МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Вятский государственный университет»**

**Колледж ВятГУ**

**КУРСОВОЙ ПРОЕКТ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| По междисциплинарному курсу | | | | | МДК 05.01 Проектирование и дизайн информационных | | | | |
| систем | | | | |
| Тема | Проектирование информационной системы для ВУЗа | | | | | | | | | |
| Студента (ки) | | | | Имя | | | | | | |
|  | | | *ФИО (полностью в родительном падеже)* | | | | | | | |
| Курс | | 3 | | | | Форма обучения | очная | |
|  | | *(арабской цифрой)* | | | |  | | *(очная, заочная)* | | |
| Основная профессиональная образовательная программа по специальности | | | | | | | | | | |
| 09.02.07 Информационные системы и программирование | | | | | | | | | | |
| (код и наименование специальности без кавычек) | | | | | | | | | | |

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия, инициалы)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(прописью, без сокращений)*

Киров, 2022

**РЕФЕРАТ**

Пояснительная записка к курсовому проекту содержит: 17 страниц, 7 рисунков, 7 использованных источников.

Объектом и предметом исследования является система автоматизированного тестирования.

Цель работы – ознакомление с процессом создания технического задания на разработку информационной системы.

Поставлена задача разработать техническое задание для информационной системы выбранного объекта исследования.

В процессе работы были проведены следующие исследования: 1) обзор предметной области, обзор аналогов и сравнительный анализ; 2) написание самого технического задания, состоящего из: назначения разработки, функциональных характеристик, условий эксплуатации и требований к составу и параметрам технических средств; 3) описание решения и концепции; 4) архитектура решения; 5) разработка схем бизнес-процессов с их описанием; 6) разработка схем алгоритмов и кода на естественном языке; 7) проектирование прототипа пользовательского интерфейса с описанием.

Элементами научного новшества полученных результатов является автоматизация тестирования.

Областью возможного практического применения являются: образовательные организации, тестирование обыденных пользователей.

Технико-экономическая и социальная значимость: внедрение систем адаптивного тестирования.

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ ПРОВЕДЕНИЯ ОБСЛЕДОВАНИЯ** 4](#_Toc164106926)

[**1.** **ОРГАНИЗАЦИОННАЯ СТРУКТУРА МАГАЗИН ПО ПРОДАЖЕ КОМПЬЮТЕРНЫХ ИГР** 5](#_Toc164106927)

[1.1. Назначение и связи подразделений 5](#_Toc164106928)

[1.2. Требования к программному обеспечению 6](#_Toc164106929)

[1.3. Анализ подразделений 6](#_Toc164106930)

[**2.** **ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 7](#_Toc164106931)

[2.1. Наименование программы 7](#_Toc164106932)

[2.2. Краткая характеристика области применения программы 7](#_Toc164106933)

[2.3 Эксплуатационное назначение программы 7](#_Toc164106934)

[2.4 Требования к программе 7](#_Toc164106935)

[2.5 Требования к функциональным характеристикам 8](#_Toc164106936)

[2.6 Требования к пользователю 8](#_Toc164106937)

[2**.**7 Основания для разработки 8](#_Toc164106938)

[2.8 Стадии и этапы разработки 9](#_Toc164106939)

[**3.** **СХЕМЫ БИЗНЕС-ПРОЦЕССОВ И ИХ ОПИСАНИЕ** 10](#_Toc164106940)

[3.1. Функциональный анализ ПО 10](#_Toc164106941)

[3.2 Логическая модель ПО 11](#_Toc164106942)

[3.3 Физическая модель ПО 13](#_Toc164106943)

[**4.** **ПРОЕКТИРОВАНИЕ ПО** 14](#_Toc164106944)

[4.1. Формирование диаграммы Use Case 14](#_Toc164106945)

[4.2. Формирование схемы алгоритма функционирования АРМ ИС 15](#_Toc164106946)

[4.3. Экранные формы 16](#_Toc164106947)

[**ЗАКЛЮЧЕНИЕ** 21](#_Toc164106948)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 22](#_Toc164106949)

# **ЦЕЛЬ ПРОВЕДЕНИЯ ОБСЛЕДОВАНИЯ**

В процессе предпроектного обследования информационной системы для ВУЗА изучаются основные направления деятельности, оказываемые услуги и организационная структура магазина по продаже компьютерных игр. Определяются функции подразделений, существующие информационные взаимосвязи между подразделениями, внутренний и внешний документооборот.

На основе анализа указанной информации определяются требуемые учетные подсистемы, охватывающие несколько подразделений, каждое из которых заинтересовано в оперативности и актуальности данных. Разрабатываются рекомендации по усовершенствованию документооборота, исключению дублирования информации.

Изучается текущий уровень автоматизации: определяется перечень разработанных подсистем, состав автоматизированных рабочих мест и круг решаемых задач с целью определения функциональной полноты системы и автоматизации учетных функций. Разрабатываются предложения по требуемому составу выбранных подсистем, уточнению перечня задач, подлежащих автоматизации, и расширению состава автоматизированных рабочих мест с целью получения полной оперативной информации по бухгалтерскому, оперативному и управленческому учету производственно-хозяйственной деятельности ВУЗА.

Осуществляется обследование существующих бизнес-процессов и бизнес-процедур.

1. **ОРГАНИЗАЦИОННАЯ СТРУКТУРА МАГАЗИН ПО ПРОДАЖЕ КОМПЬЮТЕРНЫХ ИГР**
   1. Назначение и связи подразделений

* *Ректор* руководит вузом, неся всю полноту ответственности за последствия принимаемых решений и результаты деятельности. Организует работу и взаимодействие всех структурных подразделений, направляет их деятельность на развитие и совершенствование работы вуза, повышение эффективности работы, увеличение прибыли, качества и конкурентоспособности. Организует работу по улучшению качества услуг, внедрение в систему средств комплексной механизации и автоматизации технологических процессов.
* *Администратор* ежедневно координирует и контролирует работу. В обязанности администратора входит обновление данных о студентах, кафедрах, преподаваемых дисциплинах и преподавателях.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рисунок 1 – Структурная схема вуза

* 1. Требования к программному обеспечению

Программное обеспечение, на базе которого реализованы рабочие места, разработано отделом вуза. Программное обеспечение каждого рабочего места представляет собой программный комплекс, разработанный независимо от других и работающий автономно. Программы написаны в системе СУБД PostgreSQL. Сопровождением программного обеспечения занимается IT-отдел.

