = QPushButton("Browse 📁")
    btn_browse_video.clicked.connect(self.handle_browse_video_directory)
    video_layout.addWidget(self.line_video_dir)
    video_layout.addWidget(btn_browse_video)

    # UI Sleep Learning Configuration Fields
    self.combo_first_lang = QComboBox()
    self.combo_first_lang.addItems(["English First (en -> es)", "Spanish
First (es -> en)"])
    self.combo_first_lang.setCurrentText(self.video_first_lang)
    self.combo_first_lang.currentTextChanged.connect(lambda v:
self.save_setting_to_db("video_first_language", v))

    self.spin_video_repeats = QLineEdit(self.video_repeats_count)

```

```

        self.spin_video_repeats.setFixedWidth(60)
        self.spin_video_repeats.textChanged.connect(lambda v:
self.save_setting_to_db("video_repeats_count", v))

        self.spin_pause_duration = QLineEdit(self.video_pause_duration)
        self.spin_pause_duration.setFixedWidth(60)
        self.spin_pause_duration.textChanged.connect(lambda v:
self.save_setting_to_db("video_pause_duration", v))

        form_layout.addRow("<b>Anki Deck Export Destination:</b>", anki_layout)
        form_layout.addRow("<b>Video Assembly Output Target:</b>",
video_layout)
        form_layout.addRow("<b>Introductory Anchor Audio Language:</b>",
self.combo_first_lang)
        form_layout.addRow("<b>Target Translation Loop Multiplier (Repeats):
</b>", self.spin_video_repeats)
        form_layout.addRow("<b>User Recall Repetition Frame Intermission
(Seconds):</b>", self.spin_pause_duration)

        # High-visibility sync button to calculate missing timings and rebuild
metadata cache
        self.btn_sync_cache = QPushButton("< Populate Audio & Timings Cache")
        self.btn_sync_cache.setStyleSheet("""
            QPushButton {
                font-weight: bold;
                background-color: #e67e22;
                color: white;
                padding: 10px;
                border-radius: 5px;
                font-size: 13px;
            }
            QPushButton:hover { background-color: #d35400; }
            """)

        self.btn_sync_cache.clicked.connect(self.handle_bulk_populate_audio_cache)

        layout.addWidget(QLabel("<h2>Application Preferences & Workspace
Routing</h2>"))
        layout.addWidget(settings_frame)
        layout.addWidget(self.btn_sync_cache)
        layout.addStretch()

        self.tabs.addTab(tab, "⚙ Settings")

def handle_browse_anki_directory(self):
    directory = QFileDialog.getExistingDirectory(self, "Select Anki Export
Folder", self.anki_export_dir)
    if directory:
        self.anki_export_dir = directory
        self.line_anki_dir.setText(directory)
        self.save_setting_to_db("anki_export_directory", directory)

def handle_browse_video_directory(self):
    directory = QFileDialog.getExistingDirectory(self, "Select Video Export

```

```

Folder", self.video_export_dir)
    if directory:
        self.video_export_dir = directory
        self.line_video_dir.setText(directory)
        self.save_setting_to_db("video_export_directory", directory)

def handle_bulk_populate_audio_cache(self):
    """Iterates through all relational links, runs dynamic downloads,
    analyzes audio runtime lengths via ffprobe."""
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT p1.id, p1.text, p2.id, p2.text
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
    """)
    records = cursor.fetchall()
    conn.close()

    if not records:
        QMessageBox.information(self, "Cache Synchronizer", "No valid
        translation pairs exist inside the database to process.")
        return

    print(f"< Processing structural cache updates for {len(records)} node
    linkages...")
    for row in records:
        es_id, es_text, en_id, en_text = row[0], row[1].strip(), row[2],
        row[3].strip()
        self.get_or_generate_audio_duration(en_id, en_text, "en")
        self.get_or_generate_audio_duration(es_id, es_text, "es")

    QMessageBox.information(self, "Cache Processing Complete", "All missing
    speech segments successfully written. Timings cached safely.")

