```python
"""The controller module for the minor misdemeanor dialog - it is not limited
to minor misdemeanors, but does not contain functions to account for jail time.
Loads all charges - including non-minor-misdemeanors from a database."""
import pathlib
from datetime import date, timedelta
from loguru import logger

from PyQt5 import QtCore
from PyQt5.QtCore import QDate
from PyQt5.QtWidgets import QLabel, QPushButton, QMessageBox, QComboBox, QLineEdit
from PyQt5.QtSql import QSqlDatabase, QSqlQuery

from views.custom_widgets import (
    PleaComboBox,
    FindingComboBox,
    FineLineEdit,
    FineSuspendedLineEdit,
    DeleteButton,
    AmendButton,
)
from views.minor_misdemeanor_dialog_ui import Ui_MinorMisdemeanorDialog
from models.template_types import TEMPLATE_DICT
from models.case_information import (
    CaseInformation,
    CriminalCharge,
    AmendOffenseDetails,
    LicenseSuspensionTerms,
    CommunityControlTerms,
    CommunityServiceTerms,
    OtherConditionsTerms,
)
from models.messages import TURNS_AT_INTERSECTIONS as TURNS_WARNING
from controllers.criminal_dialogs import (
    BaseCriminalDialog,
    AddConditionsDialog,
    AmendOffenseDialog,
)
from resources.db.DatabaseCreation import create_offense_list, create_statute_list


PATH = str(pathlib.Path().absolute())
TEMPLATE_PATH = PATH + "\\resources\\templates\\"
SAVE_PATH = PATH + "\\resources\\saved\\"
DB_PATH = PATH + "\\resources\\db\\"
CHARGES_DATABASE = DB_PATH + "\\charges.sqlite"


@logger.catch
def create_database_connections():
    """The databases for the application are created upon import of the module, which is done
    on application startup. The connections to the databases are created, but the opening and
    closing of the databases is handled by the appropriate class dialog."""
    offense_database_connection = QSqlDatabase.addDatabase("QSQLITE", "offenses")
    offense_database_connection.setDatabaseName(CHARGES_DATABASE)
    statute_database_connection = QSqlDatabase.addDatabase("QSQLITE", "statutes")
    statute_database_connection.setDatabaseName(CHARGES_DATABASE)
    return offense_database_connection, statute_database_connection


@logger.catch
def open_databases():
    """
    https://www.tutorialspoint.com/pyqt/pyqt_database_handling.htm
    https://doc.qt.io/qtforpython/overviews/sql-connecting.html
    NOTE: If running create_psql_table.py to update database, must delete
    the old charges.sqlite file to insure it is updated.
    """
    database_offenses.open()
    database_statutes.open()


@logger.catch
def close_databases():
    """Closes any databases that were opened at the start of the dialog."""
    database_offenses.close()
    database_offenses.removeDatabase(CHARGES_DATABASE)
    database_statutes.close()
    database_statutes.removeDatabase(CHARGES_DATABASE)


class MinorMisdemeanorDialog(BaseCriminalDialog, Ui_MinorMisdemeanorDialog):
    """The dialog inherits from the BaseCriminalDialog (controller) and the
    Ui_MinorMisdemeanorDialog (view).

    This dialog is used when there will not be any jail time imposed. It does
    not inherently limit cases to minor misdemeanors or unclassified
    misdemeanors, however, it does not include fields to enter jail time."""

    @logger.catch
    def __init__(self, judicial_officer, parent=None):
        open_databases()
        super().__init__(parent)
        self.case_information = CaseInformation(judicial_officer)
        self.modify_view()
        self.connect_signals_to_slots()
        self.template = TEMPLATE_DICT.get(self.case_information.judicial_officer.last_name)
        self.delete_button_list = []
        self.amend_button_list = []

    @logger.catch
    def modify_view(self):
        """The modify view method updates the view that is created on init from the
        Ui_MinorMisdemeanorDialog. Place items in this method that can't be added
        directly in QtDesigner so that they don't need to be changed in the view file
        each time pyuic5 is run."""
        statute_list = create_statute_list()
        self.statute_choice_box.addItems(statute_list)
        self.offense_choice_box.addItems(create_offense_list())
        self.plea_trial_date.setDate(QtCore.QDate.currentDate())
        self.balance_due_date.setDate(QtCore.QDate.currentDate())
        self.statute_choice_box.setCurrentText("")
        self.offense_choice_box.setCurrentText("")

    @logger.catch
    def connect_signals_to_slots(self):
        """The method connects any signals to slots. Generally, connecting with
        pressed is preferred to clicked because a clicked event sends a bool
        argument to the function. However, clicked is used in some instances
        because it is a press and release of a button. Using pressed sometimes
        caused an event to be triggered twice."""