* 1. Анализ подразделений

*Исполнительный отдел*, оказывает работу по работе вуза.

*Управленческий отдел*. Структурные подразделения данного блока обеспечивают административную работу. Они передают и получают информацию из других отделов, анализирует ее, преобразуя в необходимую отчетность, как внутреннюю, так и внешнюю.

1. **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

## 2.1. Наименование программы

Наименование разрабатываемой программы: Проектирование информационной системы для ВУЗа.

## 2.2. Краткая характеристика области применения программы

Программа предназначена для обеспечения работы ВУЗА. Программа позволяет хранить, просматривать, добавлять и удалять информацию о кафедрах, дисциплинах, студентах и преподавателях в базе данных.

## 2.3 Эксплуатационное назначение программы

Программа должна эксплуатироваться как desk-top приложение и предназначена для работы ВУЗА и пользования её сотрудников.

## 2.4 Требования к программе

Описание программы: приложение открывается с помощью desk-top приложения. Программа содержит несколько оконных форм. На первой форме осуществляется авторизация, после которой происходит переход к администраторской панели. В ней можно взаимодействовать со студентами, кафедрами, дисциплинами и преподавателями путём просмотра информации в отдельном оке, добавлением новых и удалением существующих.

## 2.5 Требования к функциональным характеристикам

Программа должна обеспечивать выполнение следующий функций:

– возможность авторизации пользователя (администратора);

– возможность добавлять, удалять, обновлять, просматривать информацию о дисциплинах, студентах, преподавателях, кафедрах.

## 2.6 Требования к пользователю

Пользователь должен иметь следующие навыки:

* Навыки использования персонального компьютера;
* Навыки управления манипулятором типа «мышь» и «клавиатура».

## 2.7 Основания для разработки

Основанием для разработки является задание, полученное в рамках дисциплины «Проектирование и дизайн ИС», выданное преподавателем Мамаевой Екатериной Александровной.

## 2.8 Стадии и этапы разработки

Разработка проводилась в три стадии:

– составление технического задания;

– создание технического проекта;

– создание программы.

На стадии «Составление техническое задание» происходит этап разработки, согласования и утверждения настоящего технического задания.

На стадии «Создание технического проекта» происходит разработка программы, разработка программной документации и испытания программы.

На стадии «Создание программы» происходит подготовка и передача программы на проверку.

1. **СХЕМЫ БИЗНЕС-ПРОЦЕССОВ И ИХ ОПИСАНИЕ**
   1. Функциональный анализ ПО

На рисунке 2 изображена диаграмма функциональной модели программного обеспечения для деятельности вуза.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 2 – Контекстная диаграмма магазина по продаже компьютерных игр

Внешней входящей информацией являются:

* Администратор.

Управляющим воздействием будут являться:

* правила ВУЗА.

Механизмом управления будут являться:

* бд;

Внешней исходящей информацией будет являться:

* автоматизированная работа ВУЗА.

Для разбиения сложного процесса на составляющие подфункции применяется декомпозиция.

Диаграмма декомпозиции, представляет контекстную функцию, разложенную на 4 составляющих более простых функций.

1. Выбор раздела или функции.
2. Заполнение соответствующей формы.
3. Вывод окна об успехе или ошибке формы.
4. Вывод результата.

На рисунке 3 показан результат декомпозиции процесса оказания услуги.

Изображение выглядит как текст, линия, снимок экрана, диаграмма

Автоматически созданное описание

Рисунок 3 – Результат процесса декомпозиции процесса покупки продукции

## 3.2 Логическая модель ПО

Логическая модель данных является визуальным графическим представлением структур данных, их атрибутов и связей. Логическая модель представляет данные таким образом, чтобы они легко воспринимались.

Основными компонентами логической модели являются:

* сущности;
* атрибуты сущности;
* связи между сущностями.

Для описания мы используем нотацию IDEF1X, которая представляет структуру данных, описывающего предприятия. Методология IDEF1X – один из подходов к моделированию данных, основанный на концепции "сущность – связь".

Сущность в методологии IDEF1X является не зависимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями, а если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности, то сущность такая называется зависимой.

Логическая модель по нотации IDEF1X представлена на рисунке 4.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 4 – Логическая модель программного обеспечения

## 3.3 Физическая модель ПО

Физическая модель базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей.

Физическая модель строится на основе логической с учетом ограничений, накладываемых возможностями выбранной СУБД.

На рисунке 5 представлена физическая модель программы, которая описывает 4 таблицы, которые между собой имеют связи.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

Рисунок 5 – Физическая модель программного обеспечения

1. **ПРОЕКТИРОВАНИЕ ПО**
   1. Формирование диаграммы Use Case

Диаграмма «Use Case» – диаграмма вариантов использования, которая отражает отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать программу на концептуальном уровне.

Она состоит из графической диаграммы, описывающей действующие лица, прецеденты, спецификации.

Диаграмма вариантов использования по программному обеспечению представлена на рисунке 6.

Изображение выглядит как текст, диаграмма, линия, зарисовка

Автоматически созданное описание

Рисунок 6 – Диаграмма Use Case

* 1. Формирование схемы алгоритма функционирования АРМ ИС

Линейность алгоритма заключается в том, что все действия в основном модуле программы происходят последовательно. Ветвящийся алгоритм необходим для того, чтобы выбирать необходимые данные из файлов. Это значит, что при нажатии определённой кнопки следует процесс, который был описан в событии при нажатии на каждую кнопку.

Схема алгоритма функционирования АРМ ИС бизнес-процесса представлена на рисунка 7.

