|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ ИМЕНИ К.Г. РАЗУМОВСКОГО (ПЕРВЫЙ КАЗАЧИЙ УНИВЕРСИТЕТ)»**  **(ФГБОУ ВО «МГУТУ ИМ. К.Г. РАЗУМОВСКОГО (ПКУ)»)** |
|  |
| **УНИВЕРСИТЕТСКИЙ КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ** |

|  |
| --- |
| **КУРСОВОЙ ПРОЕКТ** |
|  |
| по междисциплинарному курсу МДК.02.02. Технология разработки и защиты баз данных |
|  |
| на тему: Разработка программы-генератора билетов к экзамену. |
|  |
| студента группы ПКС-305  специальности 09.02.03 Программирование в компьютерных системах |
| Чеботарёва Степана Михайловича |

|  |  |  |
| --- | --- | --- |
| Студент |  | С.М. Чеботарёв |
| Руководитель курсового проекта |  | Е.А. Ларионова |
| Председатель ПЦК специальности 09.02.03 Программирование в компьютерных системах |  | А.И. Глускер |

|  |  |  |
| --- | --- | --- |
| Дата защиты «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. | | |
| Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Заведующий отделением № 3 |  | Е.Л. Гусева |

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc40026496)

[1 ОСНОВНАЯ ЧАСТЬ 6](#_Toc40026497)

[1.1 Назначение и цели создания системы 6](#_Toc40026498)

[1.2 Обзор и анализ предметной области 6](#_Toc40026499)

[1.3 Жизненный цикл базы данных 7](#_Toc40026500)

[1.4 Выбор и характеристика СУБД 8](#_Toc40026501)

[1.5 Выбор и характеристика среды разработки 8](#_Toc40026502)

[1.6 Выбор и характеристика языка программирования 9](#_Toc40026503)

[2 СПЕЦИАЛЬНАЯ ЧАСТЬ 10](#_Toc40026504)

[2.1 Постановка задачи 10](#_Toc40026505)

[2.2 Архитектура информационной системы 11](#_Toc40026506)

[2.2.1 Основные этапы проектирования приложения 11](#_Toc40026507)

[2.2.2 Описание паттерна MVVM 11](#_Toc40026508)

[2.2.3 Архитектура клиент-сервер 12](#_Toc40026509)

[2.2.4 Навигационная схема приложения 13](#_Toc40026510)

[2.2.5 Архитектура классов приложения 13](#_Toc40026511)

[2.3 Логическая модель базы данных 16](#_Toc40026512)

[2.4 Нормализация таблиц 16](#_Toc40026513)

[2.5 Описание таблиц 18](#_Toc40026514)

[2.6 Разработка приложения 22](#_Toc40026515)

[2.6.1 Диаграмма вариантов использования 22](#_Toc40026516)

[2.6.2 Решение главной задачи проекта 22](#_Toc40026517)

[2.6.3 Тестирование приложения 24](#_Toc40026518)

[2.6.4 Защита информационной системы 28](#_Toc40026519)

[2.6.5 Скриншоты интерфейса 29](#_Toc40026520)

[2.7 Инструкция пользователю 34](#_Toc40026521)

[2.7.1 Общие сведения об информационной системе 35](#_Toc40026522)

[2.7.2 Требования к техническим средствам 35](#_Toc40026523)

[2.7.3 Требования к программным средствам 35](#_Toc40026524)

[2.7.4 Настройка информационной системы 35](#_Toc40026525)

[2.7.5 Формы ввода 35](#_Toc40026526)

[2.7.6 Отчеты 36](#_Toc40026527)

[ЗАКЛЮЧЕНИЕ 37](#_Toc40026528)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 38](#_Toc40026529)

[ПРИЛОЖЕНИЕ А SQL скрипты на создание и заполнение базы данных, на создание учетных записей пользователей, на назначение привилегий 39](#_Toc40026530)

[ПРИЛОЖЕНИЕ Б Код программы 44](#_Toc40026531)

# ВВЕДЕНИЕ

Темой проекта является упрощение проведения комплексных экзаменов. Комплексный экзамен – это проверка знаний по нескольким смежным дисциплинам одновременно. Его цель – определить, насколько целостно учащиеся воспринимают близкие науки. На данный момент комплексные экзамены являются повсеместной практикой учебных учреждений. Как правило комплексный экзамен проводится по 2-м близким друг к другу учебных предметов. Есть определенные требования к комплексному экзамену. Все включаемые в него дисциплины обязательно взаимосвязаны и относятся к общей области знаний, а также их изучение завершается параллельно – в одном семестре. Материалы, которые выносятся на экзамен, охватывают наиболее актуальные темы и разделы включаемых в него учебных дисциплин.

Для проведения комплексного экзамена необходимы экзаменационные билеты, составление которых вручную занимает уйму времени, думая об этой проблеме мне стал интересен способ её решения путем создания программы, которая могла бы автоматически генерировать экзаменационные билеты для проведения комплексных экзаменов.

Основные задачи, которые необходимо выполнить для того, чтобы разработать данное приложение:

* подробное изучение функциональных возможностей tkinter,
* освоение тонкостей проектирования баз данных,
* изучение Peewee,
* разработка функций для работы с базой данных,
* создание функций для работы с данными внутри приложения,
* проектирование интерфейса программы,
* написание функций для корректной генерации и сохранения данных,
* проектирование архитектуры и логики приложения.

Используемые в разработке методы:

* объектно-ориентированное программирование,
* моделирование.

Обоснование структуры курсового проекта:

1. введение: в данном разделе описываются цели и задачи проекта, а также его актуальность, так как изначально необходимо определить, что предстоит изучать и делать в проекте,
2. основная часть: этот раздел включает в себя анализ предметной области, для которой разрабатывается данное приложение, назначения и цели разработки, а также жизненный цикл базы данных,
3. специальная часть: в данном разделе описаны архитектурные решения, которые применяются в разработке, а также процесс разработки программного обеспечения и базы данных,
4. заключение: вывод о проделанной работе и полученных навыках за время разработки приложения,
5. список используемых источников: этот раздел содержит в себе перечень ресурсов, которые были использованы для получения необходимой информации во время разработки.

# 1 ОСНОВНАЯ ЧАСТЬ

* 1. Назначение и цели создания системы

Приложение создается для организаторов комплексных экзаменов. База данных приложения содержит информацию о преподавателях, учебных предметов, экзаменационных вопросов, сложности вопросов, администрирующих пользователях.

Цель: создать приложение позволяющее генерировать экзаменационные билеты для комплексного экзамена по 2-ум предметам с возможностью выбора от 3 до 5 вопросов и их сложности.

Назначение системы: обеспечить простое составление экзаменационных билетов для комплексного экзамена.

Базовые сущности этой предметной области: преподаватели, учебные предметы, экзаменационные вопросы, сложности вопросов, пользователи.

В системе должны существовать один вид взаимодействия: аутентифицированный администратор.

* 1. Обзор и анализ предметной области

Предметной областью разработки является комплексный экзамен.

Список основных сущностей:

* преподаватели – это профессора, обучающие студентов, которые могут преподавать несколько предметов. Атрибуты преподавателей: идентификатор, фамилия, имя, отчество,
* пользователь – это лицо, наделенное правом генерировать экзаменационные билеты. Атрибуты пользователей: логин, пароль,
* учебные предметы – конкретные предметы, по которым проводится комплексный экзамен. Атрибуты учебных предметов: название, идентификатор профессора.
* сложности – сложности экзаменационных вопросов. Атрибуты сложностей: сложность,
* экзаменационные вопросы – экзаменационные вопросы учебных предметов. Атрибуты турниров: текст вопроса, сложность, учебный предмет,
  1. Жизненный цикл базы данных

Жизненный цикл – это процесс проектирования, реализации и управления базой данных. Он состоит из:

1. Анализа предметной области, который включает в себя планирование разработки (15.02.2020 – 25.02.2020), определение требований (15.02.2020 – 25.02.2020), сбор и анализ требований (15.02.2020 – 25.02.2020),
2. Проектирования базы данных, которое состоит из создания концептуальной модели (26.02.2020), логической модели (26.02.2020), физической модели (27.02.2020),
3. Реализации, в процессе которой пишутся скрипты на создание базы данных, связей сущностей в базе данных, а также разрабатывается интерфейс приложения и его функции (01.03.2020 – 05.05.2020).
   1. Выбор и характеристика СУБД

Для разработки базы данных была выбрана среда PostgreSQL pgAdmin 4, так как она является удобной для использования и не требует долгого освоения тонкостей для создания полноценной базы данных под программу, а также с ней можно работать, используя Peewee. Эта СУБД использует язык SQL (PL/pgSQL) для написания необходимых сущностей и заполнения их данными.