# =====
# < DATA MATRIX CONTROL & SYNCHRONIZATION VIEWS
# =====
def refresh_crud_table(self):
    conn = get_connection()
    cursor = conn.cursor()

    text_filter = self.search_text_input.text().strip()
    context_filter = self.search_context_input.text().strip()

    query = """
        SELECT t.translation_id, p1.text, p2.text, p1.word_type,
        p1.source_context, t.deck_name
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
        WHERE p1.language = 'es' AND p2.language = 'en'
    """

```

```
params = []

if text_filter:
    query += " AND (p1.text LIKE ? OR p2.text LIKE ?)"
    params.extend([f"%{text_filter}%", f"%{text_filter}%"])

if context_filter:
    query += " AND p1.source_context LIKE ?"
    params.append(f"%{context_filter}%")

query += " ORDER BY t.translation_id ASC LIMIT 250"

cursor.execute(query, params)
rows = cursor.fetchall()
conn.close()

self.translation_table.setRowCount(0)
for row_idx, row_data in enumerate(rows):
    self.translation_table.insertRow(row_idx)
    for col_idx in range(6):
        val = row_data[col_idx]
        self.translation_table.setItem(row_idx, col_idx,
            QTableWidgetItem(str(val if val is not None else "")))

def refresh_review_table(self):
    conn = get_connection()
    cursor = conn.cursor()

    context_filter = self.review_context_filter.text().strip()
    tag_filter = self.review_tag_filter.text().strip()

    query = """
        SELECT t.translation_id, p1.text, p1.source_context, t.tags, p1.id
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        WHERE p1.language = 'es'
    """

    params = []
    if context_filter:
        query += " AND p1.source_context LIKE ?"
        params.append(f"%{context_filter}%")
    if tag_filter:
        query += " AND t.tags LIKE ?"
        params.append(f"%{tag_filter}%")

    query += " ORDER BY t.translation_id ASC"

    cursor.execute(query, params)
    rows = cursor.fetchall()
    conn.close()

    self.review_table.setRowCount(0)
    self.flashcard_ids_pool = []
```

```

        for row_idx, row_data in enumerate(rows):
            self.review_table.insertRow(row_idx)
            self.flashcard_ids_pool.append(row_data[4])
            for col_idx in range(4):
                val = row_data[col_idx]
                self.review_table.setItem(row_idx, col_idx,
                    QTableWidgetItem(str(val if val is not None else "")))

def handle_table_row_select(self):
    selected_ranges = self.translation_table.selectedRanges()
    if not selected_ranges:
        return
    row = selected_ranges[0].topRow()
    tx_id_item = self.translation_table.item(row, 0)
    if not tx_id_item:
        return

    tx_id = tx_id_item.text()

    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT t.translation_id, p1.text, p2.text, p1.word_type,
p1.source_context, t.deck_name, t.notes, t.tags, p1.id, p2.id
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
        WHERE t.translation_id = ?
    """, (tx_id,))
    record = cursor.fetchone()
    conn.close()

    if record:
        self.input_tx_id.setText(str(record[0]))
        self.input_text_es.setPlainText(str(record[1]))
        self.input_text_en.setPlainText(str(record[2]))
        self.combo_type.setCurrentText(str(record[3]) if record[3] else
"phrase")
        self.input_context.setText(str(record[4]) if record[4] else "")
        self.input_tags.setText(str(record[7]) if record[7] is not None
else "")
        self.input_deck_tag.setText(str(record[5]) if record[5] else
"General")
        self.input_grammar_note.setPlainText(str(record[6]) if record[6] is
not None else "")
        self.current_sandbox_es_id = record[8]
        self.current_sandbox_en_id = record[9]

def handle_review_table_select(self):
    selected_ranges = self.review_table.selectedRanges()
    if not selected_ranges:
        return
    row = selected_ranges[0].topRow()
    phrase_id = self.flashcard_ids_pool[row]

```