Изображение выглядит как диаграмма, линия, Технический чертеж, снимок экрана

Автоматически созданное описание

Рисунок 7 – Схема алгоритма функционирования АРМ ИС

* 1. Экранные формы

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 8 – Прототип экранной формы авторизации

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 9 – Прототип главной экранной формы

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – Прототип экранной формы студентов

Изображение выглядит как текст, снимок экрана, Прямоугольник

Автоматически созданное описание

Рисунок 11 – Прототип экранной формы преподавателей

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 12 – Прототип экранной формы дисциплин

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 13 – Прототип экранной формы добавления студента

Изображение выглядит как текст, снимок экрана, Прямоугольник, Шрифт

Автоматически созданное описание

Рисунок 14 – Прототип экранной формы удаления студента

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 15 – Прототип экранной формы удаления кафедры

Изображение выглядит как текст, снимок экрана, Прямоугольник, Шрифт

Автоматически созданное описание

Рисунок 16 – Прототип экранной формы удаления преподавателя

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Автоматически созданное описание

Рисунок 17 – Прототип экранной формы удаления дисциплины

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

Рисунок 18 – Прототип экранной формы добавления дисциплины

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 19 – Прототип экранной формы добавления преподавателя

Изображение выглядит как текст, снимок экрана, Прямоугольник, диаграмма

Автоматически созданное описание

Рисунок 20 – Прототип экранной формы добавления кафедры

**ЗАКЛЮЧЕНИЕ**

В результате написания курсовой работы был проведён анализ поставленных проблем с их дальнейшим решением, а также усвоена работа аналогичных систем. Благодаря этому был решён список поставленных задач.

Целью работы было ознакомление с процессом создания технического задания на разработку информационной системы.

В процессе выполнения работы был проведён обзор предметной области, написано техническое задание, описаны решения и концепция проекта, разработаны схемы бизнес-процессов с их описанием, спроектирован прототип пользовательского интерфейса с описанием.

При разработке программного обеспечения для информационной системы ВУЗА были приобретены разнообразные навыки, необходимые для обеспечения качественной работы приложения. Также были получены навыки написания технического задания на создание автоматизированного программного обеспечения с учётом особенностей предметной области.

Тем не менее, в ходе работы не удалось реализовать вывод отчётности по происходящим в программе процессам.

Таким образом, разработанное техническое задание и программное обеспечение может значительно облегчить и ускорить качество работы сотрудникам, и организациям, которые работают в данной сфере.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы (пример) – Текст электронный // Корпоративные хранилища данных. Интеграция систем. Проектная документация: [сайт] – 2010. – URL: https://www.prj-exp.ru/patterns/pattern\_tech\_task.php#tech\_task2 (дата обращения: 09.11.2020).
2. Логическая модель ИС – Текст электронный // Студвуд: [сайт] – 2020. – URL: https://studwood.ru/1050371/informatika/logicheskaya\_model (дата обращения: 22.11.2020).
3. DFD – диаграммы потоков данных – Текст электронный // Инфопедия: [сайт] – 2016. – URL: https://infopedia.su/12x891b.html (дата обращения: 27.11.2020).
4. Проектирование USE CASE диаграммы – Текст электронный // Программные решения для бизнеса: [сайт]. – 2020. – URL: https://nationalteam.worldskills.ru/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/ (дата обращения: 30.11.2020).
5. Основы UML. Диаграммы последовательностей – Текст электронный // Блог Программиста: [сайт]. – 2016. – URL: https://pro-prof.com/archives/2769 (дата обращения: 02.12.2020).
6. Целостность баз данных – Текст электронный // URL: http://www.ssofta.narod.ru/bd/13.htm (дата обращения: 02.12.2020).
7. ГОСТ 34. Разработка автоматизированной системы управления (АСУ). – Текст: электронный // Электронный фонд правовой и нормативно-технической документации: официальный сайт. - 2020. - URL: http://docs.cntd.ru/document/gost-34-602-89 (дата обращения: 12.11.2020).

**Приложение 1. – Список сокращений**

БД – база данных;

ПО – программное обеспечение;

СУБД – система управления базами данных;

ВУЗ – высшее учебное заведение.

**Приложение 2. – Листинг программы**

Файл main.py:

import sys

from PySide6.QtWidgets import QApplication

from pages.admin\_window import AdminWindow

from pages.auth import Auth

def main():

window = []

app = QApplication(sys.argv)

auth\_window = Auth()

window.append(auth\_window)

sys.exit(app.exec())

if \_\_name\_\_ == '\_\_main\_\_':

main()

Файл \_const.py:

SIZE = {

"SMALLER": (320, 160),

"LITTLE": (480, 240),

"SMALL": (640, 320),

"DEFAULT": (880, 440),

"MEDIUM": (1024, 768),

"BIG": (1120, 560),

"LARGE": (1280, 720),

"FULL": (1920, 1080),

}

Файл auth.py:

from PySide6.QtWidgets import \*

from PySide6.QtCore import \*

from pages.\_const import SIZE

from pages.admin\_window import AdminWindow

from pages.exceptions.error\_dialog import ErrorException

class Auth(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.size = QSize(SIZE['SMALLER'][0], SIZE['SMALLER'][1])

self.setFixedSize(self.size)

self.setWindowTitle('Login window')

self.login\_label = QLabel(self)

self.password\_label = QLabel(self)

self.login = QLineEdit(self)

self.button = QPushButton('Войти', self)

self.error\_input = ErrorException(text="Неверный логин или пароль", title="Неверный логин или пароль")

self.error\_user = ErrorException(text="Пользователь не найден!", title="Пользователь не найден")

self.draw()

def draw(self):

self.login\_label.setText('login:')

self.login\_label.move(20, 20)

self.login.move(80, 20)

self.login.resize(200, 32)

self.button.clicked.connect(self.on\_click)

self.button.resize(200, 32)

self.button.move(80, 120)

self.show()

def on\_click(self):

if self.login.text() == 'admin':

self.destroy()

a = AdminWindow()

self.window.append(a)

a.show()

else:

er = ErrorException(text="Роль не найдена", title="Ошибка!")