При сравнении PostgreSQL с MySQL можно выделить несколько главных различий:

* первый предназначен для работы с любой платформой, а второй для работы с PHP,
* синтаксис ,
* в PostgreSQL документация гораздо лучше, чем в MySQL.

Весомыми плюсами PostgreSQL являются простота понимания принципов работы, синтаксиса, а также возможность создавать объекты баз данных либо с помощью скрипта, либо вручную, с помощью конструктора.

* 1. Выбор и характеристика среды разработки

Для разработки приложения была выбрана среда программирования Microsoft Visual Studio Code, потому что эта среда разработки обеспечивает возможность удобной работы практически со всеми языками.

Microsoft Visual Studio Code – продукт компании Microsoft, включающий в себя интегрированную среду разработки программного обеспечения, а также ряд других инструментальных средств.

Некоторые преимущества:

* удобная отладка и поиск ошибок,
* легковесность,
* огромное количество удобных плагинов,
* подробная документация, располагающаяся на сайте компании Microsoft,
* интеллектуальность системы, предупреждающая об ошибках и помогающая упростить и улучшить код. Данную среду разработки можно сравнить с Sublime 3. Она является менее удобной, так как не представляет такого обширного количества средств для отладки и тестирования.
  1. Выбор и характеристика языка программирования

Для написания кода приложения был выбран язык Python3.8, так как он является предпочтительным для меня, а также с целью повышения навыков его использования в различных целях.

Python - это интерпретируемый язык программирования общего назначения высокого уровня. Некоторые преимущества:

* поддерживается ООП,
* удобный синтаксис,
* присутствие библиотеки Peewee для более удобной работы с коллекциями и базами данных.

В сравнении с языком C#, Python является более удобным, лаконичным и кроссплатформенным.

# 2 СПЕЦИАЛЬНАЯ ЧАСТЬ

2.1 Постановка задачи

Главная задача проекта – спроектировать средствами PostgreSQL базу данных и создать приложение средствами Microsoft Visual Studio Code и pgAdmin 4, позволяющее генерировать экзаменационные билеты для комплексных экхаменов.

Предусмотреть:

* размещение 2-х билетов на странице
* определять количество билетов на основе количества студентов
* размещение вопросов с 2-х разных учебных предметов
* отсутствие повторяющихся вопросов
* задавать вручную количество вопросов в билете от 3 до 5
* задавать вручную сложность вопросов в билете

В системе должны существовать один вид пользователей: администратор.

Администратор может выполнять следующие действия:

* генерировать экзаменационные билеты.

2.2 Архитектура информационной системы

2.2.1 Основные этапы проектирования приложения

Первым пунктом разработки являлось создание первичной версии базы данных для последующей доработки.

После успешного завершения проектирования базы данных нужно было приступать к разработке интерфейса.

Первой была создана страница авторизации.

После этого было разработан линейный интерфейс программы, позволяющий администратору генерировать экзаменационные билеты.

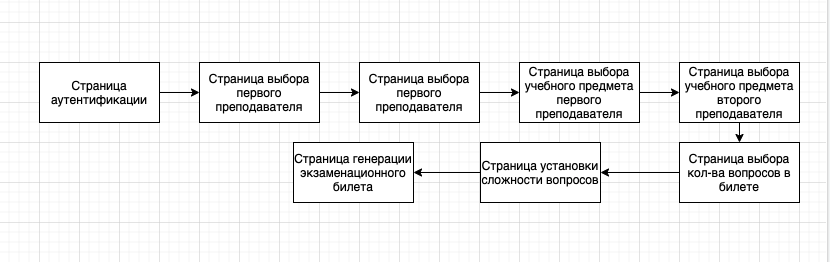
2.2.2 Навигационная схема приложения

Рисунок 3 – навигационная схема приложения

2.2.3 Архитектура классов приложения

Для создания оптимальной архитектуры в среде Microsoft Visual Studio Code я создал несколько модулей – модуль работы с базой данных ( приватный и публичный ), модуль генерации визуальной составляющей экзаменационных билетов, модуль приложения включающий в себя визуальную часть совмещённую с бизнес-логикой.

Модуль ui отвечает за следующие функции:

* обработка событий, вызванные пользователем,
* проверка вводимых пользователем данных,
* отображение полученных данных их базы,
* аутентификация пользователя.

Модуль generator отвечает за следующие функции:

* генерация визуальной части экзаменационного билета,
* генерация готового к печати A4 с двумя экзаменационными билетами.

Модули database, \_database отвечают за следующие функции:

* соединение с базой данных,
* получение данных из базы,
* добавление данных в базу,
* соединение данных таблиц,
* формирование экземпляров, с которыми могут корректно работать модули ui, generator,
* формирование экземпляров, с которыми может корректно работать база данных,
* проверки существования записей с указанными параметрами.

Модуль utils отвечает за вспомогательные функции, например:

* хранит в себе некоторые типы,
* хранит в себе набор функций для работы с конфигурационным файлом ( json ).

Исходя из архитектуры взаимодействия модулей можно составить модель, которая описывает взаимосвязи в приложении (рисунок 4).

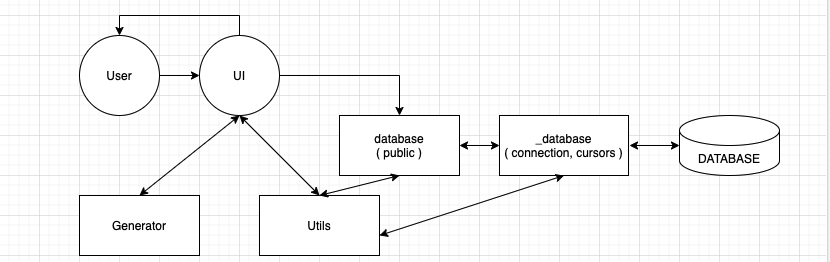


Рисунок 4 – модель архитектуры приложения

2.3 Логическая модель базы данных

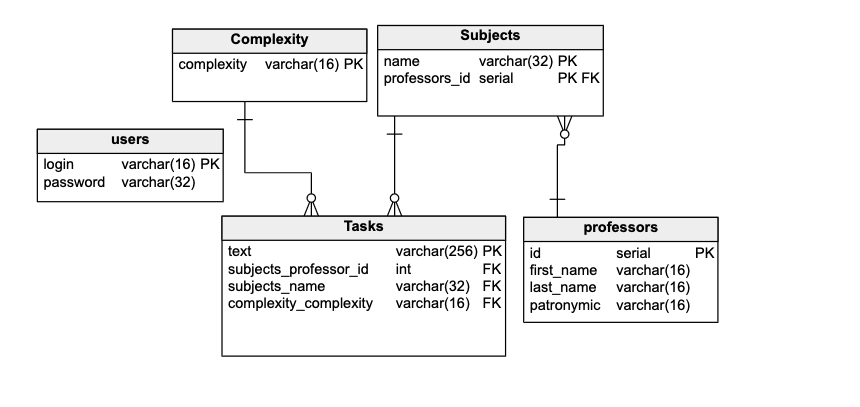
Логическая модель базы данных, используемой в приложении (Рисунок 5). 

Рисунок 5 – логическая модель базы данных

2.4 Нормализация таблиц

Первая нормальная форма – все атрибуты атомарны. Это можно продемонстрировать на примере таблицы Professors, атрибутами которой являются:

* id,
* first\_name,
* last\_name,
* patronymic,

Соответствие первой нормальной форме можно увидеть и в других таблицах спроектированной базы данных.

Вторая нормальная форма – выполнены условия первой нормальной формы, а также каждый не ключевой атрибут неприводимо зависит от первичного ключа. Неприводимость – отсутствие атрибутов, от которых можно вывести функциональную зависимость.

Пример сущностей во второй нормальной форме (Рисунок 6).