```

        self.load_flashcard_by_id(phrase_id)

def step_table_row(self, direction):
    current_row = self.translation_table.currentRow()
    next_row = current_row + direction
    if 0 <= next_row < self.translation_table.rowCount():
        self.translation_table.setCurrentCell(next_row, 0)

# =====
# + CRUD ENGINE ATOMIC OPERATIONS LOGIC
# =====
def crud_create_pair(self):
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO phrases (text, language, word_type, source_context)
VALUES (?, 'es', ?, ?)
        """, (self.input_text_es.toPlainText().strip(),
self.combo_type.currentText(), self.input_context.text().strip()))
    es_id = cursor.lastrowid

    cursor.execute("""
        INSERT INTO phrases (text, language, word_type, source_context)
VALUES (?, 'en', ?, ?)
        """, (self.input_text_en.toPlainText().strip(),
self.combo_type.currentText(), self.input_context.text().strip()))
    en_id = cursor.lastrowid

    cursor.execute("""
        INSERT INTO translations (source_phrase_id, target_phrase_id,
deck_name, notes, tags) VALUES (?, ?, ?, ?, ?)
        """, (es_id, en_id, self.input_deck_tag.text().strip() or "General",
self.input_grammar_note.toPlainText().strip(),
self.input_tags.text().strip()))

    conn.commit()
    conn.close()
    self.refresh_crud_table()
    self.refresh_review_table()
    QMessageBox.information(self, "Success", "Isolated phrase pairs created
and relational link bound.")

def crud_update_pair(self):
    tx_id = self.input_tx_id.text()
    if not tx_id:
        return

    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT source_phrase_id, target_phrase_id FROM
translations WHERE translation_id = ?", (tx_id,))
    ids = cursor.fetchone()

```

```

        if ids:
            es_id, en_id = ids
            cursor.execute("UPDATE phrases SET text=?, word_type=?,
source_context=?, duration=NULL WHERE id=?",
                           (self.input_text_es.toPlainText().strip(),
self.combo_type.currentText(), self.input_context.text().strip(), es_id))
            cursor.execute("UPDATE phrases SET text=?, word_type=?,
source_context=?, duration=NULL WHERE id=?",
                           (self.input_text_en.toPlainText().strip(),
self.combo_type.currentText(), self.input_context.text().strip(), en_id))
            cursor.execute("UPDATE translations SET deck_name=?, notes=?,
tags=? WHERE translation_id=?",
                           (self.input_deck_tag.text().strip() or "General",
self.input_grammar_note.toPlainText().strip(),
self.input_tags.text().strip(), tx_id))
            conn.commit()

        conn.close()
        self.refresh_crud_table()
        self.refresh_review_table()
        QMessageBox.information(self, "Success", "Relational node structural
update complete.")

def crud_delete_pair(self):
    tx_id = self.input_tx_id.text()
    if not tx_id:
        return
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT source_phrase_id, target_phrase_id FROM
translations WHERE translation_id = ?", (tx_id,))
    ids = cursor.fetchone()
    if ids:
        es_id, en_id = ids
        cursor.execute("DELETE FROM translations WHERE translation_id=?",
(tx_id,))
        cursor.execute("DELETE FROM phrases WHERE id=?", (es_id,))
        cursor.execute("DELETE FROM phrases WHERE id=?", (en_id,))
        conn.commit()
    conn.close()
    self.refresh_crud_table()
    self.refresh_review_table()
    self.input_tx_id.clear()
    self.input_text_es.clear()
    self.input_text_en.clear()
    self.input_grammar_note.clear()
    self.input_tags.clear()

# =====
# 🎧 AUDIO ENGINE & EXPANDED FLASHCARD ACTIONS
# =====
def load_flashcard_by_id(self, phrase_id):
    conn = get_connection()
    cursor = conn.cursor()

```