self.window.append(er)

er.show()

Файл error\_dialog.py:

from PySide6.QtCore import QSize

from PySide6.QtWidgets import QPushButton, QLabel, QMainWindow

class ErrorException(QMainWindow):

def \_\_init\_\_(self, title: str, text: str):

QMainWindow.\_\_init\_\_(self)

self.\_title = title

self.\_text = text

self.setMinimumSize(QSize(320, 140))

self.setWindowTitle('{}'.format(self.\_title))

label = QLabel(self)

label.setText('{}'.format(self.\_text))

label.move(20, 20)

label.resize(300, 20)

button = QPushButton('Ok', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 90)

def clickMetod(self):

self.close()

Файл database.py:

from decouple import config

from psycopg2 import connect

config = {

'DATABASE\_HOST': config('DATABASE\_HOST'),

'DATABASE\_NAME': config('DATABASE\_NAME'),

'DATABASE\_USER': config('DATABASE\_USER'),

'DATABASE\_PASSWORD': config('DATABASE\_PASSWORD'),

}

def connect\_to\_database():

return connect(

host=config['DATABASE\_HOST'],

database=config['DATABASE\_NAME'],

user=config['DATABASE\_USER'],

password=config['DATABASE\_PASSWORD']

)

Файл department.py:

from lib.database import connect\_to\_database

class DepartmentService:

def \_\_init\_\_(self):

pass

def \_\_repr\_\_(self):

return "".format()

def open\_connection(self):

connection = connect\_to\_database()

cursor = connection.cursor()

return connection, cursor

def create(self, id, name, phone):

print(id)

conn, cur = self.open\_connection()

cur.execute(

'INSERT INTO "department" (id, name, phone) VALUES (%s, %s, %s)',

(id, name, phone))

conn.commit()

cur.close()

conn.close()

def get(self):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "department"')

rows = cur.fetchall()

cur.close()

conn.close()

result = []

for row in rows:

data = {

"id": row[0],

"name": row[1],

"phone": row[2]

}

result.append(data)

try:

return result

except:

return None

def get\_by\_id(self, id):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "department" WHERE id=%s', (id,))

row = cur.fetchone()

cur.close()

conn.close()

try:

return {

"id": row[0],

"name": row[1],

"phone": row[2]

}

except:

return None

def update(self, id, name, phone) -> None:

conn, cur = self.open\_connection()

cur.execute(

'UPDATE "department" SET "name"=%s, "phone"=%s, WHERE id=%s',

(name, phone, id))

conn.commit()

cur.close()

conn.close()

def delete(self, id) -> None:

conn, cur = self.open\_connection()

cur.execute('DELETE FROM "department" WHERE id=%s', (id,))

conn.commit()

cur.close()

conn.close()

def count(self):

conn, cur = self.open\_connection()

cur.execute('SELECT MAX("id") FROM "department"')

data = cur.fetchone()

cur.close()

conn.close()

for i, v in enumerate(data):

data = v

break

if data is None:

return 0

return data

Файл student.py:

from lib.database import connect\_to\_database

class StudentService:

def \_\_init\_\_(self):

pass

def \_\_repr\_\_(self):

return "".format()

def open\_connection(self):

connection = connect\_to\_database()

cursor = connection.cursor()

return connection, cursor

def create(self, id, number\_of\_student\_book, full\_name, group\_name, city):

conn, cur = self.open\_connection()

cur.execute(

'INSERT INTO "student" (id, number\_of\_student\_book, full\_name, group\_name, city) VALUES (%s, %s, %s, %s, %s)',

(id, number\_of\_student\_book, full\_name, group\_name, city))

conn.commit()

cur.close()

conn.close()

def get(self):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "student"')

rows = cur.fetchall()

cur.close()

conn.close()

result = []

for row in rows:

data = {

"id": row[0],

"number\_book": row[1],

"full\_name": row[2],

"group\_name": row[3],

"city": row[4],

}

result.append(data)

try:

return result

except:

return None

def get\_by\_id(self, id):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "student" WHERE id=%s', (id,))

row = cur.fetchone()

cur.close()

conn.close()

try:

return {

"id": row[0],

"number\_book": row[1],

"full\_name": row[2],

"group\_name": row[3],

"city": row[4],

}

except:

return None

def update(self, id, number\_of\_student\_book, full\_name, group\_name, city) -> None:

conn, cur = self.open\_connection()

cur.execute(

'UPDATE "student" SET full\_name=%s, academic\_degree=%s, department\_id=%s, descipline\_id WHERE id=%s',

(number\_of\_student\_book, full\_name, group\_name, city, id))

conn.commit()

cur.close()

conn.close()

def delete(self, id) -> None:

conn, cur = self.open\_connection()

cur.execute('DELETE FROM "student" WHERE id=%s', (id,))

conn.commit()

cur.close()

conn.close()

def count(self):

conn, cur = self.open\_connection()

cur.execute(f'SELECT MAX(id) FROM "student"')

data = cur.fetchone()

cur.close()

conn.close()

for i, v in enumerate(data):

data = v

break

if data is None:

return 0

return data

Файл teacher.py:

from lib.database import connect\_to\_database

class TeacherService:

def \_\_init\_\_(self):

pass

def \_\_repr\_\_(self):

return "".format()

def open\_connection(self):

connection = connect\_to\_database()

cursor = connection.cursor()

return connection, cursor

def create(self, full\_name, academic\_degree, department\_id, discipline\_id):

teacher\_number = self.count() + 1

conn, cur = self.open\_connection()

cur.execute(

'INSERT INTO "teacher" (teacher\_number, full\_name, academic\_degree, department\_id, discipline\_id) VALUES (%s, %s, %s, %s, %s)',

(teacher\_number, full\_name, academic\_degree, department\_id, discipline\_id)