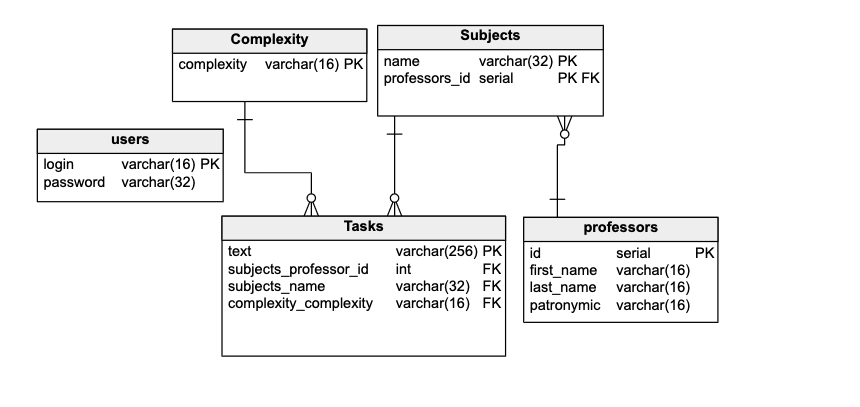


Рисунок 6 – пример второй нормальной формы

Третья нормальная форма – соблюдены условия второй нормальной формы, а также каждый не ключевой атрибут нетранзитивно зависит от первичного ключа.

База данных приведена к третьей нормальной форме.

2.5 Описание таблиц

В этом разделе описаны названия таблиц, их атрибуты, свойства и типы атрибутов, первичные и внешние ключи.

Определения:

* FK – внешний ключ,
* PK – первичный ключ,
* NULL – разрешены нулевые значения.

Таблица 1 – Professors

|  |  |  |
| --- | --- | --- |
| Название | Тип | Дополнительно |
| Id | serial | PK |
| first\_name | varchar(16) |  |
| last\_name | varchar(16) |  |
| patronymic | varchar(16) |  |

Таблица 2 – Subjects

|  |  |  |
| --- | --- | --- |
| Название | Тип | Дополнительно |
| name | varchar(32) | PK |
| professors\_id | serial | FK |

Таблица 3 – Complexity

|  |  |  |
| --- | --- | --- |
| Название | Тип | Дополнительно |
| complexity | varchar(16) | PK |

Таблица 4 – Tasks

|  |  |  |
| --- | --- | --- |
| Название | Тип | Дополнительно |
| text | varchar(256) | PK |
| subjects\_professor\_id | int | FK |
| subjects\_name | varchar(32) | FK |
| complexity\_complexity | varchar(16) | FK |

Таблица 5 – Users

|  |  |  |
| --- | --- | --- |
| Название | Тип | Дополнительно |
| login | varchar(16) | PK |
| password | varchar(32) |  |

2.6 Разработка приложения

2.6.1 Диаграмма вариантов использования

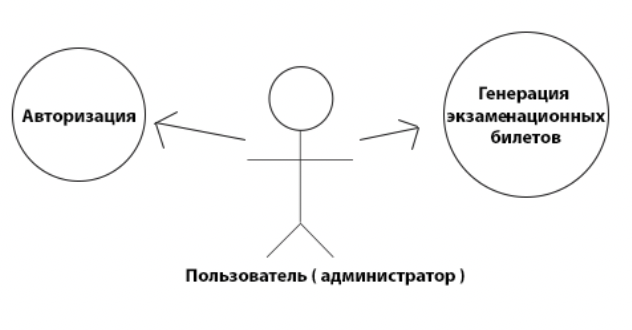


Рисунок 7 – диаграмма вариантов использования

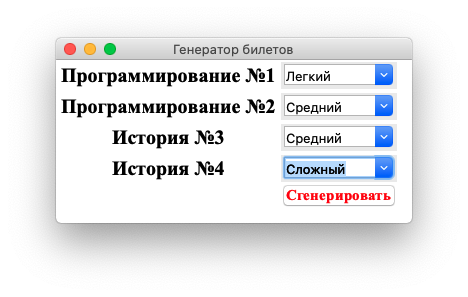
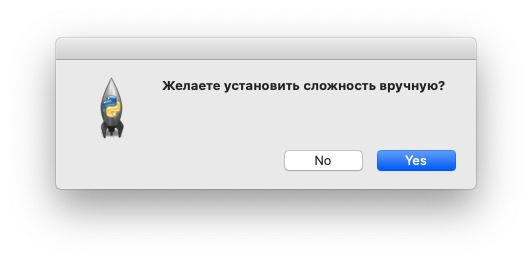
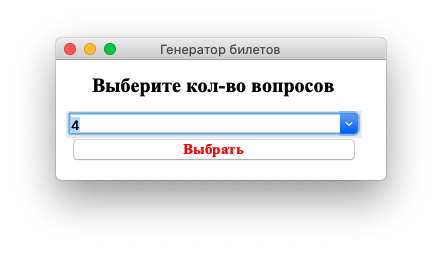
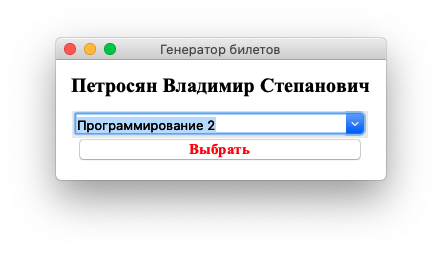
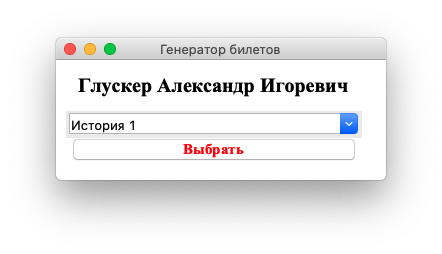
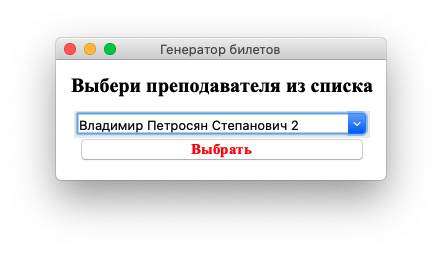
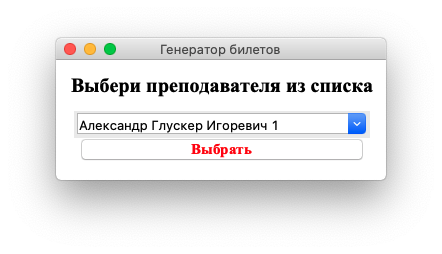
2.6.2 Решение главной задачи проекта

Главная задача проекта – генерация экзаменационных билетов для комплексного экзамена.

Необходимо:

* выбрать двух преподавателей,
* выбрать два учебных предмета из списка преподаваемых профессорами,
* выбрать количество вопросов в билетах,
* выбрать сложность каждого вопроса в экзаменационных билетах, либо запустить автоматический выбор сложности вопросов,
* произвести генерацию и визуализацию экзаменационных билетов.

Обеспечение возможности генерации экзаменационных билетов для комплексного экзамена (Рисунки 8-14).

Рисунки 8-14 – возможность генерации экзаменационных билетов

2.6.3 Тестирование приложения

Тестирование не производится.

2.6.4 Защита информационной системы

В разрабатываемом приложении предусмотрена защита всех форм ввода от неправильной и некорректной информации. При попытке ввода неверных данных пользователю будет выведено сообщение об ошибке. При вводе корректных данных все будет успешно использовано.

Защищена страница авторизации

2.6.5 Скриншоты интерфейса

Интерфейс представлен в линейном виде для упрощения процесса координации.

Ознакомиться с интерфейсом можно на рисунках ( 8-14, в подпункте 2.6.2)

2.7 Инструкция пользователю

Для начала использования приложения следует пройти авторизацию, При вводе некорректных данных будет выведено сообщение об ошибке.

Данные для входа администратора: login – @dmin, password - @dmin

Требования к паролю – минимум один символ из набора “!@#$%^”.

Требования к логину – уникальность.

2.7.1 Общие сведения об информационной системе

Информационная система разработана для использования ответственными за проведение комплексных экзаменов должностными лицами образовательных учреждений.

2.7.2 Требования к техническим средствам

Необходимо иметь локальный компьютер с установленной PostgreSQL, pgAdmin 4 в котором необходимо создать базу данных приложения используя скрипт.

2.7.3 Требования к программным средствам

Для использования программы на локальном компьютере пользователя должны быть установлены PostgreSQL 12, pgAdmin 4, Python 3.8, а так же сама программа и база данных, приложенная к ней.

2.7.4 Настройка информационной системы

Информационная система настроена для интуитивного взаимодействия с пользователем. Интерфейс будет дорабатываться в соответствии с критикой и в целях достижения максимального удобства.

