Методические указания

по лабораторной работе №4

на тему: «Реализация сервиса для взаимодействия с разработанной базой данных»

по дисциплине: Информационная безопасность баз данных

Санкт-Петербург

2024

Цель работы

Получение навыков использования серверных языков программирования, фреймворков по работе с БД, ORM-систем

Задание

1. На основе БД, созданной в предыдущих лабораторных работах, необходимо создать сервис, который взаимодействует с разработанной БД. В качестве сервиса среди прочего могут быть разработаны:

- веб-интерфейс;

- desktop приложение;

- API;

- сервисы предобработки данных, которые заполняют базу данных (обработка websocket соединений, парсинг страниц, json и др)

- сервисы для защиты данных при их обработке, записи, считывании из/в БД (шифрование, обезличивание, сетевая защита и др.)

(*список не исчерпывает все возможные сервисы, см. п.4 в примечании*)

Дайте краткое обоснование выбранной среде разработки, языку и фреймворку, который вы используете для взаимодействия с базой данных.

2. В рамках ЛР опишите каким образом вы взаимодействуете с базой данных. С помощью каких фреймворков или языков. Продемонстрируйте несколько примеров по выборке, вставке, удалении данных из вашей БД с помощью выбранного вами фреймворка или языка программирования. Проконтролируйте корректность работы систем логирования и разграничения доступа, которые реализованы в 1-3 ЛР.

3. Для реализованного сервиса укажите кратко его основные функции, назначение, кратко опишите структуру. Дополнительно опишите и оцените методы защиты данных, реализованные вами и имеющиеся в наличии в разработанном сервисе. Выделите аналоги, недостатки и преимущества решений, которые вы реализовали.

Примечание

1. При составлении сложных запросов подбирайте отношения, подходящие для роли пользователя.
2. В отчете необходимо приводить полный листинг кода пользовательского интерфейса, достаточного для запуска исходников. Если для сборки/запуска необходимо наличие определенной среды/дополнительных инструментов, кратко их опишите.
3. В отчете необходимо кратко описать назначение и алгоритм всех разработанных модулей пользовательского интерфейса.
4. Для четвертой лабораторной работы могут быть рассмотрены альтернативы, например, составление не веб-интерфейса (как это показано в примере отчета), а составление API с набором методов, которые обращаются к БД или вместо интерфейса можно реализовать слой, который занимается предобработкой данных, перед тем как эта информация поступает в БД. В качестве альтернативы данной лабораторной работы может быть рассмотрены другие программные реализации для окружения разработанной БД удовлетворяющие следующим требования:
   1. Программная реализация каким-то образом взаимодействует с разработанной БД и ее системой защиты (потребляет данные или наоборот обновляет/генерирует данные для БД);
   2. Для разработанной программной реализации также необходимо рассмотреть вопросы информационной безопасности. Например, если вы рассматриваете создание API над БД, то можно включить в отчет информацию об используемых вами алгоритмах аутентификации (например, из oauth2 и пр.)

Пример отчета по лабораторной работе №4.

В качестве сервиса, взаимодействующего с БД, решено разработать многопользовательский интерфейс для взаимодействия с базой данных

**Обоснование выбора среды программирования и фреймворка**. В качестве языка программирования для реализации интерфейса был выбран Python, среда программирования PyCharm. Устанавливаем среду программирования (например, с официального сайта https://www.jetbrains.com/ru-ru/pycharm/download/?section=windows). Установим дополнительно следующие пакеты и фреймворки с помощью будет выполняться взаимодействие с базой данных, нами были выбраны:

*pip install sqlalchemy* //ORM для python

*pip install psycopg2* //драйвер для доступа в postgres DB

*pip install Flask* // фреймворк, который позволит реализовать простой веб-интерфейс.

