Klient bazy danych

Projekt z Języków symbolicznych

Anna Krasowska GL 31

Temat projektu

Projektem jest aplikacja okienkowa do zarządzania danymi przedstawionymi w na wzór bazy danych, zatem istnieją: * Tabele * Kolumny * Wiersze

Można również wykonywać operacje na powyższych obiektach, czyli: * **Tabele** - dodawanie, usuwanie * **Kolumny** - dodawanie * **Wiersze** - dodawanie, usuwanie, filtrowanie (wyszukiwanie)

Stan bazy jest odczytywany z pliku oraz przy wyjściu zapisywany do pliku JSON.

Wykorzystana biblioteka do rysowania okienek to DearPyGui (https://github.com/hoffstadt/DearPyGui).

Funkcjonalność

Sekcja tabel:

- Przycisk Dodaj tabele, który wyświetla okienko pozwalające na dodanie tabeli wraz z kolumnami (lub bez)
- Tabela tabel z kolumnami Nazwa, Wiersze (ilość wierszy w tabeli) i Akcja (możliwa akcja, czyli Usuń)
- Naciśnięcie na nazwę powoduje podświetlenie pola i wybranie tabeli

Okienko dodania nowej tabeli

- Pola tekstowe z nazwą tabeli i kolumny do dodania
- Przycisk typu radio z typem dodawanej kolumny
- Tabela z aktualnie dodanymi kolumnami
- Możliwość powrotu przyciskiem Wróć, co nie powoduje żadnych zmian

Sekcja dodania kolumny:

Po wybraniu tabeli jest możliwość dodania kolejnej kolumny, wprowadzając nazwę i wybierając typ elementu typu radio

Sekcja wierszy:

- Nazwa tabeli
- Tabela z wierszami wybranej tabeli z kolumnami Wiersz (numer wiersza), [kolumny tabeli], Akcje (Usuń przy istniejących, dodaj przy wierszu wprowadzania nowego wiersza)
- Wiersz dodania nowego wiersza - jako placeholder/hint jest wyświetlany typ danej kolumny z możliwością szybkiego dodania nowego wiersza przyciskiem Dodaj
- Pole tekstowe do filtrowania wierszy, razem z przyciskiem Szukaj (inicjuje akcję filtrowania) i Reset (wraca do oryginalnego stanu). W polu tekstowym jest możliwość wpisania poprawnego wyrażenia lambda, które zostanie wykonane na wierszach i wynik zostanie wyświetlony zamiast aktualnych danych

Odczyt i zapis do pliku .json

Na start programu jest wyszukiwany plik `db.json`, jeżeli jest znaleziony to dane z niego są wczytywane przez aplikację. Po zakończeniu działania programu (zamknięcie okna) jest wykonywany zapis do pliku `db.json`.

Pakiety

application

Klasa TableService ➔ link

Służy jako interfejs do operacji na logice programu, a zatem wykonywanie operacji dodawania kolumn, dodawania wierszy.

infrastructure

Klasa Decoder ➔ link

Odczytuje plik .json korzystając z metody `object_hook` używanej przez klasę bazową `JSONDecoder`. Korzysta z dodatkowych pól, by móc odróżnić różne typy od siebie.

Klasa Encoder → link

Zapisuje dane do pliku .json korzystając z metody `default` używanej przez klasę bazową `JSONEncoder`. Tworzy wpisy pomocnicze celem późniejszego łatwiejszego odczytu.

lib

Klasa BaseObservable → link

Pozwala na łatwiejsze bindowanie danych między modelami, a widokami. Przetrzymuje słownik z funkcjami, które potem klasa dziedzicząca może wykonać przy pomocy metody `_doCallbacks`.

Klasa Observable → link

Podobnie jak klasa `BaseObservable`, tylko bardziej jako wrapper na pojedyncze dane.

Metaklasa SingletonMeta → link

Metaklasa pozwalająca na łatwą implementację wzorca projektowego Singleton. Klasy dziedziczące mogą mieć tylko jedną instancję.

model

Klasa Repository → link

Singleton przechowujący dane o tabelach. Posiada podstawowe metody do dodawania, usuwania oraz wyszukiwania tabel. Dziedziczy również po `BaseObservable` celem łatwiejszego zbindowania modelu z UI.

Klasa Row → link

Model wiersza. Dane przechowywane w postaci słownika, metody pozwalają na dodanie wartości i pozyskanie słownika.

Klasa Table → link

Model tabeli. Metody umożliwiają dodawanie, usuwanie wierszy, dodawanie kolumn. Operacje dodawania, usuwania są walidowane i w tej klasie są zdefiniowane walidatory dla tabeli. Dziedziczy również po `BaseObservable`.

model/column

Klasa Column → link

Podstawowa klasa po której dziedziczą wszystkie inne klasy `...Column`. Posiada metodę do walidacji nazwy kolumny, udostępnia `property` z nazwą. Posiada dwie niezaimplementowane metody `cast` i `type`, które powinny być przesłonięte przez klasy dziedziczące.