```

        cursor.execute("""
            SELECT t.translation_id, p1.text, p1.source_context, t.deck_name,
p1.id, t.tags
            FROM translations t
            JOIN phrases p1 ON t.source_phrase_id = p1.id
            WHERE p1.id = ?
        """, (phrase_id,))
        record = cursor.fetchone()
        conn.close()

        if record:
            self.current_flashcard_id = record[4]
            self.lbl_card_text.setText(record[1])
            self.lbl_card_meta.setText(f"Link ID: {record[0]} • Context:
{record[2]} • Tag: {record[5]} • Deck: {record[3]}")

def handle_load_next_card(self):
    if not self.flashcard_ids_pool:
        QMessageBox.information(self, "Empty Pool", "No flashcards match
your selected filter configurations.")
        return

    target_id = random.choice(self.flashcard_ids_pool)

    try:
        matched_idx = self.flashcard_ids_pool.index(target_id)
        self.review_table.setCurrentCell(matched_idx, 0)
    except ValueError:
        pass

    self.load_flashcard_by_id(target_id)

def handle_play_voice(self):
    if not self.current_flashcard_id:
        return
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT text, language FROM phrases WHERE id = ?",
(self.current_flashcard_id,))
    row = cursor.fetchone()
    conn.close()

    if row:
        text_str, lang = row
        safe_name = "".join([c for c in text_str if c.isalnum() or c in ("
", "_")]).strip().replace(" ", "_").lower()
        target_file = f"media/{safe_name}_{lang}_female.mp3"

        self.get_or_generate_audio_duration(self.current_flashcard_id,
text_str, lang)

        if os.path.exists(target_file):

self.media_player.setSource(QUrl.fromLocalFile(os.path.abspath(target_file)

```

```

))

self.media_player.setPlaybackRate(self.slider_review_speed.value() / 100.0)
    self.media_player.play()

def handle_flip_card(self):
    if not self.current_flashcard_id:
        return
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT p2.text, t.notes FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
        WHERE p1.id = ?
    """, (self.current_flashcard_id,))
    row = cursor.fetchone()
    conn.close()

    if row:
        clean_es = self.lbl_card_text.text().split("\n\n👉")[0]
        display_text = f"{clean_es}\n\n👉 [ {row[0]} ]"
        if row[1]:
            display_text += f"\n\n💡 Note: {row[1]}"
        self.lbl_card_text.setText(display_text)

def handle_sandbox_play_es(self):
    text_str = self.input_text_es.toPlainText().strip()
    if not text_str or not hasattr(self, 'current_sandbox_es_id'):
        return

    self.get_or_generate_audio_duration(self.current_sandbox_es_id,
    text_str, "es")
    safe_name = "".join([c for c in text_str if c.isalnum() or c in (" ",
    "_")]).strip().replace(" ", "_").lower()
    target_file = f"media/{safe_name}_es_female.mp3"

    self.media_player.setSource(QUrl.fromLocalFile(os.path.abspath(target_file)
    ))

    self.media_player.setPlaybackRate(float(self.combo_speed_es.currentText().r
    eplace("x", "")))
    self.media_player.play()

def handle_sandbox_play_en(self):
    text_str = self.input_text_en.toPlainText().strip()
    if not text_str or not hasattr(self, 'current_sandbox_en_id'):
        return

    self.get_or_generate_audio_duration(self.current_sandbox_en_id,
    text_str, "en")
    safe_name = "".join([c for c in text_str if c.isalnum() or c in (" ",
    "_")]).strip().replace(" ", "_").lower()

```

```

target_file = f"media/{safe_name}_en_female.mp3"

self.media_player.setSource(QUrl.fromLocalFile(os.path.abspath(target_file)
))

self.media_player.setPlaybackRate(float(self.combo_speed_en.currentText().r
eplace("x", "")))
self.media_player.play()

def handle_live_speed_change(self, value):
    rate = value / 100.0
    self.lbl_review_speed.setText(f"{rate:.2f}x")
    if self.media_player.playbackState() ==
QMediaPlayer.PlaybackState.PlayingState:
        self.media_player.setPlaybackRate(rate)