)

conn.commit()

cur.close()

conn.close()

def get(self):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "teacher"')

rows = cur.fetchall()

cur.close()

conn.close()

result = []

for row in rows:

data = {

"teacher\_number": row[0],

"full\_name": row[1],

"academic\_degree": row[2],

"department\_id": row[3],

"discipline\_id": row[4],

}

result.append(data)

try:

return result

except:

return None

def get\_by\_id(self, id):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "teacher" WHERE id=%s', (id,))

row = cur.fetchone()

cur.close()

conn.close()

try:

return {

"teacher\_number": row[0],

"full\_name": row[1],

"academic\_degree": row[2],

"department\_id": row[3],

"discipline\_id": row[4],

}

except:

return None

def update(self, name: str, degree, depart, discipline) -> None:

conn, cur = self.open\_connection()

cur.execute('UPDATE "teacher" SET full\_name=%s, academic\_degree=%s, department\_id=%s, descipline\_id WHERE id=%s',

(name, degree, depart, discipline, id))

conn.commit()

cur.close()

conn.close()

def delete(self, id) -> None:

conn, cur = self.open\_connection()

cur.execute('DELETE FROM "teacher" WHERE teacher\_number=%s', (id,))

conn.commit()

cur.close()

conn.close()

def count(self):

conn, cur = self.open\_connection()

cur.execute(f'SELECT COUNT(teacher\_number) FROM "teacher"')

data = cur.fetchone()

cur.close()

conn.close()

for i, v in enumerate(data):

data = v

break

if data is None:

return 0

return data

Файл discipline.py:

from lib.database import connect\_to\_database

class DisciplineService:

def \_\_init\_\_(self):

pass

def \_\_repr\_\_(self):

return "".format()

def open\_connection(self):

connection = connect\_to\_database()

cursor = connection.cursor()

return connection, cursor

def create(self, subject\_code, subject\_name, hours, student\_id):

conn, cur = self.open\_connection()

cur.execute(

'INSERT INTO "discipline" (subject\_code, subject\_name, hours, student\_id) VALUES (%s, %s, %s, %s)',

(subject\_code, subject\_name, hours, student\_id))

conn.commit()

cur.close()

conn.close()

def get(self):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "discipline"')

rows = cur.fetchall()

cur.close()

conn.close()

result = []

for row in rows:

data = {

"subject\_code": row[0],

"subject\_name": row[1],

"hours": row[2],

"student\_id": row[3]

}

result.append(data)

try:

return result

except:

return None

def get\_by\_id(self, id):

conn, cur = self.open\_connection()

cur.execute('SELECT \* FROM "discipline" WHERE subject\_code=%s', (id,))

row = cur.fetchone()

cur.close()

conn.close()

try:

return {

"subject\_code": row[0],

"subject\_name": row[1],

"hours": row[2],

"student\_id": row[3]

}

except:

return None

def update(self, id, subject\_name, hours, student\_id) -> None:

conn, cur = self.open\_connection()

cur.execute(

'UPDATE "discipline" SET subject\_name=%s, hours=%s, student\_id=%s WHERE id=%s',

(subject\_name, hours, student\_id, id))

conn.commit()

cur.close()

conn.close()

def delete(self, id) -> None:

conn, cur = self.open\_connection()

cur.execute('DELETE FROM "discipline" WHERE subject\_code=%s', (id,))

conn.commit()

cur.close()

conn.close()

def count(self):

conn, cur = self.open\_connection()

cur.execute(f'SELECT MAX(subject\_code) FROM "discipline"')

data = cur.fetchone()

cur.close()

conn.close()

for i, v in enumerate(data):

data = v

break

if data is None:

return 0

return data

Файл admin\_window.py:

from PySide6.QtWidgets import \*

from PySide6.QtGui import \*

from PySide6.QtCore import \*

from data.department import DepartmentService

from data.discipline import DisciplineService

from data.student import StudentService

from data.teacher import TeacherService

from pages.exceptions.error\_dialog import ErrorException

class AdminWindow(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.depart\_service = DepartmentService()

self.disp\_service = DisciplineService()

self.student\_service = StudentService()

self.teacher\_service = TeacherService()

data\_rows = []

departs = self.depart\_service.get()

for depart in departs:

data\_rows.append(

[depart['id'], depart['name'], depart['phone']])

self.size = QSize(940, 800)

self.setFixedSize(self.size)

self.setWindowTitle('Админская панель')

menuBar = QMenuBar(self)

self.setMenuBar(menuBar)

toolbar = QToolBar('My main toolbar')

toolbar.setIconSize(QSize(16, 16))

self.addToolBar(toolbar)

toolbar\_button\_create = QAction(QIcon('img/ui/plus.png'), 'Добавить', self)

toolbar\_button\_create.setStatusTip('добавить')

toolbar\_button\_create.setCheckable(True)

toolbar\_button\_create.triggered.connect(self.call\_create\_depart)

toolbar.addAction(toolbar\_button\_create)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_delete = QAction(QIcon('img/ui/minus.png'), 'Удалить', self)

toolbar\_button\_delete.setStatusTip('Удалить')

toolbar\_button\_delete.setCheckable(True)

toolbar\_button\_delete.triggered.connect(self.call\_form\_delete)

toolbar.addAction(toolbar\_button\_delete)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_discipline = QAction(QIcon('img/ui/books.png'), 'Дисциплины', self)

toolbar\_button\_discipline.setStatusTip('Дисциплины')

toolbar\_button\_discipline.setCheckable(True)

toolbar\_button\_discipline.triggered.connect(self.call\_discipline)

toolbar.addAction(toolbar\_button\_discipline)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_students = QAction(QIcon('img/ui/user.png'), 'Студенты', self)

toolbar\_button\_students.setStatusTip('Студенты')

toolbar\_button\_students.setCheckable(True)

toolbar\_button\_students.triggered.connect(self.call\_students)

toolbar.addAction(toolbar\_button\_students)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_teachers = QAction(QIcon('img/ui/book-alt.png'), 'Преподаватели', self)

toolbar\_button\_teachers.setStatusTip('Преподаватели')

toolbar\_button\_teachers.setCheckable(True)

toolbar\_button\_teachers.triggered.connect(self.call\_teachers)