2.7.5 Формы ввода

В приложении присутствуют такие формы ввода как:

* авторизация. Содержит поля логин и пароль,
* выбор преподавателя. Содержит ФИО и идентификатор преподавателя – выбираются из списка предложенных,
* выбор учебного предмета. Содержит название учебного предмета и идентификатор преподавателя, выбираются из списка предложенных,
* выбор количества вопросов в экзаменационном билете. Содержит количество вопросов, выбирается из списка предложенных,
* выбор сложности вопросов. Содержит сложность, выбирается из списка предложенных,
* дополнительные поля ввода. Содержат дополнительную информацию, все заполняются в соответствии сообщения программы, string

2.7.6 Отчеты

# ЗАКЛЮЧЕНИЕ

Была проведена работа в ходе которой было спроектировано и создано приложение для генерации экзаменационных билетов комплексного экзамента.

Разработанная система может быть усовершенствована с целью охвата большего количества полезных функций. Были выполнены следующие цели проекта:

* улучшение навыков проектирования баз данных и архитектуры приложений,
* создание базы данных для системы,
* написание программы-генератора экзаменационных билетов комплексных экзаменов.

Подводя итоги необходимо сказать, что приложение и база данных были успешно выполнены, протестированы и готовы к работе в течение 2-х месяцев. Во время выполнения проекта возникало множество проблем, которые были успешно решены. В частности проблемы возникали при написании функций для взаимодействия с базой данных. Решение этих проблем помогло узнать много нового о работе Peewee. В будущем возможны доработки базы данных и приложения для более простого взаимодействия с пользователем.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация по языку Python [Электронный ресурс]. – 2020 – URL: https://docs.python.org/3/ (дата обращения 15.12.2020),
2. Документация по ORM Peewee [Электронный ресурс]. – 2020 – URL: http://docs.peewee-orm.com/en/latest/ (дата обращения 15.11.2020),
3. Онлайн инструмент для визуального проектирования баз данных [Электронный ресурс]. – 2020 – URL: https://my.vertabelo.com/model/yfelvsbCdjOOpQE9kyiL00uNsUVX5rxr (дата обращения 01.02.2021).
4. Документация по СУБД PostgreSQL [Электронный ресурс]. – 2020 – URL: https://www.postgresql.org/docs/ (дата обращения 14.11.2020).

# ПРИЛОЖЕНИЕ А SQL скрипты на создание и заполнение базы данных, на создание учетных записей пользователей, на назначение привилегий

Все скрипты на создание и заполнение базы данных доступны по ссылке на GitHub <https://github.com/golangman/generator_builets>.

*--*

*-- PostgreSQL database dump*

*--*

*-- Dumped from database version 12.5 (Ubuntu 12.5-1.pgdg16.04+1)*

*-- Dumped by pg\_dump version 12.5 (Ubuntu 12.5-1.pgdg16.04+1)*

SET statement\_timeout = 0;

SET lock\_timeout = 0;

SET idle\_in\_transaction\_session\_timeout = 0;

SET client\_encoding = 'UTF8';

SET standard\_conforming\_strings = on;

SELECT pg\_catalog.set\_config('search\_path', '', false);

SET check\_function\_bodies = false;

SET xmloption = content;

SET client\_min\_messages = warning;

SET row\_security = off;

SET default\_tablespace = '';

SET default\_table\_access\_method = heap;

*--*

*-- Name: complexity; Type: TABLE; Schema: public; Owner: misha*

*--*

CREATE TABLE public.complexity (

complexity character varying(16) NOT NULL

);

ALTER TABLE public.complexity OWNER TO misha;

*--*

*-- Name: professors; Type: TABLE; Schema: public; Owner: misha*

*--*

CREATE TABLE public.professors (

id integer NOT NULL,

first\_name character varying(16) NOT NULL,

last\_name character varying(16) NOT NULL,

patronymic character varying(16) NOT NULL

);

ALTER TABLE public.professors OWNER TO misha;

*--*

*-- Name: professors\_id\_seq; Type: SEQUENCE; Schema: public; Owner: misha*

*--*

CREATE SEQUENCE public.professors\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

ALTER TABLE public.professors\_id\_seq OWNER TO misha;

*--*

*-- Name: professors\_id\_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: misha*

*--*

ALTER SEQUENCE public.professors\_id\_seq OWNED BY public.professors.id;

*--*

*-- Name: subjects; Type: TABLE; Schema: public; Owner: misha*

*--*

CREATE TABLE public.subjects (

professor\_id integer NOT NULL,

name character varying(32) NOT NULL

);

ALTER TABLE public.subjects OWNER TO misha;

*--*

*-- Name: tasks; Type: TABLE; Schema: public; Owner: misha*

*--*

CREATE TABLE public.tasks (

text character varying(256) NOT NULL,

complexity\_id character varying(16) NOT NULL,

subject\_professor\_id integer NOT NULL,

subject\_name character varying(32) NOT NULL

);

ALTER TABLE public.tasks OWNER TO misha;

*--*

*-- Name: users; Type: TABLE; Schema: public; Owner: misha*

*--*

CREATE TABLE public.users (

login character varying(32) NOT NULL,

password character varying(32) NOT NULL

);

ALTER TABLE public.users OWNER TO misha;

*--*

*-- Name: professors id; Type: DEFAULT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.professors ALTER COLUMN id SET DEFAULT nextval('public.professors\_id\_seq'::regclass);

*--*

*-- Data for Name: complexity; Type: TABLE DATA; Schema: public; Owner: misha*

*--*

COPY public.complexity (complexity) FROM stdin;

Легкий

Средний

Сложный

\.

*--*

*-- Data for Name: professors; Type: TABLE DATA; Schema: public; Owner: misha*

*--*

COPY public.professors (id, first\_name, last\_name, patronymic) FROM stdin;

1 Александр Глускер Игоревич

2 Владимир Петросян Степанович

\.

*--*

*-- Data for Name: subjects; Type: TABLE DATA; Schema: public; Owner: misha*

*--*

COPY public.subjects (professor\_id, name) FROM stdin;

1 История

2 Информатика

2 Программирование

1 ОБЖ

\.

*--*

*-- Data for Name: tasks; Type: TABLE DATA; Schema: public; Owner: misha*

*--*

COPY public.tasks (text, complexity\_id, subject\_professor\_id, subject\_name) FROM stdin;

Сколько лет Путину? Легкий 2 Информатика

Возникновение и развитие Древнерусского государства (IX – начало ХII в.) Легкий 1 История

Экономическое и социально-политическое развитие России в начале ХХ в Легкий 1 История

Культура Древней Руси (Х–ХIII вв.). Значение принятия христианства Легкий 1 История

Революция 1905–1907 гг.: причины, этапы, значение. Средний 1 История

Борьба Руси против внешних вторжений в ХIII в Средний 1 История

Реформы П.А. Столыпина. Направления, итоги и значение аграрной реформы. Средний 1 История

Объединение русских земель вокруг Москвы и становление единого Российского государства в ХIV–XV вв. Средний 1 История

Культура России в начале ХХ в. (1900–1917 гг.), ее вклад в мировую культуру Средний 1 История

Московская Русь в эпоху Ивана Грозного Сложный 1 История

Участие России в Первой мировой войне: причины, роль Восточного фронта, последствия Сложный 1 История

Основные направления внешней политики и расширение территории Российского государства в ХV–XVI вв. Сложный 1 История

1917 год в России (основные события, их характер и значение). Сложный 1 История

Культура и духовная жизнь Руси в ХIV–XV вв Сложный 1 История

Сколько мне лет? Сложный 2 Информатика

Внешняя политика России в конце XIX – начале ХХ в. Русско-японская война: причины, ход военных действий, итоги и последствия. Легкий 1 История

Политическая раздробленность на Руси. Русь удельная (XII–XIII вв.) Легкий 1 История

Образование государства Киевская Русь: предпосылки, причины, точки зрения на эту проблемы в историографии. Легкий 1 История

«Норманнская теория» и антинорманнизм в российской исторической науке. Легкий 1 История

Важнейшие причины принятия христианства Руcью.\n Легкий 1 История

Проблема возникновения феодализма в России. Агрессия немецких и шведских феодалов. Ледовое побоище. Легкий 1 История

Походы Батыя на Северо-Восточную и Южную Русь. Влияние татаро-монгольского нашествия и ига на развитие русских земель. Легкий 1 История

Особенности объединительного процесса на Руси. Легкий 1 История

Становление самодержавия в России.\n Легкий 1 История

Внутренняя политика Московской Руси во второй половине XVI века. Легкий 1 История