# =====
# 📦 ARTIFACT EXPORT GATEWAYS (GENANKI LIVE ENGINE WITH DUAL AUDIO)
# =====
def handle_export_anki_deck(self):
    """Compiles active subset into functional .apkg with bundled Spanish
and English audio tracks."""
    if not self.flashcard_ids_pool:
        QMessageBox.warning(self, "Export Cancelled", "The current study
stack is empty. Verify your search filters.")
        return

    model_hash =
hashlib.sha256(b"castilian_voice_trainer_model_v2").hexdigest()
    model_id = int(model_hash[:13], 16)

    spanish_note_model = genanki.Model(
        model_id,
        'Castilian Audio Flashcard Model v2',
        fields=[
            {'name': 'SpanishPhrase'},
            {'name': 'EnglishTranslation'},
            {'name': 'GrammarNotes'},
            {'name': 'SpanishAudio'},
            {'name': 'EnglishAudio'}
        ],
        templates=[
            {
                'name': 'Card 1: Auditory Identification',
                'qfmt': (
                    '<div style="font-family: Arial; font-size: 24px; text-
align: center; color: #2c3e50;">{{SpanishPhrase}}</div>'
                    '<br><div style="text-align: center;">{{SpanishAudio}}
</div>'
                ),
                'afmt': (
                    '{{FrontSide}}<hr id="answer">'
                    '<div style="font-family: Arial; font-size: 20px; text-

```

```

align: center; color: #27ae60; font-weight: bold;">{{EnglishTranslation}}
</div>'
        '<div style="text-align: center; margin-top: 5px;">
{{EnglishAudio}}</div><br>'
        '<div style="font-family: Arial; font-size: 14px; text-
align: center; color: #7f8c8d; font-style: italic;">{{GrammarNotes}}</div>'
    ),
    ],
    css='.card { font-family: arial; font-size: 20px; text-align:
center; background-color: #f8f9fa; }'
)

context_txt = self.review_context_filter.text().strip()
tag_txt = self.review_tag_filter.text().strip()

if context_txt and tag_txt:
    file_title =
f"Spanish_Export_Context_{context_txt}_Tag_{tag_txt}.apkg"
    elif context_txt:
        file_title = f"Spanish_Export_Context_{context_txt}.apkg"
    elif tag_txt:
        file_title = f"Spanish_Export_Tag_{tag_txt}.apkg"
    else:
        file_title = "Spanish_Master_Deck.apkg"

    file_title = "".join([c for c in file_title if c.isalnum() or c in
(".", "_", "-")]).strip()
    destination_path = os.path.join(self.anki_export_dir, file_title)

    decks_map = {}
    media_files_manifest = []

    conn = get_connection()
    cursor = conn.cursor()
    placeholders = ",".join(["?"] * len(self.flashcard_ids_pool))
    query = f"""
        SELECT t.deck_name, p1.text, p2.text, t.notes, t.tags, p1.id, p2.id
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
        WHERE p1.id IN ({placeholders})
    """
    cursor.execute(query, self.flashcard_ids_pool)
    records = cursor.fetchall()
    conn.close()

    for row in records:
        db_deck_name = row[0].strip() if row[0] else "Castilian Spanish
Master"
        es_text, en_text = row[1].strip(), row[2].strip()
        notes_text, tags_string = row[3].strip() if row[3] else "",
row[4].strip() if row[4] else ""
        es_id, en_id = row[5], row[6]

```