        self.cancel_Button.pressed.connect(self.close_event)
        self.clear_fields_case_Button.pressed.connect(self.clear_case_information_fields)
        self.create_entry_Button.pressed.connect(self.create_entry_process)
        self.add_conditions_Button.pressed.connect(self.start_add_conditions_dialog)
        self.add_charge_Button.clicked.connect(self.add_charge_process)
        self.clear_fields_charge_Button.pressed.connect(self.clear_charge_fields)
        self.statute_choice_box.currentTextChanged.connect(self.set_offense)
        self.offense_choice_box.currentTextChanged.connect(self.set_statute)
        self.fra_in_file_box.currentTextChanged.connect(self.set_fra_in_file)
        self.fra_in_court_box.currentTextChanged.connect(self.set_fra_in_court)
        self.ability_to_pay_box.currentTextChanged.connect(self.set_pay_date)
        self.guilty_all_Button.pressed.connect(self.guilty_all_plea_and_findings)
        self.no_contest_all_Button.pressed.connect(self.no_contest_all_plea_and_findings)
        self.costs_and_fines_Button.clicked.connect(self.show_costs_and_fines)

    def clear_case_information_fields(self):
        self.defendant_first_name_lineEdit.clear()
        self.defendant_last_name_lineEdit.clear()
        self.case_number_lineEdit.clear()
        self.defendant_first_name_lineEdit.setFocus()

    def clear_charge_fields(self):
        """Clears the fields that are used for adding a charge. The
        statute_choice_box and offense_choice_box use the clearEditText
        method because those boxes are editable."""
        self.statute_choice_box.clearEditText()
        self.offense_choice_box.clearEditText()

    @logger.catch
    def start_amend_offense_dialog(self):
        """Opens the amend offense dialog as a modal window. The
        case_information is passed to the dialog class in order to populate
        the case information banner."""
        self.update_case_information()
        AmendOffenseDialog(self.case_information).exec()

    @logger.catch
    def start_add_conditions_dialog(self):
        """Opens the add conditions dialog as a modal window. It passes the
        instance of the MinorMisdemeanorDialog class (self) as an argument
        so that the AddConditionsDialog can access all data from the
        MinorMisdemeanorDialog when working in the AddConditionsDialog."""
        AddConditionsDialog(self).exec()

    @logger.catch
    def close_event(self):
        """Place any cleanup items (i.e. close_databases) here that should be
        called when the entry is created and the dialog closed."""
        close_databases()
        self.close_window()

    @logger.catch
    def add_charge(self):
        """The add_charge_process, from which this is called, creates a criminal
        charge object and adds the data in the view to the object. The criminal
        charge (offense, statute, degree and type) is then added to the case
        information model (by appending the charge object to the criminal
        charges list).

        The offense, statute and degree are added to the view by the method
        add_charge_to_view, not this method. This method is triggered on
        clicked() of the Add Charge button."""
        self.criminal_charge.offense = self.offense_choice_box.currentText()
        self.criminal_charge.statute = self.statute_choice_box.currentText()
        self.criminal_charge.degree = self.degree_choice_box.currentText()
        self.criminal_charge.type = self.set_offense_type()
        self.case_information.add_charge_to_list(self.criminal_charge)
        self.add_charge_to_view()
        self.statute_choice_box.setFocus()

    @logger.catch
    def add_charge_to_view(self):
        """Adds the charge that was added through add_charge method to the
        view/GUI. The first row=0 because of python zero-based indexing. The
        column is set at one more than the current number of columns because
        it is the column to which the charge will be added.

        :added_charge_index: - The added charge index is one less than the
        total charges in charges_list because of zero-based indexing. Thus, if
        there is one charge, the index of the charge to be added to the
        charge_dict from the charges_list is 0.

        The python builtin vars function returns the __dict__ attribute of
        the object.

        The self.criminal_charge.offense added as a parameter for FineLineEdit
        is the current one added when "Add Charge" is pressed.