Причины, сущность и последствия Смуты. Проблема царской власти в период Смуты. Легкий 1 История

Особенности политического развития России во второй половине XVII века. Становление абсолютизма.\n Легкий 1 История

Сущность и последствия церковной реформы и Раскола в России в XVII века. Легкий 1 История

Особенности «просвещенного абсолютизма» в Западной Европе и в России. Легкий 1 История

Основные направления и влияние на развитие страны реформ Петра I. Легкий 1 История

Начало модернизации в России и ее особенности. Легкий 1 История

Основные направления внутренней и внешней политика России при Екатерине II. Легкий 1 История

Особенности социально-экономического и политического развития России в первой половине XIX века. Сущность «правительственного либерализма» Средний 1 История

Основные направления развития войны 1812 года и заграничных походов российской армии.\n Средний 1 История

Декабристы и их влияние на развитые страны.\n Средний 1 История

Основные направления развития общественной мысли в 30-50-е годы XIX века.\n Средний 1 История

Крымская война и е влияние на развитые страны.\n Средний 1 История

Сущность второй промышленной революции и ее влияние на мировое развитие.\n Средний 1 История

Содержание «Великих реформ» в России. Соотношение либерального и консервативного курсов в правительственной политике Александра II.\n Средний 1 История

Внешняя политика России во второй половине XIX – начале XX вв.\n Средний 1 История

Народническое движение.\n Средний 1 История

Цель и значение реформ С.Ю.Витте.\n Средний 1 История

Особенности внутренней политики правительства Александра III.\n Средний 1 История

Причины и следствия революции 1905 – 1907 гг. в России.\n Средний 1 История

Особенности российского парламентаризма.\n Средний 1 История

Крестьянская община в России: ее достоинства и недостатки.\n Средний 1 История

Цели и содержание реформ П.Столыпина.\n Средний 1 История

Предпосылки и итоги Первой мировой войны. Влияние войны на внутреннее положение России.\n Средний 1 История

Особенности российского либерализма. Либеральные партии в России.\n Средний 1 История

Цели и программные положения революционных партий России.\n Средний 1 История

Цели и программные положения консервативных партий в России.\n Сложный 1 История

Причины и особенности революции в 1917 году в России.\n Сложный 1 История

Представления о будущей государственности у большевиков до 1917 и их реальная политика в 1917-1918 гг.\n Сложный 1 История

Причины и следствия гражданской войны в России.\n Сложный 1 История

Основные черты политики «военного коммунизма».\n Сложный 1 История

Новая экономическая политика: причины, сущность, следствия.\n Сложный 1 История

Основные тенденции мирового развития СССР в 20-е – 30-е гг.\n Сложный 1 История

Причины зарождения фашизма в Европе.\n Сложный 1 История

Особенности международной ситуации в 30 – 40-х гг. XX в. и внешняя политика СССР.\n Сложный 1 История

Великая Отечественная война, ее основные этапы, итоги и уроки. Проблемы изучения Второй мировой войны в историографии.\n Сложный 1 История

Основные тенденции развития СССР в 1950 – 1980-е гг.\n Сложный 1 История

Сущность третьей научно-технической революции, постиндустриальной цивилизации.\n Сложный 1 История

Основные тенденции мирового развития на современном этапе.\n Сложный 1 История

Особенности развития СССР в период «перестройки».\n Сложный 1 История

Распад СССР и его последствия.\n Сложный 1 История

Внутренняя политика России в 1991 – 2011 гг.\n Сложный 1 История

Внешняя политика СССР и РФ 1985 – 2011 гг.\n Сложный 1 История

Объекты Средний 2 Программирование

Основы алгоритмизации и программирования. Алгоритмы: свойства, способы, описания. Средний 2 Программирование

Виды алгоритмов и основные принципы составления. Средний 2 Программирование

Основные положения и методы использования базовых алгоритмических структур. Средний 2 Программирование

Введение в программирование. Средний 2 Программирование

Инструменты программирования. Средний 2 Программирование

Языки программирования. Средний 2 Программирование

Трансляторы Средний 2 Программирование

Языки программирования: эволюция, классификация. Средний 2 Программирование

Основные элементы языка. Идентификаторы Средний 2 Программирование

Структура программы Средний 2 Программирование

Типы данных в программе: целые типы, вещественные типы, логический тип, символьный тип. Средний 2 Программирование

Типы данных в программе: строковый тип данных. Средний 2 Программирование

Выражения и арифметические операции. Средний 2 Программирование

Оператор присваивания, оператор ввода и оператор вывода. Сложный 2 Программирование

Основные алгоритмические конструкции. Линейный оператор. Приведите пример. Сложный 2 Программирование

Основные алгоритмические конструкции. Оператор ветвления. Приведите пример. Сложный 2 Программирование

Основные алгоритмические конструкции. Оператор выбора. Приведите пример. Сложный 2 Программирование

Основные алгоритмические конструкции. Оператор цикла с параметром. Приведите пример. Сложный 2 Программирование

Понятие подпрограммы. Достоинства подпрограмм. Сложный 2 Программирование

Графические средства QBasic: графические режимы окна, цвет фона и цвет рисунка. Сложный 2 Программирование

Графические средства QBasic: графические примитивы, закраски и заполнения. Сложный 2 Программирование

Символьные строки QBasic: основные функции. Примеры. Сложный 2 Программирование

Структурированный тип данных: одномерные массивы. Приведите пример. Сложный 2 Программирование

Структурированный тип данных: двумерные массивы. Приведите пример Сложный 2 Программирование

Основные понятия ООП Сложный 2 Программирование

Классы объектов Средний 2 Программирование

Свойства Средний 2 Программирование

Метод Средний 2 Программирование

События Средний 2 Программирование

Инкапсуляция Средний 2 Программирование

Наследование Средний 2 Программирование

Полиморфизм Средний 2 Программирование

Визуальное проектирование интерфейса\t Средний 2 Программирование

Этапы разработки приложения Средний 2 Программирование

Структура проекта VB Легкий 2 Программирование

Среда разработки VB Легкий 2 Программирование

Переменные Легкий 2 Программирование

Константы Легкий 2 Программирование

Функция InputBox Легкий 2 Программирование

Функция MsgBox Легкий 2 Программирование

Ввод и вывод данных в VB Легкий 2 Программирование

Вывод результатов Легкий 2 Программирование

Математические функции и функции преобразования данных Легкий 2 Программирование

Функция Format Легкий 2 Программирование

Условные операторы Легкий 2 Программирование

Циклы Легкий 2 Программирование

Циклы с параметром Легкий 2 Программирование

Методы графики в VB\t Легкий 2 Программирование

Задание цвета\t Легкий 2 Программирование

Свойства объектов, влияющих на графические методы Легкий 2 Программирование

Применение метода Scale Легкий 2 Программирование

Объект управления Shape (Фигура) Легкий 2 Программирование

Объект управления Timer Легкий 2 Программирование

Объект управления ProgressBar\t Легкий 2 Программирование

Объект управления Slider (Движок) Легкий 2 Программирование

Объект управления CheckBox (Флажок) Легкий 2 Программирование

Объект управления OptionButton (Переключатель) Легкий 2 Программирование

Работа с несколькими формами Легкий 2 Программирование

\.

*--*

*-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: misha*

*--*

COPY public.users (login, password) FROM stdin;

@dmin @dmin

\.