toolbar.addAction(toolbar\_button\_teachers)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/refresh.png'), 'Обновить таблицу', self)

toolbar\_button\_update\_table.setStatusTip('Обновить таблицу')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.callUpdateTable)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

central = QWidget(self)

self.setCentralWidget(central)

self.grid = QGridLayout(self)

central.setLayout(self.grid)

self.table = QTableWidget(self)

self.table.setColumnCount(3)

self.table.setRowCount(15)

self.table.setHorizontalHeaderLabels(['ID', 'Имя', 'Телефон'])

departs = self.depart\_service.get()

self.table.setRowCount(50)

for i, v in enumerate(departs):

self.table.setItem(i, 0, QTableWidgetItem(str(v['id'])))

self.table.setItem(i, 1, QTableWidgetItem(str(v['name'])))

self.table.setItem(i, 2, QTableWidgetItem(str(v['phone'])))

self.grid.addWidget(self.table, 0, 0)

self.show()

def callUpdateTable(self, state):

if state:

self.close()

a = AdminWindow()

self.window.append(a)

a.show()

def call\_discipline(self, state):

if state:

a = Discipline()

self.window.append(a)

a.show()

def call\_students(self, state):

if state:

a = StudentsWindow()

self.window.append(a)

a.show()

def call\_create\_depart(self, state):

if state:

a = FormCreateDepart()

self.window.append(a)

a.show()

def call\_form\_delete(self, state):

if state:

a = FormDelete()

self.window.append(a)

a.show()

def call\_teachers(self, state):

if state:

a = Teachers()

self.window.append(a)

a.show()

class FormCreateDepart(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.depart\_service = DepartmentService()

self.setMinimumSize(QSize(320, 270))

self.setWindowTitle('Создание кафедры')

self.name\_label = QLabel(self)

self.name\_label.setText("Имя")

self.name\_label.move(20, 20)

self.name = QLineEdit(self)

self.name.move(110, 20)

self.name.resize(200, 32)

self.phone\_label = QLabel(self)

self.phone\_label.setText("Телефон")

self.phone\_label.move(20, 60)

self.phone = QLineEdit(self)

self.phone.move(110, 60)

self.phone.resize(200, 32)

button = QPushButton('Создать', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 225)

def clickMetod(self):

id = self.depart\_service.count() + 1

print(id)

name = self.name.text()

phone = self.phone.text()

self.depart\_service.create(id, name, phone)

self.close()

a = ErrorException(title='Успех', text='Студент создан')

self.window.append(a)

a.show()

class FormDelete(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.depart\_service = DepartmentService()

self.setMinimumSize(QSize(320, 180))

self.setWindowTitle('Удалить кафедру')

self.idLabel = QLabel(self)

self.idLabel.setText('ID:')

self.idLabel.move(20, 20)

self.idLine = QComboBox(self)

for item in self.depart\_service.get():

self.idLine.addItem(str(item['id']))

self.idLine.move(80, 20)

self.idLine.resize(200, 32)

button = QPushButton('Удалить', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 120)

def clickMetod(self):

self.depart\_service.delete(int(self.idLine.currentText()))

self.idLine.clear()

self.close()

a = ErrorException(title='Успех', text='Удалено')

self.window.append(a)

a.show()

class StudentsWindow(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.students\_service = StudentService()

data\_rows = []

students = self.students\_service.get()

for student in students:

data\_rows.append(

[student['id'], student['number\_book'], student['full\_name'], student['group\_name'], student['city']])

self.size = QSize(600, 750)

self.setFixedSize(self.size)

self.setWindowTitle('AdminWindow')

menuBar = QMenuBar(self)

self.setMenuBar(menuBar)

toolbar = QToolBar('My main toolbar')

toolbar.setIconSize(QSize(16, 16))

self.addToolBar(toolbar)

toolbar\_button\_create = QAction(QIcon('img/ui/plus.png'), 'Добавить студента', self)

toolbar\_button\_create.setStatusTip('Добавить студента')

toolbar\_button\_create.setCheckable(True)

toolbar\_button\_create.triggered.connect(self.create\_student)

toolbar.addAction(toolbar\_button\_create)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_create = QAction(QIcon('img/ui/minus.png'), 'Удалить студента', self)

toolbar\_button\_create.setStatusTip('Удалить студента')

toolbar\_button\_create.setCheckable(True)

toolbar\_button\_create.triggered.connect(self.delete\_student)

toolbar.addAction(toolbar\_button\_create)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/refresh.png'), 'Обновить таблицу', self)

toolbar\_button\_update\_table.setStatusTip('Обновить таблицу')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.callUpdateTable)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

central = QWidget(self)

self.setCentralWidget(central)

self.grid = QGridLayout(self)

central.setLayout(self.grid)

self.table = QTableWidget(self)

self.table.setColumnCount(5)

self.table.setRowCount(50)

self.table.setHorizontalHeaderLabels(['ID', 'Номер', 'ФИО', 'Группа', 'Город'])

for i, v in enumerate(self.students\_service.get()):

self.table.setItem(i, 0, QTableWidgetItem(str(v['id'])))

self.table.setItem(i, 1, QTableWidgetItem(str(v['number\_book'])))

self.table.setItem(i, 2, QTableWidgetItem(str(v['full\_name'])))

self.table.setItem(i, 3, QTableWidgetItem(str(v['group\_name'])))

self.table.setItem(i, 4, QTableWidgetItem(str(v['city'])))

self.grid.addWidget(self.table, 0, 0)

self.show()

def callUpdateTable(self, state):

if state:

self.close()

a = StudentsWindow()

self.window.append(a)

a.show()

def create\_student(self, state):

if state:

a = FormCreateStudent()

self.window.append(a)

a.show()

def delete\_student(self, state):

if state:

a = FormDeleteStudent()

self.window.append(a)

a.show()

class FormCreateStudent(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.student\_service = StudentService()

self.setMinimumSize(QSize(320, 270))

self.setWindowTitle('Создание пользователя')

self.stud\_number\_label = QLabel(self)