        TODO: Refactor so that there isn't a need for a if branch to skip the
        attribute for charge type."""
        row = 0
        column = self.charges_gridLayout.columnCount() + 1
        added_charge_index = len(self.case_information.charges_list) - 1
        charge = vars(self.case_information.charges_list[added_charge_index])
        for value in charge.values():
            if value is not None:
                if value in ["Moving Traffic", "Non-moving Traffic", "Criminal"]:
                    break
                self.charges_gridLayout.addWidget(QLabel(value), row, column)
                row += 1
        self.charges_gridLayout.addWidget(PleaComboBox(), row, column)
        row +=1
        self.charges_gridLayout.addWidget(FindingComboBox(), row, column)
        row +=1
        self.charges_gridLayout.addWidget(FineLineEdit(self.criminal_charge.offense), row, column)
        row +=1
        self.charges_gridLayout.addWidget(FineSuspendedLineEdit(), row, column)
        row +=1
        self.add_delete_button_to_view(row, column)
        row +=1
        self.add_amend_button_to_view(row, column)

    def add_delete_button_to_view(self, row, column):
        delete_button = DeleteButton()
        self.delete_button_list.append(delete_button)
        delete_button.pressed.connect(self.delete_charge)
        self.charges_gridLayout.addWidget(delete_button, row, column)

    def add_amend_button_to_view(self, row, column):
        amend_button = AmendButton()
        self.amend_button_list.append(amend_button)
        amend_button.pressed.connect(self.start_amend_offense_dialog)
        self.charges_gridLayout.addWidget(amend_button, row, column)

    @logger.catch
    def delete_charge(self):
        """Deletes the offense from the case_information.charges list. Then
        decrements the total charges by one so that other functions using the
        total charges for indexing are correct."""
        index = self.delete_button_list.index(self.sender())
        del self.case_information.charges_list[index]
        del self.delete_button_list[index]
        self.delete_charge_from_view()
        self.statute_choice_box.setFocus()

    @logger.catch
    def delete_charge_from_view(self):
        """Uses the delete_button that is indexed to the column to delete the
        QLabels for the charge."""
        index = self.charges_gridLayout.indexOf(self.sender())
        column = self.charges_gridLayout.getItemPosition(index)[1]
        for row in range(self.charges_gridLayout.rowCount()):
            layout_item = self.charges_gridLayout.itemAtPosition(row, column)
            if layout_item is not None:
                layout_item.widget().deleteLater()
                self.charges_gridLayout.removeItem(layout_item)

    @logger.catch
    def update_case_information(self):
        """The method calls functions to update the case information model
        with the data for the case that is in the fields on the view. This does
        not update the model
        with information in the charge fields (offense, statute, plea, etc.)
        the charge information is transferred to the model upon press of the
        add charge button.

        Fields that are updated upon pressed() of createEntryButton = case
        number, first name, last name, ability to pay time, balance due date,
        date of plea/trial,operator license number, date of birth, FRA (proof
        of insurance) in complaint, FRA in court."""
        self.update_party_information()
        self.update_costs_and_fines_information()
        self.add_dispositions_and_fines()
        self.check_add_conditions()
        self.calculate_costs_and_fines()

    @logger.catch
    def update_party_information(self):
        self.case_information.case_number = self.case_number_lineEdit.text()
        self.case_information.defendant.first_name = self.defendant_first_name_lineEdit.text()
        self.case_information.defendant.last_name = self.defendant_last_name_lineEdit.text()
        self.case_information.plea_trial_date = self.plea_trial_date.date().toString("MMMM dd, yyyy")

    @logger.catch
    def update_costs_and_fines_information(self):
        self.case_information.court_costs_ordered = self.court_costs_box.currentText()
        self.case_information.ability_to_pay_time = self.ability_to_pay_box.currentText()
        self.case_information.balance_due_date = self.balance_due_date.date().toString("MMMM dd, yyyy")

    @logger.catch
    def add_dispositions_and_fines(self):
        """Row 3 - plea, 4 - finding, 5 - fine, 6 fine-suspended.
        Columns start at 0 for labels and 2 for first entry then 4 etc.

        Column count increases by 2 instead of one due to grid adding two
        columns when a charge is added (odd numbered column is empty)."""
        column = 2
        try:
            for index in range(len(self.case_information.charges_list)):
                self.case_information.charges_list[index].plea = self.charges_gridLayout.itemAtPosition(3,column).widget().currentText()
                self.case_information.charges_list[index].finding = self.charges_gridLayout.itemAtPosition(4,column).widget().currentText()
                self.case_information.charges_list[index].fines_amount = self.charges_gridLayout.itemAtPosition(5,column).widget().text()
                if self.charges_gridLayout.itemAtPosition(6,column).widget().text() == "":
                    self.case_information.charges_list[index].fines_suspended = "0"
                else:
                    self.case_information.charges_list[index].fines_suspended = self.charges_gridLayout.itemAtPosition(6,column).widget().text()
                index +=1
                column +=2
        except AttributeError:
            print("Attribute error allowed to pass for lack of widget")

    @logger.catch
    def check_add_conditions(self):
        """Checks to see what conditions in the Add Conditions box are checked and then
        transfers the information from the conditions to case_information model if the
        box is checked.