Установим все необходимые зависимости в программное окружение. Продемонстрируем работу ORM *sqlalchemy.*Для начала работы с классами ORM нужно выполнить маппинг таблиц БД в структуру классов. Маппинг и подключение к БД представлены ниже в теле модуля «orm\_test.py»:

from sqlalchemy import Column, Integer, Text, ForeignKey  
from sqlalchemy import create\_engine  
from sqlalchemy.orm import declarative\_base  
from sqlalchemy.orm import sessionmaker  
  
Base = declarative\_base()  
  
  
# маппинг структуры таблиц БД в структуру классов ORM  
class Teachers(Base):  
 \_\_tablename\_\_ = 'teachers'  
  
 teacher\_id = Column(Integer(), primary\_key=True)  
 teacher\_full\_name = Column(Text)  
 department = Column(Text)  
 email = Column(Text)  
 phone = Column(Text)  
 consultation\_time = Column(Text)  
 working\_bulding = Column(Text)  
  
  
class Subject(Base):  
 \_\_tablename\_\_ = 'subject'  
  
 subject\_id = Column(Integer, primary\_key=True)  
 subject\_name = Column(Text)  
 subject\_type = Column(Text)  
 teacher\_id = Column(Integer, ForeignKey('teachers.teacher\_id'))  
 group\_id = Column(Integer, ForeignKey('groups.group\_id'))  
  
  
class Speciality(Base):  
 \_\_tablename\_\_ = "specialty"  
  
 specialty\_code = Column(Integer, primary\_key=True)  
 specialty\_name = Column(Text)  
 department = Column(Text)  
  
  
class SecretData(Base):  
 \_\_tablename\_\_ = 'secret\_data'  
  
 ID = Column(Integer, primary\_key=True)  
 username = Column(Text)  
 secret\_token = Column(Text)  
  
  
class MainLog(Base):  
 \_\_tablename\_\_ = 'main\_log'  
  
 log\_item\_id = Column(Integer, primary\_key=True)  
 operation\_type = Column(Text)  
 operation\_date = Column(Text)  
 user\_operator = Column(Text)  
 changed\_data = Column(Text)  
  
  
class Groups(Base):  
 \_\_tablename\_\_ = 'groups'  
  
 group\_id = Column(Integer, primary\_key=True)  
 course\_number = Column(Integer)  
 admin\_full\_name = Column(Text)  
 admin\_contacts = Column(Text)  
 specialty\_code = Column(Integer, ForeignKey('specialty.specialty\_code'))  
  
  
class Classes(Base):  
 \_\_tablename\_\_ = 'classes'  
  
 classes\_id = Column(Integer, primary\_key=True)  
 classes\_start\_time = Column(Text)  
 classes\_end\_time = Column(Text)  
 classroom = Column(Text)  
 building = Column(Text)  
 subject\_id = Column(Integer, ForeignKey('subject.subject\_id'))  
  
  
class CompactSchedule(Base):  
 \_\_tablename\_\_ = 'compact\_shedule'  
  
 id\_schel = Column(Integer, primary\_key=True)  
 subject\_name = Column(Text)  
 classes\_start\_time = Column(Text)  
 classes\_end\_time = Column(Text)  
 building = Column(Text)  
 teacher\_full\_name = Column(Text)  
  
  
class Consultations(Base):  
 \_\_tablename\_\_ = 'consultations'  
  
 id\_consult = Column(Integer, primary\_key=True)  
 subject\_name = Column(Text)  
 teacher\_full\_name = Column(Text)  
 department = Column(Text)  
 email = Column(Text)  
 phone = Column(Text)  
 consultation\_time = Column(Text)  
 working\_bulding = Column(Text)  
  
  
class SpecialityContacts(Base):  
 \_\_tablename\_\_ = 'speciality\_and\_contacts'  
  
 id\_contact = Column(Integer, primary\_key=True)  
 specialty\_name = Column(Text)  
 group\_id = Column(Integer)  
 course\_number = Column(Integer)  
 admin\_full\_name = Column(Text)  
 admin\_contacts = Column(Text)  
  
  
class SpecialityTeachers(Base):  
 \_\_tablename\_\_='speciality\_and\_teachers'  
  
 id\_sat = Column(Integer, primary\_key=True)  
 specialty\_name = Column(Text)  
 department = Column(Text)  
 group\_id = Column(Integer)  
 subject\_name = Column(Text)  
 teacher\_full\_name = Column(Text)  
  
  
# непосредственно подключение к базе данных  
# psw пароль индивидуальной учетки ("admin" для учетки petya\_admin,  
# "roman" для учетки Romanov\_student, "kozlov" для учетки kozlov\_teacher)  
psw = "admin"  
db = "dbs24"  
  
# вместо petya\_admin подставляем любую индивидуальную роль из БД  
engine = create\_engine('postgresql+psycopg2://petya\_admin:' + psw + '@localhost/' + db)  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
  
db = SessionLocal()

Выполним несколько запросов к БД от хотя бы двух различных пользователей. При удачном запросе, выводим содержимое на экран, при отказах доступа показываем ошибки.