self.stud\_number\_label.setText("Номер студ")

self.stud\_number\_label.move(20, 20)

self.stud\_number = QLineEdit(self)

self.stud\_number.move(110, 20)

self.stud\_number.resize(200, 32)

self.fio\_label = QLabel(self)

self.fio\_label.setText("ФИО")

self.fio\_label.move(20, 60)

self.fio = QLineEdit(self)

self.fio.move(110, 60)

self.fio.resize(200, 32)

self.group = QLineEdit(self)

self.group.move(110, 125)

self.group.resize(200, 32)

self.group\_label = QLabel(self)

self.group\_label.setText("Группа")

self.group\_label.move(20, 125)

self.city = QLineEdit(self)

self.city.move(110, 160)

self.city.resize(200, 32)

self.line\_name\_label = QLabel(self)

self.line\_name\_label.setText("Город")

self.line\_name\_label.move(20, 160)

button = QPushButton('Создать', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 225)

def clickMetod(self):

id = self.student\_service.count() + 1

stud\_number = self.stud\_number.text()

fio = self.fio.text()

group = self.group.text()

city = self.city.text()

self.student\_service.create(id, stud\_number, fio, group, city)

self.close()

a = ErrorException(title='Успех', text='Студент создан')

self.window.append(a)

a.show()

class FormDeleteTeacher(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.teacher\_service = TeacherService()

self.setMinimumSize(QSize(320, 180))

self.setWindowTitle('Удалить преподавателя')

self.idLabel = QLabel(self)

self.idLabel.setText('ID:')

self.idLabel.move(20, 20)

self.idLine = QComboBox(self)

for item in self.teacher\_service.get():

self.idLine.addItem(str(item['teacher\_number']))

self.idLine.move(80, 20)

self.idLine.resize(200, 32)

button = QPushButton('Удалить', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 120)

def clickMetod(self):

self.teacher\_service.delete(int(self.idLine.currentText()))

self.idLine.clear()

self.close()

a = ErrorException(title='Успех', text='Удалено')

self.window.append(a)

a.show()

class FormDeleteStudent(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.student\_service = StudentService()

self.setMinimumSize(QSize(320, 180))

self.setWindowTitle('Удалить студента')

self.idLabel = QLabel(self)

self.idLabel.setText('ID:')

self.idLabel.move(20, 20)

self.idLine = QComboBox(self)

for item in self.student\_service.get():

self.idLine.addItem(str(item['id']))

self.idLine.move(80, 20)

self.idLine.resize(200, 32)

button = QPushButton('Удалить', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 120)

def clickMetod(self):

self.student\_service.delete(int(self.idLine.currentText()))

self.idLine.clear()

self.close()

a = ErrorException(title='Успех', text='Удалено')

self.window.append(a)

a.show()

class Teachers(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.teachers\_service = TeacherService()

data\_rows = []

teachers = self.teachers\_service.get()

for teacher in teachers:

data\_rows.append(

[teacher['teacher\_number'], teacher['full\_name'], teacher['academic\_degree'], teacher['department\_id'],

teacher['discipline\_id']])

self.size = QSize(600, 750)

self.setFixedSize(self.size)

self.setWindowTitle('Преподаватели')

menuBar = QMenuBar(self)

self.setMenuBar(menuBar)

toolbar = QToolBar('My main toolbar')

toolbar.setIconSize(QSize(16, 16))

self.addToolBar(toolbar)

toolbar\_button\_update\_table = QAction(QIcon('img/ui/plus.png'), 'Создать учителя', self)

toolbar\_button\_update\_table.setStatusTip('Создать учителя')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.call\_create\_teacher)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/minus.png'), 'Удалить преподавателя', self)

toolbar\_button\_update\_table.setStatusTip('Удалить преподавателя')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.call\_delete\_teacher)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/refresh.png'), 'Обновить таблицу', self)

toolbar\_button\_update\_table.setStatusTip('Обновить таблицу')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.callUpdateTable)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

central = QWidget(self)

self.setCentralWidget(central)

self.grid = QGridLayout(self)

central.setLayout(self.grid)

self.table = QTableWidget(self)

self.table.setColumnCount(5)

self.table.setRowCount(50)

self.table.setHorizontalHeaderLabels(['ID', 'ФИО', 'Степень', 'ID кафедры', 'ID дисциплины'])

for i, v in enumerate(self.teachers\_service.get()):

print(v)

self.table.setItem(i, 0, QTableWidgetItem(str(v['teacher\_number'])))

self.table.setItem(i, 1, QTableWidgetItem(str(v['full\_name'])))

self.table.setItem(i, 2, QTableWidgetItem(str(v['academic\_degree'])))

self.table.setItem(i, 3, QTableWidgetItem(str(v['department\_id'])))

self.table.setItem(i, 4, QTableWidgetItem(str(v['discipline\_id'])))

self.grid.addWidget(self.table, 0, 0)

self.show()

def callUpdateTable(self, state):

if state:

self.close()

a = Teachers()

self.window.append(a)

a.show()

def call\_create\_teacher(self, state):

if state:

a = FormCreateTeacher()

self.window.append(a)

a.show()

def call\_delete\_teacher(self, state):

if state:

a = FormDeleteTeacher()

self.window.append(a)

a.show()

class FormCreateTeacher(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.teachers\_service = TeacherService()

self.depart\_service = DepartmentService()

self.discipline\_service = DisciplineService()

self.setMinimumSize(QSize(320, 270))

self.setWindowTitle('Создание преподавателя')

self.fio\_label = QLabel(self)

self.fio\_label.setText("Фио")

self.fio\_label.move(20, 20)

self.fio = QLineEdit(self)

self.fio.move(110, 20)

self.fio.resize(200, 32)

self.degree\_label = QLabel(self)

self.degree\_label.setText("Степень")

self.degree\_label.move(20, 60)

self.degree = QLineEdit(self)

self.degree.move(110, 60)

self.degree.resize(200, 32)

self.depart = QComboBox(self)

for teacher in self.depart\_service.get():