```

        self.get_or_generate_audio_duration(es_id, es_text, "es")
        self.get_or_generate_audio_duration(en_id, en_text, "en")

        safe_es = "".join([c for c in es_text if c.isalnum() or c in (" ",
            "_")]).strip().replace(" ", "_").lower()
        safe_en = "".join([c for c in en_text if c.isalnum() or c in (" ",
            "_")]).strip().replace(" ", "_").lower()

        relative_es_path = f"media/{safe_es}_es_female.mp3"
        relative_en_path = f"media/{safe_en}_en_female.mp3"

        media_files_manifest.extend([relative_es_path, relative_en_path])

        if db_deck_name not in decks_map:
            deck_hash = hashlib.sha256(db_deck_name.encode('utf-
8')).hexdigest()
            deck_id = int(deck_hash[:13], 16)
            decks_map[db_deck_name] = genanki.Deck(deck_id, db_deck_name)

        parsed_tags = [t for t in tags_string.replace(",", " ").split(" ")
if t]
        flash_note = genanki.Note(
            model=spanish_note_model,
            fields=[es_text, en_text, notes_text, f"[sound:
{safe_es}_es_female.mp3]", f"[sound:{safe_en}_en_female.mp3]"],
            tags=parsed_tags
        )
        decks_map[db_deck_name].add_note(flash_note)

    try:
        package = genanki.Package(list(decks_map.values()))
        package.media_files = [m for m in set(media_files_manifest) if
os.path.exists(m)]
        package.write_to_file(destination_path)
        QMessageBox.information(self, "Export Complete", f"🌟 Packaged
complete!\nOutput: {file_title}")
    except Exception as export_error:
        QMessageBox.critical(self, "Export Failed", f"Genanki
failure:\n{export_error}")

# =====
# 📺 DYNAMIC SLEEP-LEARNING VIDEO GENERATION LAYER
# =====
def create_video_frame_image(self, text, output_path):
    """Renders a visual slide text frame optimized for dark sleep study
rooms."""
    img = Image.new('RGB', (1920, 1080), color='#111a24')
    canvas = ImageDraw.Draw(img)
    try:
        font = ImageFont.load_default()
    except:
        font = None

```

```

        canvas.text((960, 540), text, fill="#e2e8f0", anchor="mm")
        img.save(output_path)

def handle_export_video_assets(self):
    """Compiles filtered translation pairs into structural sleep loops
    using dynamic timelines."""
    if not self.flashcard_ids_pool:
        QMessageBox.warning(self, "Video Generation Cancelled", "The active
        filter queue contains no records.")
        return

    try:
        repeat_count = int(self.spin_video_repeats.text().strip())
        pause_sec = float(self.spin_pause_duration.text().strip())
    except ValueError:
        QMessageBox.critical(self, "Configuration Error", "Check repeat
        multiplier numbers and decimal timing definitions.")
        return

    temp_dir = os.path.join(os.getcwd(), "video_scratch_pad")
    os.makedirs(temp_dir, exist_ok=True)

    conn = get_connection()
    cursor = conn.cursor()
    placeholders = ",".join(["?"] * len(self.flashcard_ids_pool))
    query = f"""
        SELECT p1.id, p1.text, p2.id, p2.text
        FROM translations t
        JOIN phrases p1 ON t.source_phrase_id = p1.id
        JOIN phrases p2 ON t.target_phrase_id = p2.id
        WHERE p1.id IN ({placeholders})
    """
    cursor.execute(query, self.flashcard_ids_pool)
    records = cursor.fetchall()
    conn.close()

    print(f"🔧 Compiling timeline clips for {len(records)} study
    pairs...")
    video_segment_paths = []

    try:
        for idx, row in enumerate(records):
            es_id, es_text = row[0], row[1].strip()
            en_id, en_text = row[2], row[3].strip()

            es_duration = self.get_or_generate_audio_duration(es_id,
            es_text, "es")
            en_duration = self.get_or_generate_audio_duration(en_id,
            en_text, "en")

            safe_es = "".join([c for c in es_text if c.isalnum() or c in ("
            ", "_)]).strip().replace(" ", "_").lower()
            safe_en = "".join([c for c in en_text if c.isalnum() or c in ("
            ", "_)]).strip().replace(" ", "_").lower()

```