**Запрос 1 от пользователя «Romanov\_student» к представлению SpecialityTeachers .**

Скрипт для выполнения запроса:

query\_example = db.query(SpecialityTeachers).all()  
for row in query\_example:  
 print(row.id\_sat, row.specialty\_name, row.group\_id, row.subject\_name)

Результат:



**Запрос 2 от пользователя «Romanov\_student» к таблицу Teachers.**

query\_example = db.query(Teachers).all()  
for row in query\_example:  
 print(row.teacher\_id, row.teacher\_full\_name, row.consultation\_time, row.email)

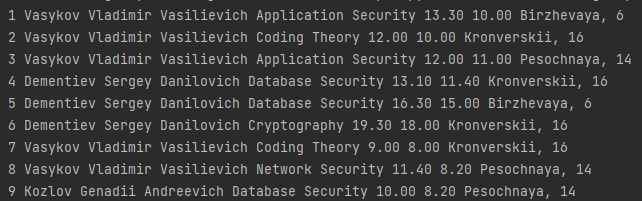
Результат:



**Запрос 3 от пользователя «Romanov\_student» к представлению.**

query\_example = db.query(CompactSchedule).all()  
for row in query\_example:  
 print(row.id\_schel, row.teacher\_full\_name, row.subject\_name, row.classes\_end\_time, row.classes\_start\_time, row.building)

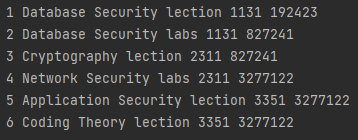
Результат:



**Запрос 4 от пользователя «petya\_admin» к таблице Subject**

query\_example = db.query(Subject).all()  
for row in query\_example:  
 print(row.subject\_id, row.subject\_name, row.subject\_type, row.group\_id, row.teacher\_id)

Результат:

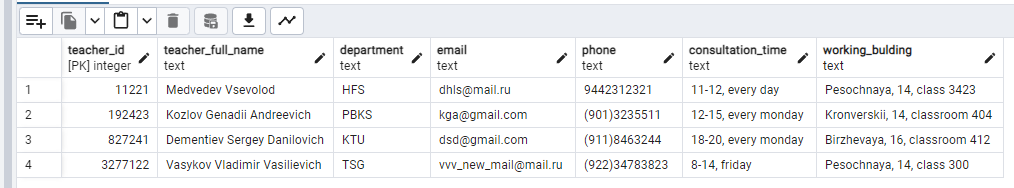


Покажем также примеры выполнения запросов на вставку/удаление данных с помощью ORM. Поскольку у созданных груповых ролей нет прав на редактирование информации БД, выполним эти запросы со стороны учетной записи postgres.

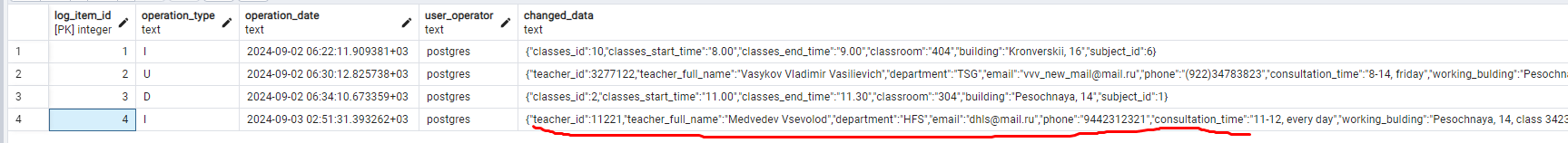
**Запрос 5. Вставим информацию о новом преподавателе.**

newTeacher = Teachers(  
 teacher\_id=11221,  
 teacher\_full\_name='Medvedev Vsevolod',  
 department='HFS',  
 email='dhls@mail.ru',  
 phone='9442312321',  
 consultation\_time='11-12, every day',  
 working\_bulding='Pesochnaya, 14, class 3423'  
)  
db.add(newTeacher)  
db.commit()

Результат:



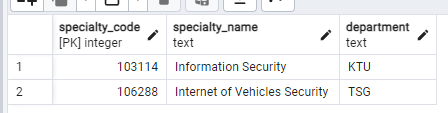
Проверяем работу системы мониторинга для запроса 4.