        TODO:
        in future refactor this to have it loop through the different conditions so code
        doesn't need to be added each time a condition is added."""
        if self.license_suspension_checkBox.isChecked():
            self.case_information.license_suspension_details.license_suspension_ordered = (
                True
            )
        if self.community_control_checkBox.isChecked():
            self.case_information.community_control_terms.community_control_required = (
                True
            )
        if self.community_service_checkBox.isChecked():
            self.case_information.community_service_terms.community_service_ordered = (
                True
            )
        if self.other_conditions_checkBox.isChecked():
            self.case_information.other_conditions_details.other_conditions_ordered = (
                True
            )

    @logger.catch
    def calculate_costs_and_fines(self):
        self.case_information.court_costs = 0
        if self.court_costs_box.currentText() == "Yes":
            for index, charge in enumerate(self.case_information.charges_list):
                if self.case_information.court_costs == 124:
                    break
                else:
                    if charge.type == "Moving Traffic":
                        if self.case_information.court_costs < 124:
                            self.case_information.court_costs = 124
                    elif charge.type == "Criminal":
                        if self.case_information.court_costs < 114:
                            self.case_information.court_costs = 114
                    elif charge.type == "Non-moving Traffic":
                        if self.case_information.court_costs < 95:
                            self.case_information.court_costs = 95
        total_fines = 0
        try:
            for index, charge in enumerate(self.case_information.charges_list):
                if charge.fines_amount == '':
                    charge.fines_amount = 0
                total_fines = total_fines + int(charge.fines_amount)
            self.case_information.total_fines = total_fines
            total_fines_suspended = 0
            for index, charge in enumerate(self.case_information.charges_list):
                if charge.fines_suspended == '':
                    charge.fines_suspended = 0
                total_fines_suspended = total_fines_suspended + int(charge.fines_suspended)
            self.case_information.total_fines_suspended = total_fines_suspended
        except TypeError:
            print("A type error was allowed to pass - this is because of deleted charge.")

    def show_costs_and_fines(self, bool):
        """The bool is the toggle from the clicked() of the button pressed. No
        action is taken with respect to it."""
        self.update_case_information()
        message = QMessageBox()
        message.setIcon(QMessageBox.Information)
        message.setWindowTitle("Total Costs and Fines")
        message.setInformativeText("Costs: $" + str(self.case_information.court_costs) +\
            "\nFines: $" + str(self.case_information.total_fines) +\
            "\nFines Suspended: $" + str(self.case_information.total_fines_suspended) +\
            "\n\n*Does not include possible bond forfeiture or other costs \n that may be assesed as a result of prior actions in case. ")
        total_fines_and_costs = (self.case_information.court_costs +\
            self.case_information.total_fines) - self.case_information.total_fines_suspended
        message.setText("Total Costs and Fines Due By Due Date: $" + str(total_fines_and_costs))
        message.setStandardButtons(QMessageBox.Ok)
        message.exec_()

    def guilty_all_plea_and_findings(self):
        """Sets the plea and findings boxes to guilty for all charges currently
        in the charges_gridLayout."""
        for column in range(self.charges_gridLayout.columnCount()):
            try:
                if isinstance(self.charges_gridLayout.itemAtPosition(3, column).widget(), PleaComboBox):
                    self.charges_gridLayout.itemAtPosition(3,column).widget().setCurrentText("Guilty")
                    self.charges_gridLayout.itemAtPosition(4,column).widget().setCurrentText("Guilty")
                    column +=1
            except AttributeError:
                pass
        self.set_cursor_to_FineLineEdit()

    def no_contest_all_plea_and_findings(self):
        """Sets the plea box to no contest and findings boxes to guilty for all
        charges currently in the charges_gridLayout."""
