Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

Отчет
по лабораторной работе N2

Выполнил: Жолнерчик И. А.
Группа 121701

Проверил: Бутрин С. В.

Минск 2023

Условие задания:

Вариант 7

Значения атрибутов:
Состав: основной/запасной/ “n/a”.
Позиция: в зависимости от вида спорта.
Титулы: некоторое число.
Разряд: 1-й юношеский, 2-й разряд, 3й-разряд, кмс, мастер спорта.

ФИО спортсмена	Состав (если имеется)	Позиция	титулы	Вид спорта	Разряд

Условия поиска и удаления:

- по фио или виду спорта;
- по количеству завоеваний титула (при задании должны быть указаны верхний и нижний предел);
- по фио или разряду

Замечание: Список видов спорта и список видов разрядов в диалоге поиска, собирается системой и выводится в выпадающий список;

Работоспособность программы осуществляется через следующие файлы:

- main.py
- myscreen.py
- dialog_windows.py
- xml_generator.py

Файл main.py:
Сначала в строках 1-4 импортируются необходимые модули и классы из различных файлов приложения.

Затем создается класс PassMVC, который является подклассом MDApp из KivyMD. В методе init создается объект MDDataTable, который представляет таблицу с данными. Он содержит различные параметры, такие как размер, количество строк, цвет фона и содержимое столбцов.

Затем создаются объекты MyScreenModel и MyScreenController, которые представляют модель и контроллер соответственно. Они используются для связи таблицы данных и пользовательского интерфейса.

Метод build используется для создания графического интерфейса, устанавливает размер окна и возвращает экран, который был создан в контроллере.

В конце кода создается объект PassMVC и запускается его метод run(), что запускает приложение.

```python
from kivymd.app import MDApp
from kivymd.uix.datatables import MDDataTable
from Controller.myscreen import MyScreenController
from Model.myscreen import MyScreenModel
from kivy.core.window import Window
from kivy.metrics import dp


class PassMVC(MDApp):
    def __init__(self):
        super().__init__()
        self.table = MDDataTable(
            pos_hint={'center_x': 0.5, 'center_y': 0.55},
            size_hint=(0.9, 0.9),
            use_pagination=True,
            elevation=2,
            rows_num=7,
            pagination_menu_height=240,
            background_color=(0, 1, 0, .10),
            column_data=[
                ("[color=#123487]FIO[/color]", dp(40)),
                ("[color=#123487]Line-up (if available)[/color]", dp(40)),
                ("[color=#123487]Position[/color]", dp(20)),
                ("[color=#123487]Titles[/color]", dp(20)),
                ("[color=#123487]Sport type[/color]", dp(30)),
                ("[color=#123487]Rank[/color]", dp(25)),
            ],
        )
        self.model = MyScreenModel(table=self.table)
        self.controller = MyScreenController(self.model)

    def build(self):
        Window.size = (1920, 1080)
        return self.controller.get_screen()


PassMVC().run()
```

Файл myscreen.py:

В этом коде определяется класс MyScreenController, который является контроллером в архитектуре MVC (Model-View-Controller).

В конструкторе создаются объекты модели (model) и представления (view), которые передаются в контроллер. Кроме того, создается список _observers,

который используется для уведомления объектов-наблюдателей об изменениях в модели.

Методы add_observer, remove_observer и notify_observers используются для управления списком наблюдателей и оповещения их об изменениях в модели.

Методы refresh, input_sportsman, dialog, get_degrees, filter_sportsmans, delete_sportsmans, upload_from_file, save_in_file, open_dialog и close_dialog представляют различные операции, которые может выполнить контроллер. Они используют методы модели, чтобы выполнить соответствующие действия, и вызывают методы представления, чтобы обновить интерфейс или показать диалоговое окно.

Таким образом, контроллер связывает модель и представление, обеспечивает их взаимодействие и управляет пользовательским интерфейсом.

Файл xml_generator.py:

В этом коде определяется класс XMLGenerator, который содержит статический метод generate_xml_files для генерации XML-файлов со случайными данными о спортсменах.

Метод generate_xml_files принимает два аргумента: files_count - количество файлов, которые необходимо создать, и sportsman_count - количество спортсменов, которые должны быть в каждом файле.

Для каждого файла метод создает словарь data_dict, который содержит случайные данные о спортсменах, такие как имя, позиция, звание и т.д. Для генерации случайных данных используются функции из библиотеки numpy.random и модуля names.

Затем метод создает объект XmlWriter, который использует DOM-парсер для генерации XML-файла. Для каждого спортсмена метод вызывает метод create_sportsman объекта XmlWriter, который добавляет запись о спортсмене в XML-файл.

После завершения цикла создания записей для спортсменов метод create_xml_file объекта XmlWriter вызывается для создания XML-файла.

Наконец, в функции main создается экземпляр класса XMLGenerator и вызывается метод generate_xml_files для создания 10 файлов, каждый из которых содержит 50 записей о спортсменах.

```python
class XMLGenerator:
    def __init__(self):
        pass

    @staticmethod
    def generate_xml_files(files_count: int, sportsman_count: int) -> None:
        """
        Generate N files with each of which has one number of players
        :param files_count: amount of files
        :param sportsman_count: amount of student in each file
        :return: None
        """
        for i in range(files_count):
            path = f"../xml/{str(i)}.xml"
            data_dict = {}
            line_up_values = np.array(['main', 'reserve', 'n/a'])
            sick_values = np.array(['goalkeeper', 'forward', 'defender', 'midfielder'])
            rank_values = np.array(['1st', '2nd', '3d', 'CMS', 'master'])
            sport_type_values = np.array(['football'])
            with open(path, 'w') as file:
                dom_writer = XmlWriter(path)
                for _ in range(sportsman_count):
                    # dictionary filling
                    data_dict["name"] = names.get_full_name()
                    data_dict["line_up"] = choice(line_up_values)
                    data_dict["position"] = choice(sick_values)
                    data_dict["titles"] = str(randint(0, 150))
                    data_dict["sport_type"] = choice(sport_type_values)
                    data_dict["rank"] = choice(rank_values)
                    # adding record for each sportsman
                    dom_writer.create_sportsman(data_dict)
                # creating xml file using dom parser
                dom_writer.create_xml_file()


def main():
    # creating 10 files with 50 sportsmans in each other
    XMLGenerator.generate_xml_files(files_count=10, sportsman_count=50)


if __name__ == "__main__":
    main()
```