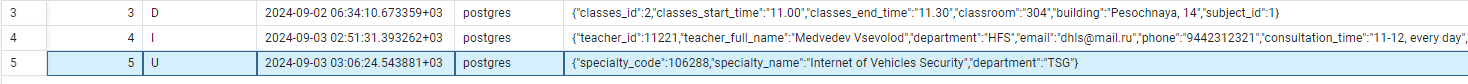
**Запрос 6. Изменим один из атрибутов в таблице Speciality**

target\_spec = db.query(Speciality).filter\_by(specialty\_name='IoT Security').first()  
target\_spec.specialty\_name = 'Internet of Vehicles Security'  
db.commit()

Результат



Проверяем работу лога:



В качестве основы разрабатываемого сервиса будет использована упрощенная схема взаимодействия с информацией из БД. Для созданных ранее таблиц в СУБД сформированы статические html страницы с помощью следующего python скрипта:

import pandas as pd

# функция формирует html код на основании выборки, отправляемой на вход данной функции

def sql\_to\_html\_func(query\_result, html\_page\_name):

df = pd.DataFrame()

table\_header = list(vars(query\_result[0]).keys())

table\_header.pop(0)

header\_frame = pd.DataFrame(table\_header).T

df = pd.concat([df, header\_frame])

for row\_class in query\_result:

row\_table = list(vars(row\_class).values())

row\_table.pop(0) # PK

df2 = pd.DataFrame(row\_table).T

df = pd.concat([df, df2])

df.to\_html('templates/'+html\_page\_name+'.html')

К этим страницам будет осуществляться многопользовательский доступ. Реализуем доступ к сгенерированным страницам с помощью flask\_login, добавим метод для регистрации новых пользователей. Опишем ограничения для доступа к каждой из таблиц. Исходный код для решения вышеописанных задач ниже:

from flask import Flask, render\_template, request, url\_for, redirect

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, UserMixin, login\_user, logout\_user

import enum

from sqlalchemy import Enum

psw = "qwerty"

db\_name = "dbs24"

app = Flask(\_\_name\_\_)

app.config["SQLALCHEMY\_DATABASE\_URI"] = 'postgresql+psycopg2://postgres:' + psw + '@localhost/' + db\_name

app.config["SECRET\_KEY"] = "abc"

app.debug = True

db = SQLAlchemy()

login\_manager = LoginManager()

login\_manager.init\_app(app)

class MyRoles(enum.Enum):

student = "student"

teacher = "teacher"

admin = "admin"

class Users(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(250), unique=True, nullable=False)

password = db.Column(db.String(250), nullable=False)

role = db.Column(Enum(MyRoles))

db.init\_app(app)

roles\_distr = {

"student": ['compact\_scheldule', 'consultations'],

"teacher": ['speciality\_and\_teachers', 'speciality\_and\_contacts'],

"admin": [

'classes', 'groups', 'main\_log', 'secret\_data', 'speciality', 'subjects', 'teachers',

'compact\_scheldule', 'consultations', 'speciality\_and\_teachers', 'speciality\_and\_contacts'

]

}

with app.app\_context():

db.create\_all()

@login\_manager.user\_loader

def loader\_user(user\_id):

return Users.query.get(user\_id)

@app.route('/register', methods=["GET", "POST"])

def register():

if request.method == "POST":

user = Users(

username=request.form.get("username"),

password=request.form.get("password"),

role=request.form.get("role")

)

db.session.add(user)

db.session.commit()

return redirect(url\_for("login"))

return render\_template("sign\_up.html")

@app.route("/login", methods=["GET", "POST"])

def login():

if request.method == "POST":

user = Users.query.filter\_by(

username=request.form.get("username")).first()

if user.password == request.form.get("password"):

login\_user(user)

return redirect(url\_for("home"))

return render\_template("login.html")

@app.route("/logout")

def logout():

logout\_user()

return redirect(url\_for("home"))

@app.route("/")

def home():

return render\_template("home.html", roles\_distr=roles\_distr)

@app.route("/subjects")

def subjects():

return render\_template("subjects.html")

@app.route("/teachers")

def teachers():

return render\_template("teachers.html")

@app.route("/classes")

def classes():

return render\_template("classes.html")

@app.route("/groups")

def groups():

return render\_template("groups.html")

@app.route("/main\_log")

def main\_log():

return render\_template("main\_log.html")

@app.route("/secret\_data")

def secret\_data():

return render\_template("secret\_data.html")