        for column in range(self.charges_gridLayout.columnCount()):
            try:
                if isinstance(self.charges_gridLayout.itemAtPosition(3, column).widget(), PleaComboBox):
                    self.charges_gridLayout.itemAtPosition(3,column).widget().setCurrentText("No Contest")
                    self.charges_gridLayout.itemAtPosition(4,column).widget().setCurrentText("Guilty")
                    column +=1
            except AttributeError:
                pass
        self.set_cursor_to_FineLineEdit()

    def set_cursor_to_FineLineEdit(self):
        for column in range(self.charges_gridLayout.columnCount()):
            try:
                if isinstance(self.charges_gridLayout.itemAtPosition(5, column).widget(), FineLineEdit):
                    self.charges_gridLayout.itemAtPosition(5, column).widget().setFocus()
                    break
            except AttributeError:
                pass

    @logger.catch
    def set_fra_in_file(self, current_text):
        """Sets the FRA (proof of insurance) to true if the view indicates 'yes'
        that the FRA was shown in the complaint of file."""
        if current_text == "Yes":
            self.case_information.fra_in_file = True
            self.fra_in_court_box.setCurrentText("No")
        elif current_text == "No":
            self.case_information.fra_in_file = False
        else:
            self.case_information.fra_in_file = None

    @logger.catch
    def set_fra_in_court(self, current_text):
        """Sets the FRA (proof of insurance) to true if the view indicates 'yes'
        that the FRA was shown in court."""
        if current_text == "Yes":
            self.case_information.fra_in_court = True
        elif current_text == "No":
            self.case_information.fra_in_court = False
        else:
            self.case_information.fra_in_court = None

    def set_offense_type(self):
        key = self.statute_choice_box.currentText()
        if self.freeform_entry_checkBox.isChecked():
            return None
        query = QSqlQuery(database_statutes)
        query.prepare("SELECT * FROM charges WHERE " "statute LIKE '%' || :key || '%'")
        query.bindValue(":key", key)
        query.exec()
        while query.next():
            statute = query.value(2)
            offense_type = query.value(4)
            if statute == key:
                return offense_type

    @logger.catch
    def set_statute(self, key):
        """This method queries based on the offense and then sets the statute
        and degree based on the offense in the database.

        :key: is the string that is passed by the function each time the field
        is changed on the view. This is the offense."""
        if self.freeform_entry_checkBox.isChecked():
            return None
        query = QSqlQuery(database_offenses)
        query.prepare("SELECT * FROM charges WHERE " "offense LIKE '%' || :key || '%'")
        query.bindValue(":key", key)
        query.exec()
        while query.next():
            offense = query.value(1)
            statute = query.value(2)
            degree = query.value(3)
            if offense == key:
                self.statute_choice_box.setCurrentText(statute)
                self.degree_choice_box.setCurrentText(degree)
                break

    @logger.catch
    def set_offense(self, key):
        """This method queries based on the statute and then sets the offense
        and degree based on the statute in the database.

        :key: is the string that is passed by the function each time the field
        is changed on the view. This is the statute."""
        if self.freeform_entry_checkBox.isChecked():
            return None
        query = QSqlQuery(database_statutes)
        query.prepare("SELECT * FROM charges WHERE " "statute LIKE '%' || :key || '%'")
        query.bindValue(":key", key)
        query.exec()
        while query.next():
            offense = query.value(1)
            statute = query.value(2)
            degree = query.value(3)
            if statute == key:
                self.offense_choice_box.setCurrentText(offense)
                self.degree_choice_box.setCurrentText(degree)
                break

    @logger.catch
    def set_pay_date(self, time_to_pay_text):
        """Sets the balance of fines and costs to a future date (or today)
        depending on the selection of ability_to_pay_box. The inner function
        will move the actual date to the next tuesday per court procedure for
        show cause hearings being on Tuesday. Would need to be modified if the
        policy changed."""
        days_to_add = self.pay_date_dict[time_to_pay_text]
        future_date = date.today() + timedelta(days_to_add)
        today = date.today()

        def next_tuesday(future_date, weekday=1):
            """This function returns the number of days to add to today to set
            the payment due date out to the Tuesday after the number of days
            set in the set_pay_date function. The default of 1 for weekday is
            what sets it to a Tuesday. If it is 0 it would be Monday, 3 would
            be Wednesday, etc."""
            days_ahead = weekday - future_date.weekday()
            if days_ahead <= 0:  # Target day already happened this week
                days_ahead += 7
            return future_date + timedelta(days_ahead)

        future_date = next_tuesday(future_date, 1)
        total_days_to_add = (future_date - today).days
        self.balance_due_date.setDate(QDate.currentDate().addDays(total_days_to_add))


if __name__ == "__main__":
    print("MMD ran directly")
else:
    print("MMD ran when imported")
    database_offenses, database_statutes = create_database_connections()
```