*--*

*-- Name: professors\_id\_seq; Type: SEQUENCE SET; Schema: public; Owner: misha*

*--*

SELECT pg\_catalog.setval('public.professors\_id\_seq', 3, true);

*--*

*-- Name: complexity complexity\_pkey; Type: CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.complexity

ADD CONSTRAINT complexity\_pkey PRIMARY KEY (complexity);

*--*

*-- Name: professors professors\_pkey; Type: CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.professors

ADD CONSTRAINT professors\_pkey PRIMARY KEY (id);

*--*

*-- Name: subjects subjects\_pkey; Type: CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.subjects

ADD CONSTRAINT subjects\_pkey PRIMARY KEY (professor\_id, name);

*--*

*-- Name: tasks tasks\_pkey; Type: CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.tasks

ADD CONSTRAINT tasks\_pkey PRIMARY KEY (text);

*--*

*-- Name: users users\_pkey; Type: CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.users

ADD CONSTRAINT users\_pkey PRIMARY KEY (login);

*--*

*-- Name: subjects\_professor\_id; Type: INDEX; Schema: public; Owner: misha*

*--*

CREATE INDEX subjects\_professor\_id ON public.subjects USING btree (professor\_id);

*--*

*-- Name: tasks\_complexity\_id; Type: INDEX; Schema: public; Owner: misha*

*--*

CREATE INDEX tasks\_complexity\_id ON public.tasks USING btree (complexity\_id);

*--*

*-- Name: tasks\_subject\_professor\_id; Type: INDEX; Schema: public; Owner: misha*

*--*

CREATE INDEX tasks\_subject\_professor\_id ON public.tasks USING btree (subject\_professor\_id);

*--*

*-- Name: subjects subjects\_professor\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.subjects

ADD CONSTRAINT subjects\_professor\_id\_fkey FOREIGN KEY (professor\_id) REFERENCES public.professors(id) ON UPDATE CASCADE ON DELETE CASCADE;

*--*

*-- Name: tasks tasks\_complexity\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.tasks

ADD CONSTRAINT tasks\_complexity\_id\_fkey FOREIGN KEY (complexity\_id) REFERENCES public.complexity(complexity);

*--*

*-- Name: tasks tasks\_subject\_professor\_id\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.tasks

ADD CONSTRAINT tasks\_subject\_professor\_id\_fkey FOREIGN KEY (subject\_professor\_id) REFERENCES public.professors(id) ON UPDATE CASCADE ON DELETE CASCADE;

*--*

*-- Name: tasks tasks\_subject\_professor\_id\_subject\_name\_fkey; Type: FK CONSTRAINT; Schema: public; Owner: misha*

*--*

ALTER TABLE ONLY public.tasks

ADD CONSTRAINT tasks\_subject\_professor\_id\_subject\_name\_fkey FOREIGN KEY (subject\_professor\_id, subject\_name) REFERENCES public.subjects(professor\_id, name);

*--*

*-- PostgreSQL database dump complete*

*--*

# ПРИЛОЖЕНИЕ Б Код программы

\_database.py

from peewee import (

BigIntegerField,

CharField,

ForeignKeyField,

PostgresqlDatabase,

Model,

CompositeKey,

AutoField,

SQL,

)

from utils import config\_get\_params

\_name, \_host, \_port, \_user, \_password = config\_get\_params(

[

"database\_name",

"database\_host",

"database\_port",

"database\_username",

"database\_password",

]

)

driver = PostgresqlDatabase

database = driver(

\_name, host=\_host, port=\_port, user=\_user, password=\_password, autorollback=True

)

*# \* Базовая модель моделей базы данных для наследования драйвера*

class BaseModel(Model):

class Meta:

database = database

autorollback = True

*# \* Модель пользователей*

class Users(BaseModel):

login = CharField(max\_length=32, primary\_key=True, null=False)

password = CharField(max\_length=32)

*# \* Модель сложностей*

class Complexity(BaseModel):

complexity = CharField(max\_length=16, null=False, primary\_key=True)

*# \* Модель профессоров ( преподавателей )*

class Professors(BaseModel):

id = AutoField()

first\_name = CharField(max\_length=16, null=False)

last\_name = CharField(max\_length=16, null=False)

patronymic = CharField(max\_length=16, null=False)

*# \* Модель учебных предметов*

class Subjects(BaseModel):

professor = ForeignKeyField(

Professors,

to\_field="id",

on\_delete="CASCADE",

on\_update="CASCADE",

backref="subjects",

)

name = CharField(max\_length=32, null=False)

class Meta:

primary\_key = CompositeKey("professor", "name")

*# \* Модель заданий*

class Tasks(BaseModel):

text = CharField(max\_length=32, primary\_key=True, null=False)

complexity = ForeignKeyField(Complexity,)

subject\_professor = ForeignKeyField(

Professors,

to\_field="id",

on\_delete="CASCADE",

on\_update="CASCADE",

backref="subjects",

)

subject\_name = CharField(max\_length=32, null=False)

class Meta:

database = database

constraints = [

SQL(

"FOREIGN KEY(subject\_professor\_id, subject\_name) "

"REFERENCES Subjects(professor\_id, name)"

)

]

*# \* Инициализация таблиц базы данных*

if database:

Users.create\_table(True)

Complexity.create\_table(True)

Professors.create\_table(True)

Subjects.create\_table(True)

Tasks.create\_table(True)

database.py

from peewee import PeeweeException

from \_database import \*

from utils import \_AutoInit

class \_Professor(\_AutoInit):

pass

class \_Subject(\_AutoInit):

pass

class \_Difficulty(\_AutoInit):

pass

class \_TaskCount(\_AutoInit):

pass

class DatabaseManager:

def \_\_init\_\_(self) -> None:

*self*.driver = database

def sign\_in(self, login: str, password: str) -> bool or Exception:

cursor = database.execute\_sql(

f"SELECT \* FROM users WHERE users.login='{login}' AND users.password='{password}';"

)

user = cursor.fetchone()

if not user:

return False

return user

def sign\_up(self, login: str, password: str) -> bool or Exception:

try:

database.execute\_sql(

f"INSERT INTO users(login, password) VALUES('{login}', '{password}');"

)

return True

except PeeweeException:

return False

except:

return None

def get\_subjects\_for\_professor(self, professor\_id: int) -> list:

cursor = database.execute\_sql(

f"SELECT name FROM subjects WHERE professor\_id={professor\_id};"

)

subjects = list()

for row in cursor.fetchall():

subjects.append(\_Subject(professor\_id=professor\_id, name=row[0],))

return subjects

def get\_all\_professors(self) -> list:

cursor = database.execute\_sql(

f"SELECT id, first\_name, last\_name, patronymic FROM professors;"

)

professors = list()

for row in cursor.fetchall():

professors.append(

\_Professor(

id=row[0], first\_name=row[1], last\_name=row[2], patronymic=row[3],

)

)

return professors

def get\_all\_difficulties(self) -> list:

cursor = database.execute\_sql(f"SELECT complexity FROM complexity;")

difficulties = list()

for row in cursor.fetchall():

difficulties.append(\_Difficulty(complexity=row[0],))

return difficulties

*# def get\_tasks(complexity) -> dict:*

# cursor = database.execute\_sql(f"SELECT complexity from complexity;")

# tasks = list()

# for row in cursor.fetchall():

# difficulties.append(\_Difficulty(complexity=row[0],))

# return difficulties

def get\_count\_tasks(

self, comlexity\_list, subject\_professor\_id, subject\_name

) -> list:

comlexity\_list\_str = "".join(

[f"'{comlexity}'," for comlexity in comlexity\_list]

)[:-1]

cursor = database.execute\_sql(

f"SELECT tasks.complexity\_id, count(tasks.text) FROM tasks WHERE tasks.complexity\_id IN ({comlexity\_list\_str}) AND tasks.subject\_professor\_id = {subject\_professor\_id} AND subject\_name = '{subject\_name}' GROUP BY tasks.complexity\_id;"

)

tasks = list()

for row in cursor.fetchall():

tasks.append(\_TaskCount(comlexity=row[0], count=row[1]))

comlexity\_list.remove(row[0])

for comlexity in comlexity\_list:

tasks.append(\_TaskCount(comlexity=comlexity, count=0))

return tasks

def get\_tasks(self, complexity, subject\_professor\_id, subject\_name, limit):

cursor = database.execute\_sql(

f"""SELECT tasks.text FROM tasks

WHERE tasks.complexity\_id = '{complexity}'

AND tasks.subject\_professor\_id = {subject\_professor\_id}

AND tasks.subject\_name = '{subject\_name}'

LIMIT {limit};"""

)

tasks = list()

for row in cursor.fetchall():

tasks.append(row[0].strip())

return tasks

if \_\_name\_\_ == "\_\_main\_\_":

db = DatabaseManager()

result = db.sign\_up("login4", "password2")

print(result)

main.py

import tkinter

from tkinter import Button, Entry, Frame, StringVar, messagebox

from database import DatabaseManager

from ui import Window

*# label = tkinter.Label(root, text="Hello World!") # Create a text label*

*# label.pack(padx=0, pady=0) # Pack it into the window*

def main(root):

window = Window(root, db)

window.login\_page\_init()

root.eval("tk::PlaceWindow . center")