```

es_audio_path = f"media/{safe_es}_es_female.mp3"
en_audio_path = f"media/{safe_en}_en_female.mp3"

    if "English First" in self.combo_first_lang.currentText():
        prime_text, prime_audio, prime_dur = en_text,
en_audio_path, en_duration
        target_text, target_audio, target_dur = es_text,
es_audio_path, es_duration
    else:
        prime_text, prime_audio, prime_dur = es_text,
es_audio_path, es_duration
        target_text, target_audio, target_dur = en_text,
en_audio_path, en_duration

    # --- Clip 1: Anchor Language Ingestion block ---
    img_prime = os.path.join(temp_dir, f"frame_prime_{idx}.png")
    self.create_video_frame_image(prime_text, img_prime)
    clip_prime_path = os.path.join(temp_dir,
f"chunk_prime_{idx}.mp4")

    subprocess.run([
        'ffmpeg', '-y', '-loop', '1', '-i', img_prime, '-i',
prime_audio,
        '-c:v', 'libx264', '-t', str(prime_dur), '-pix_fmt',
'yuv420p',
        '-c:a', 'aac', '-b:a', '192k', clip_prime_path
    ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    video_segment_paths.append(clip_prime_path)

    # --- Clip 2: Target Repeat Frame block ---
    img_target = os.path.join(temp_dir, f"frame_target_{idx}.png")
    self.create_video_frame_image(target_text, img_target)
    clip_target_path = os.path.join(temp_dir,
f"chunk_target_{idx}.mp4")

    subprocess.run([
        'ffmpeg', '-y', '-loop', '1', '-i', img_target, '-i',
target_audio,
        '-c:v', 'libx264', '-t', str(target_dur), '-pix_fmt',
'yuv420p',
        '-c:a', 'aac', '-b:a', '192k', clip_target_path
    ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

    # --- Clip 3: Silent Practice Intermission frame block ---
    clip_silent_path = os.path.join(temp_dir,
f"chunk_silent_{idx}.mp4")
    subprocess.run([
        'ffmpeg', '-y', '-f', 'lavfi', '-i',
f'color=c=#111a24:s=1920x1080:d={pause_sec}',
        '-f', 'lavfi', '-i', 'anullsrc=cl=stereo:r=44100',
        '-t', str(pause_sec), '-c:v', 'libx264', '-pix_fmt',
'yuv420p',
        '-c:a', 'aac', clip_silent_path
    ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

```

```

        ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

        # Loop target segments sequentially matching step 3 -> 4 -> 5
loops
        for _ in range(repeat_count):
            video_segment_paths.append(clip_target_path)
            video_segment_paths.append(clip_silent_path)

        if not video_segment_paths:
            QMessageBox.warning(self, "Export Error", "Timeline compilation
matrix is empty.")
            return

        # --- Concat Loop: Merge all segments into a master movie file ---
        manifest_path = os.path.join(temp_dir, "manifest.txt")
        with open(manifest_path, "w", encoding="utf-8") as f:
            for path in video_segment_paths:
                f.write(f"file '{os.path.abspath(path)}'\n")

        output_file = os.path.join(self.video_export_dir,
"Spanish_Sleep_Learning_Master.mp4")
        subprocess.run([
            'ffmpeg', '-y', '-f', 'concat', '-safe', '0', '-i',
manifest_path,
            '-c', 'copy', output_file
        ], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

        QMessageBox.information(self, "Success", f"Sleep Learning
compilation track generated successfully!\nLocation: {output_file}")

    except Exception as e:
        QMessageBox.critical(self, "Video Synthesis Suite Error",
f"Timeline compiler hit a hitch:\n{e}")
        finally:
            if os.path.exists(temp_dir):
                shutil.rmtree(temp_dir)

```

```

if name == "main": print("🚀 Launching Core PyQt6 Framework Threads...") try: app =
QApplication(sys.argv) window = SpanishTrainerApp() window.show() sys.exit(app.exec()) except
Exception as fatal_error: import traceback traceback.print_exc() sys.exit(1)

```