self.depart.addItem(str(teacher['id']))

self.depart.move(110, 125)

self.depart.resize(200, 32)

self.combo\_box\_label\_end = QLabel(self)

self.combo\_box\_label\_end.setText("ID кафедры")

self.combo\_box\_label\_end.move(20, 125)

self.combo\_box\_label\_user = QLabel(self)

self.combo\_box\_label\_user.setText("ID дисциплины")

self.combo\_box\_label\_user.move(20, 160)

self.combo\_discipline = QComboBox(self)

for discipline in self.discipline\_service.get():

self.combo\_discipline.addItem(str(discipline['subject\_code']))

self.combo\_discipline.move(110, 160)

self.combo\_discipline.resize(200, 32)

button = QPushButton('Создать', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 225)

def clickMetod(self):

fio = self.fio.text()

degree = self.degree.text()

depart = self.depart.currentText()

disc = self.combo\_discipline.currentText()

self.teachers\_service.create(fio, degree, depart, disc)

self.close()

a = ErrorException(title='Успех', text='Создано')

self.window.append(a)

a.show()

class Discipline(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.discipline\_service = DisciplineService()

data\_rows = []

discipline = self.discipline\_service.get()

for dis in discipline:

data\_rows.append(

[dis['subject\_code'], dis['subject\_name'], dis['hours'], dis['student\_id']])

self.size = QSize(600, 750)

self.setFixedSize(self.size)

self.setWindowTitle('Дисциплины')

menuBar = QMenuBar(self)

self.setMenuBar(menuBar)

toolbar = QToolBar('My main toolbar')

toolbar.setIconSize(QSize(16, 16))

self.addToolBar(toolbar)

toolbar\_button\_update\_table = QAction(QIcon('img/ui/plus.png'), 'Создать дисциплину', self)

toolbar\_button\_update\_table.setStatusTip('Создать дисциплину')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.call\_create\_discipline)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/minus.png'), 'Удалить дисциплину', self)

toolbar\_button\_update\_table.setStatusTip('Удалить дисциплину')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.call\_delete\_discipline)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

toolbar\_button\_update\_table = QAction(QIcon('img/ui/refresh.png'), 'Обновить таблицу', self)

toolbar\_button\_update\_table.setStatusTip('Обновить таблицу')

toolbar\_button\_update\_table.setCheckable(True)

toolbar\_button\_update\_table.triggered.connect(self.callUpdateTable)

toolbar.addAction(toolbar\_button\_update\_table)

self.setStatusBar(QStatusBar(self))

toolbar.addSeparator()

central = QWidget(self)

self.setCentralWidget(central)

self.grid = QGridLayout(self)

central.setLayout(self.grid)

self.table = QTableWidget(self)

self.table.setColumnCount(4)

self.table.setRowCount(50)

self.table.setHorizontalHeaderLabels(['Код дисциплины', 'Наименование', 'Часы', 'ID студента'])

for i, v in enumerate(self.discipline\_service.get()):

self.table.setItem(i, 0, QTableWidgetItem(str(v['subject\_code'])))

self.table.setItem(i, 1, QTableWidgetItem(str(v['subject\_name'])))

self.table.setItem(i, 2, QTableWidgetItem(str(v['hours'])))

self.table.setItem(i, 3, QTableWidgetItem(str(v['student\_id'])))

self.grid.addWidget(self.table, 0, 0)

self.show()

def callUpdateTable(self, state):

if state:

self.close()

a = Discipline()

self.window.append(a)

a.show()

def call\_create\_discipline(self, state):

if state:

a = FormCreateDiscipline()

self.window.append(a)

a.show()

def call\_delete\_discipline(self, state):

if state:

a = FormDeleteDiscipline()

self.window.append(a)

a.show()

class FormCreateDiscipline(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.dis\_service = DisciplineService()

self.student\_service = StudentService()

self.setMinimumSize(QSize(320, 270))

self.setWindowTitle('Создание дисциплины')

self.name\_label = QLabel(self)

self.name\_label.setText("Наименование")

self.name\_label.move(20, 20)

self.name = QLineEdit(self)

self.name.move(110, 20)

self.name.resize(200, 32)

self.hours\_label = QLabel(self)

self.hours\_label.setText("Часы")

self.hours\_label.move(20, 60)

self.hours = QLineEdit(self)

self.hours.move(110, 60)

self.hours.resize(200, 32)

self.combo\_box\_label\_user = QLabel(self)

self.combo\_box\_label\_user.setText("ID студента")

self.combo\_box\_label\_user.move(20, 125)

self.combo\_discipline = QComboBox(self)

for student in self.student\_service.get():

self.combo\_discipline.addItem(str(student['id']))

self.combo\_discipline.move(110, 125)

self.combo\_discipline.resize(200, 32)

button = QPushButton('Создать', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 225)

def clickMetod(self):

id = self.dis\_service.count() + 1

name = self.name.text()

hours = self.hours.text()

student = self.combo\_discipline.currentText()

self.dis\_service.create(id, name, hours, student)

self.close()

a = ErrorException(title='Успех', text='Дисциплина создана')

self.window.append(a)

a.show()

class FormDeleteDiscipline(QMainWindow):

def \_\_init\_\_(self):

self.window = []

QMainWindow.\_\_init\_\_(self)

self.dis = DisciplineService()

self.setMinimumSize(QSize(320, 180))

self.setWindowTitle('Удалить дисциплину')

self.idLabel = QLabel(self)

self.idLabel.setText('ID:')

self.idLabel.move(20, 20)

self.idLine = QComboBox(self)

for item in self.dis.get():

self.idLine.addItem(str(item['subject\_code']))

self.idLine.move(80, 20)

self.idLine.resize(200, 32)

button = QPushButton('Удалить', self)

button.clicked.connect(self.clickMetod)

button.resize(200, 32)

button.move(80, 120)

def clickMetod(self):

self.dis.delete(int(self.idLine.currentText()))

self.idLine.clear()

self.close()

a = ErrorException(title='Успех', text='Удалено')

self.window.append(a)

a.show()