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

db = DatabaseManager()

root = tkinter.Tk()

main(root)

utils.py

from json import dump, load

CONFIG\_PATH = "./config.json"

def config\_read(path: str = CONFIG\_PATH):

with open(path) as fh:

config = load(fh)

return config

def config\_write(data, path: str = CONFIG\_PATH):

with open(path, "w") as outfile:

dump(data, outfile)

return data

def config\_get(parameter: str, path: str = CONFIG\_PATH):

config = config\_read(path)

return config[parameter] if parameter in config else None

def config\_get\_params(parameters: list, path: str = CONFIG\_PATH):

config = config\_read(path)

result = [config[parameter] for parameter in parameters]

return result

def config\_set(parameter, value, path: str = CONFIG\_PATH):

config = config\_read(path)

config[parameter] = value

config\_write(config)

return config[parameter] if parameter in config else None

def md5(string):

from hashlib import md5 as \_md5

return \_md5(string.encode("utf-8")).hexdigest()

class \_AutoInit:

def \_\_init\_\_(self, \*\*kwargs) -> None:

for key, value in list(kwargs.items()):

*self*.\_\_dict\_\_[key] = value

class PrintData:

def \_\_init\_\_(self, professor\_name, subject\_name) -> None:

pass

ui.py

from tkinter import Button, Entry, Frame, StringVar, messagebox, Label

from tkinter.constants import GROOVE

from tkinter.ttk import Combobox

from tkinter.simpledialog import askstring

from utils import config\_read, md5

from random import SystemRandom

from collections import Counter

from os import mkdir

import os

import subprocess

class Window:

def \_\_init\_\_(self, root, database\_manager) -> None:

*self*.root = root

*self*.root.title("Авторизация")

*self*.root.geometry("255x175+0+0")

*self*.database\_manager = database\_manager

def main\_page\_init(self):

*self*.root.geometry("330x120+0+0")

*self*.root.title("Генератор билетов")

*self*.root.eval("tk::PlaceWindow . center")

*self*.FrameButtons = Frame(*self*.root, bg="white")

*self*.FrameButtons.place(x=0, y=0)

*self*.\_professors = list()

professors = *self*.database\_manager.get\_all\_professors()

*self*.professors\_combobox\_values = list()

for professor in professors:

full\_name = f"{professor.first\_name} {professor.last\_name} {professor.patronymic} {professor.id}"

*self*.professors\_combobox\_values.append(full\_name)

*self*.message\_label = Label(

*self*.FrameButtons,

text="Выбери преподавателя из списка",

font=("times new roman", 20, "bold"),

)

*self*.message\_label.grid(row=0, column=0, pady=10, padx=10)

*self*.combobox\_professor = Combobox(

*self*.FrameButtons,

values=*self*.professors\_combobox\_values,

state="readonly",

width=30,

)

*self*.combobox\_professor.current(0)

*self*.combobox\_professor.grid(row=1, column=0, padx=10)

*self*.button\_create = Button(

*self*.FrameButtons,

command=*self*.create,

text="Выбрать",

font=("times new roman", 15, "bold"),

bg="yellow",

fg="red",

width=35,

)

*self*.button\_create.grid(row=2, column=0, padx=10)

def create(self):

if *self*.\_\_dict\_\_.get("professors\_names") is None:

*self*.professors\_names = [

*self*.combobox\_professor.get(),

]

*self*.combobox\_professor.destroy()

*self*.professors\_combobox\_values.remove(*self*.professors\_names[0])

*self*.combobox\_professor = Combobox(

*self*.FrameButtons,

values=*self*.professors\_combobox\_values,

state="readonly",

width=30,

)

*self*.combobox\_professor.current(0)

*self*.combobox\_professor.grid(row=1, column=0, padx=10)

return

elif len(*self*.professors\_names) == 1 and (

*self*.\_\_dict\_\_.get("subjects") is None

):

*self*.professors\_names.append(*self*.combobox\_professor.get())

*self*.combobox\_professor.destroy()

professor = *self*.professors\_names.pop(0)

professor\_split = professor.split()

*self*.\_professors.append(

f"{professor\_split[1]} {professor\_split[0][:1]}.{professor\_split[2][:1]}."

)

professor\_id = int(professor\_split[-1])

*self*.message\_label.destroy()

*self*.message\_label = Label(

*self*.FrameButtons,

text=f"{professor\_split[1]} {professor\_split[0]} {professor\_split[2]}",

font=("times new roman", 20, "bold"),

)

*self*.message\_label.grid(row=0, column=0, pady=10, padx=10)

subjects = *self*.database\_manager.get\_subjects\_for\_professor(

professor\_id=professor\_id

)

combobox\_subjects\_values = list()

for subject in subjects:

combobox\_subjects\_values.append(

f"{subject.name} {subject.professor\_id}"

)

*self*.combobox\_subjects = Combobox(

*self*.FrameButtons,

values=combobox\_subjects\_values,

state="readonly",

width=30,

)

*self*.combobox\_subjects.current(0)

*self*.combobox\_subjects.grid(row=1, column=0, padx=10)

*self*.subjects = list()

return

elif len(*self*.\_\_dict\_\_.get("subjects")) == 0:

*self*.subjects.append(*self*.combobox\_subjects.get())

*self*.combobox\_subjects.destroy()

professor = *self*.professors\_names.pop(0)

professor\_split = professor.split()

*self*.\_professors.append(

f"{professor\_split[1]} {professor\_split[0][:1]}.{professor\_split[2][:1]}."

)

professor\_id = int(professor\_split[-1])

*self*.message\_label.destroy()

*self*.message\_label = Label(

*self*.FrameButtons,

text=f"{professor\_split[1]} {professor\_split[0]} {professor\_split[2]}",

font=("times new roman", 20, "bold"),

)

*self*.message\_label.grid(row=0, column=0, pady=10, padx=10)

subjects = self.database\_manager.get\_subjects\_for\_professor(

professor\_id=professor\_id

)

combobox\_subjects\_values = list()

for subject in subjects:

combobox\_subjects\_values.append(

f"{subject.name} {subject.professor\_id}"

)

*self*.combobox\_subjects = Combobox(

*self*.FrameButtons,

values=combobox\_subjects\_values,

state="readonly",

width=30,

)

*self*.combobox\_subjects.current(0)

*self*.combobox\_subjects.grid(row=1, column=0, padx=10)

return

elif len(*self*.subjects) == 1:

*self*.subjects.append(*self*.combobox\_subjects.get())

*self*.combobox\_subjects.destroy()

*self*.message\_label.destroy()

*self*.message\_label = Label(

*self*.FrameButtons,

text="Выберите кол-во вопросов",

font=("times new roman", 20, "bold"),

)

*self*.message\_label.grid(row=0, column=0, pady=10, padx=10)

*self*.combobox\_questions\_count = Combobox(

*self*.FrameButtons, values=[3, 4, 5], state="readonly", width=30,

)

*self*.combobox\_questions\_count.current(0)

*self*.combobox\_questions\_count.grid(row=1, column=0, padx=10)

*self*.questions\_count = 0

return

elif *self*.questions\_count == 0:

*self*.questions\_count = int(*self*.combobox\_questions\_count.get())

*self*.combobox\_questions\_count.destroy()

*self*.message\_label.destroy()

*self*.is\_manually\_difficulty = messagebox.askyesno(

None, "Желаете установить сложность вручную?"

)

*# \* Перемешивание для непостоянства нечётного вопроса*

SystemRandom().shuffle(self.subjects)

if not self.is\_manually\_difficulty:

self.button\_create.destroy()

self.comboboxs\_difficulty = []

questions\_count\_range = range(self.questions\_count)

difficulties = [

\_.complexity for \_ in self.database\_manager.get\_all\_difficulties()

]

for \_ in questions\_count\_range:

combobox\_difficulty = Combobox(

self.FrameButtons,

values=difficulties,

state="readonly",

width=10,

)

combobox\_difficulty.current(

SystemRandom().randint(0, len(difficulties) - 1)

)

self.comboboxs\_difficulty.append((None, combobox\_difficulty,))

self.generate\_bilets()

return # FIXME: вызов генератора

self.button\_create.destroy()

self.comboboxs\_difficulty = []

difficulties = [

\_.complexity for \_ in self.database\_manager.get\_all\_difficulties()

]

questions\_count\_range = range(self.questions\_count)

for \_ in questions\_count\_range:

combobox\_difficulty = Combobox(

self.FrameButtons, values=difficulties, state="readonly", width=10,

)

subject = (

self.subjects[0]

if \_ < (self.questions\_count // 2)

else self.subjects[1]