Klasa FloatColumn → link

Klasa dziedzicząca po `Column`. `type` zwraca `float`, a `cast` rzutuje typ na `float`. W razie błędnego rzutowania jest wyrzucany wyjątek ColumnTypeError.

Klasa IntegerColumn → link

Klasa dziedzicząca po `Column`. `type` zwraca `int`, a `cast` rzutuje typ na `int`. W razie błędnego rzutowania jest wyrzucany wyjątek ColumnTypeError.

Klasa TextColumn → link

Klasa dziedzicząca po `Column`. `type` zwraca `str`, a `cast` rzutuje typ na `str`. W razie błędnego rzutowania jest wyrzucany wyjątek ColumnTypeError.

Klasa ColumnTypeError → link

Własny wyjątek zwracany, kiedy występuje problem z wprowadzanym typem danych, a typem danych kolumny.

ui/root

Klasa RootView → link

Definiuje wysokość i szerokość okienka, tworzy `viewport` aplikacji. Posiada metody do wyśrodkowywania elementu.

Klasa RootHandler ➔ link

Inicjalizuje `RootView` oraz wczytuje dane do `Repository` z pliku, jeżeli istnieją. Definiuje również handler wywoływany w momencie zamknięcia aplikacji.

ui/table

Klasa TableView ➔ link

Główna klasa typu `View`, która rysuje lub wywołuje klasy rysujące elementy w interfejsie. Zawiera większość logiki związanej z UI, oprócz funkcji `callback`, które są przekazywane z `TableHandler`.

Klasa TableHandler ➔ link

Przechowuje wszystkie funkcje do obsługi wydarzeń w zakresie aplikacji. W tym miejscu są one również przekazywane do widoku `TableView` razem z jego inicjalizacją.

ui/widgets

W pakiecie znajdują się klasy tworzące podstawowe elementy.

Klasa AddTableModal ➔ link

Tworzy elementy w interfejsie graficznym odpowiadające za okienko `AddTable`. Udostępnia `property form`, które zczytuje dane z elementów interfejsu.

Klasa ConfirmationModal ➔ link

Tworzy elementy w interfejsie graficznym odpowiadające za `ConfirmationModal`, czyli potwierdzenie.

ErrorPopup ➔ link

Tworzy elementy w interfejsie graficznym odpowiadające za `ErrorPopup`, czyli popup błędu. Element jest tworzony obok wskazanego w konstruktorze elementu.

TablesTable ➔ link

Tworzy elementy w interfejsie graficznym odpowiadające za `TablesTable`, czyli tabelę z obiektami `Table`.

MainApplication

Główny punkt wejścia programu.

Funkcja run ➔ link

Wywołuje podstawowe funkcje biblioteki dearpygui.

Funkcja exit ➔ link

Niszczy kontekst elementów biblioteki dearpygui.

Testy ➔ link

Po każdym teście jest czyszczony singleton `Repository` w metodzie `tearDown`.

test_shouldAddValidTableAndColumn

Sprawdza dodawanie tabel wraz z kolumnami.

test_shouldAddValidRow

Sprawdza dodawanie wierszy do istniejących już tabeli.

test_shouldNotAddInvalidRowWhenTypeIsIncorrect

Sprawdza negatywny przypadek dodawania wierszy do istniejących już tabeli.

test_shouldNotAddTableWithEmptyName

Sprawdza negatywny przypadek dodawania nowych tabeli z niepoprawną nazwą.

test_shouldNotAddColumnWithEmptyField

Sprawdza negatywny przypadek dodawania tabeli z niepoprawnymi kolumnami.

test_shouldQueryRows

Sprawdza czy filtrowanie poprzez przekazanie funkcji `lambda` działa.

Konstrukcje

Wyrażenia lambda

Pobranie wartości jako callback → link

Pobranie nazwy aktualnie wybranej tabeli → link

Test funkcjonalności filtrowania → link

List & dict comprehension

Oba sposoby są używane dość powszechnie w projekcie, poniżej 3 przykłady:

Rzutowanie wartości w słowniku → link

Czytanie wartości z elementów UI → link

Filtrowanie wartości na podstawie przekazanej funkcji → link

Klasy

Projekt został wykonany w większości w paradygmacie OOP. W projekcie znajduje się kilka klas po których dziedziczą inne klasy, z czego jedna to metaklasa. Występuje również podział na klasy od logiki i interfejsu. Przykłady poniżej:

Klasa bazowa Column → link

Klasa dziedzicząca FloatColumn → link

Klasa dziedzicząca IntegerColumn → link

Klasa dziedzicząca TextColumn → link

Wyjątki

Jest zdefiniowana jedna własna klasa wyjątku. Przy walidacji danych powszechne w projekcie jest łapanie i rzucanie wyjątków.

Klasa wyjątku → link

Łapanie wyjątku → link

Moduły

Za wyjątkiem `MainApplication.py` wszystkie pliki zawierają po jednej klasie i projekt jest uporządkowany według pakietów.

Metaklasy

Singleton → link

Dekoratory

Dekoratory są używane w wielu miejscach ze względu na użycie dekoratora `@property`.

Rozszerzenie `@property` przez ustawienie settera ➔ link

`@abc.abstractmethod` ➔ link