)

subject = (" ".join(subject.split()[:-1])).strip()

message\_label = Label(

self.FrameButtons,

text=f"{subject} №{\_+1}",

font=("times new roman", 20, "bold"),

justify="left",

)

message\_label.grid(row=0 + \_, column=0)

combobox\_difficulty.current(0)

combobox\_difficulty.grid(row=0 + \_, column=1)

self.comboboxs\_difficulty.append((message\_label, combobox\_difficulty,))

self.button\_create = Button(

self.FrameButtons,

command=self.generate\_bilets,

text="Сгенерировать",

font=("times new roman", 15, "bold"),

bg="yellow",

fg="red",

)

self.button\_create.grid(

row=questions\_count\_range[-1] + 1, column=1,

)

self.root.geometry("400x200+0+0")

self.root.eval("tk::PlaceWindow . center")

return

def login\_page\_init(self) -> None:

self.Frame = Frame(self.root, bg="white")

self.Frame.place(x=0, y=0)

self.username = StringVar()

self.password = StringVar()

# \*Login entry

Entry(

self.Frame, textvariable=self.username, bd=5, relief=GROOVE, font=("", 15),

).grid(row=1, column=1, padx=20, pady=10)

# \*Password entry

Entry(

self.Frame,

textvariable=self.password,

bd=5,

relief=GROOVE,

font=("", 15),

show="\*",

).grid(row=2, column=1, padx=20)

# \*Sing in button

Button(

self.Frame,

command=self.sign\_in,

text="Sign In",

width=15,

font=("times new roman", 15, "bold"),

bg="yellow",

fg="red",

).grid(row=3, column=1, pady=10)

# \*Sing up button

Button(

self.Frame,

command=self.sign\_up,

text="Sign Up",

width=15,

font=("times new roman", 15, "bold"),

bg="yellow",

fg="red",

).grid(row=4, column=1, pady=10)

def sign\_up(self):

login, password = self.username.get(), self.password.get()

result = self.database\_manager.sign\_up(login, md5(password))

if not result:

messagebox.showerror("Регистрация", "Ошибка регистрации!", icon="warning")

return

messagebox.showinfo("Регистрация", "Пользователь успешно зарегистрирован!")

def sign\_in(self):

login, password = self.username.get(), self.password.get()

is\_sign\_in = self.database\_manager.sign\_in(login, md5(password))

if is\_sign\_in:

# \* Очистка фрейма, мусорных атрибутов

self.Frame.destroy()

del self.Frame

del self.username

del self.password

# \* Инициализация основной страницы

self.main\_page\_init()

return True

messagebox.showerror(

"Авторизация", "Логин и/или пароль неверны!", icon="warning"

)

return False

def generate\_bilets(self):

from tkinter.simpledialog import askinteger

from generator import draw, generate\_a4

config\_data = config\_read()

protocol = config\_data["default\_protocol"]

specialty = (config\_data["default\_specialty"],)

course = config\_data["default\_course"]

semester = config\_data["default\_semester"]

zam\_umr = config\_data["default\_zam\_umr"]

cicl\_comission = config\_data["default\_cicl\_comission"]

students\_count = askinteger(

None, "Сколько необходимо билетов?", initialvalue="20"

)

if not students\_count or students\_count < 1:

messagebox.showinfo(

"Информация",

"Вы не указали необхомое число студентов!\n\nПовторите попытку!",

)

return

protocol = self.get\_string(

"Введите номер протокола", 410, initialvalue=protocol

)

specialty = self.get\_string(

"Введите номер специальности", 440, initialvalue=specialty

)

course = self.get\_string(

"Введите номер курса", 220, initialvalue=course, \_get\_int=True

)

semester = self.get\_string(

"Введите номер семестра", 250, initialvalue=semester, \_get\_int=True

)

zam\_umr = self.get\_string(

"Введите краткое ФИО заместителя директора по УМР",

740,

initialvalue=zam\_umr,

)

cicl\_comission = self.get\_string(

"Введите краткое ФИО председателя цикловой комисси",

740,

initialvalue=cicl\_comission,

)

tasks\_difficulty = [combobox[1].get() for combobox in self.comboboxs\_difficulty]

cof = self.questions\_count // 2

first\_subject\_tasks, last\_subject\_tasks = (

tasks\_difficulty[:cof],

tasks\_difficulty[cof:],

)

first\_subject\_tasks\_count = dict(Counter(first\_subject\_tasks))

last\_subject\_tasks\_count = dict(Counter(last\_subject\_tasks))

rr = lambda x: list(x.split())

subjects = list(map(rr, self.subjects))

first\_subject\_has = self.database\_manager.get\_count\_tasks(

list(set(first\_subject\_tasks)), subjects[0][1], subjects[0][0]

)

last\_subject\_has = self.database\_manager.get\_count\_tasks(

list(set(last\_subject\_tasks)), subjects[1][1], subjects[1][0]

)

tasks\_list = list()

for comlexity in first\_subject\_has:

\_tasks\_limit = (

first\_subject\_tasks\_count[comlexity.comlexity] \* students\_count

)

if comlexity.count < \_tasks\_limit:

messagebox.showerror(

None,

f"Необходимо ещё {\_tasks\_limit-comlexity.count} заданий с сложностью '{comlexity.comlexity}' в предмете {subjects[0][0]} для генерации {students\_count} билетов!",

)

return

\_tasks\_list = self.database\_manager.get\_tasks(

complexity=comlexity.comlexity,

subject\_professor\_id=subjects[0][1],

subject\_name=subjects[0][0],

limit=\_tasks\_limit,

)

tasks\_list.append(\_tasks\_list)

for comlexity in last\_subject\_has:

\_tasks\_limit = (

last\_subject\_tasks\_count[comlexity.comlexity] \* students\_count

)

if comlexity.count < \_tasks\_limit:

messagebox.showerror(

None,

f"Необходимо ещё {\_tasks\_limit-comlexity.count} заданий с сложностью '{comlexity.comlexity}' в предмете {subjects[0][0]} для генерации {students\_count} билетов!",

)

return

\_tasks\_list = self.database\_manager.get\_tasks(

complexity=comlexity.comlexity,

subject\_professor\_id=subjects[1][1],

subject\_name=subjects[1][0],

limit=\_tasks\_limit,

)

tasks\_list.append(\_tasks\_list)

builet = None

from datetime import datetime

date = datetime.now()

dirname = f"./results/{students\_count} билетов {date.day}.{date.month}.{date.year} {date.hour}:{date.minute}:{date.second}"

mkdir(dirname)

students\_count\_range = range(students\_count)

for task\_idx in students\_count\_range:

builet\_tasks = list()

\_subject\_tasks = list(first\_subject\_tasks\_count.items()) + list(

last\_subject\_tasks\_count.items()

)

for idx in range(len(\_subject\_tasks)):

\_, count = \_subject\_tasks[idx]

for \_ in range(count):

builet\_tasks.append(tasks\_list[idx].pop(0))

new\_builet = draw(

str(protocol),

str(specialty),

str(course),

str(semester),

str(task\_idx + 1),

str(zam\_umr),

str(cicl\_comission),

self.\_professors,

[subject[0] for subject in subjects],

builet\_tasks,

)

if builet:

a4 = generate\_a4(builet, new\_builet)

a4.save(f"{dirname}/билеты{task\_idx}\_{task\_idx+1}.png")

builet = None

continue

if task\_idx == students\_count\_range[-1]:

a4 = generate\_a4(new\_builet, None)

a4.save(f"{dirname}/билет{task\_idx+1}.png")

break

builet = new\_builet

from subprocess import Popen

subprocess.Popen(["open", dirname])

exit()

def get\_string(

self, message: str, width: int, initialvalue: str, \_get\_int: bool = False

):

def check\_width(string, width):

from PIL import ImageFont, ImageDraw, Image

font = ImageFont.truetype("./docs/font.ttf", 60)

draw = ImageDraw.Draw(Image.new("RGBA", (1000, 1000,)), "RGBA",)

\_width, \_ = draw.textsize(string, font=font)

return True if \_width < width else False

def get(message):

result = askstring(None, message, initialvalue=initialvalue)

if not result or not (type(result) is str):

messagebox.showwarning(None, "Не оставляй поле пустым!")

return get(message)

if \_get\_int and not result.isdigit():

messagebox.showwarning(None, "Введите число!")

return get(message)

return result

string = get(message)

if check\_width(string, width):

return string

messagebox.showwarning(

None, "Допустимый лимит длинны нарушен, попробуйте что-то покороче!"

)

return self.get\_string(message, width, initialvalue)