@app.route("/speciality")

def speciality():

return render\_template("speciality.html")

@app.route("/compact\_scheldule")

def compact\_scheldule():

return render\_template("compact\_scheldule.html")

@app.route("/consultations")

def consultations():

return render\_template("consultations.html")

@app.route("/speciality\_and\_teachers")

def speciality\_and\_teachers():

return render\_template("speciality\_and\_teachers.html")

@app.route("/speciality\_and\_contacts")

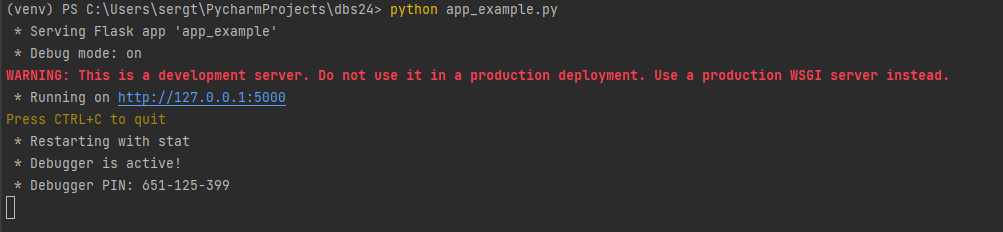
def speciality\_and\_contacts():

return render\_template("speciality\_and\_contacts.html")

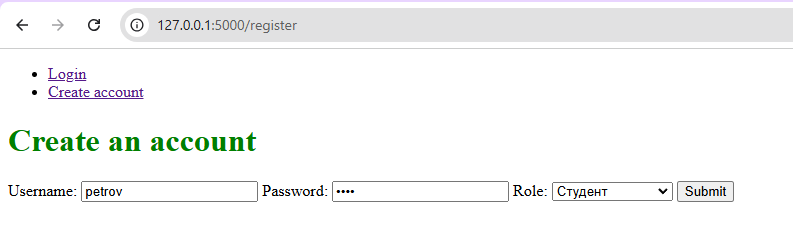
if \_\_name\_\_ == "\_\_main\_\_":

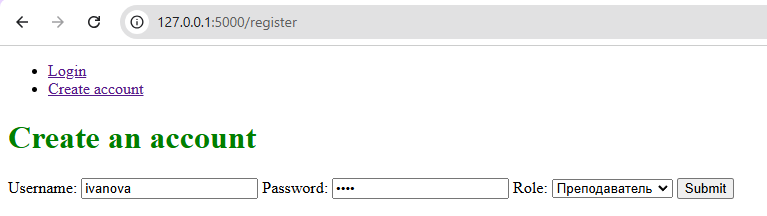
app.run()

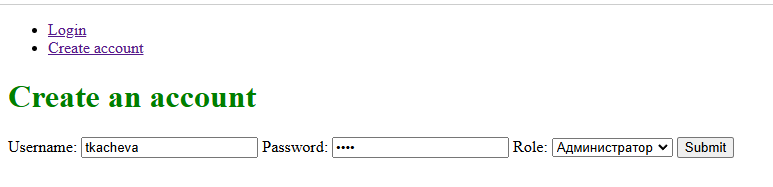
Запустим созданный сервис



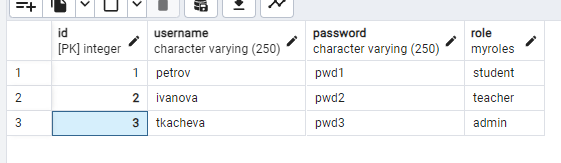
Зарегистрируем трех новых пользователей для каждой из ролей:





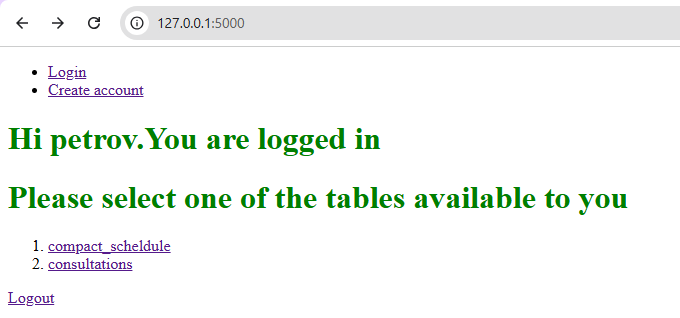


Итоговая таблица с новыми пользователями из СУБД:

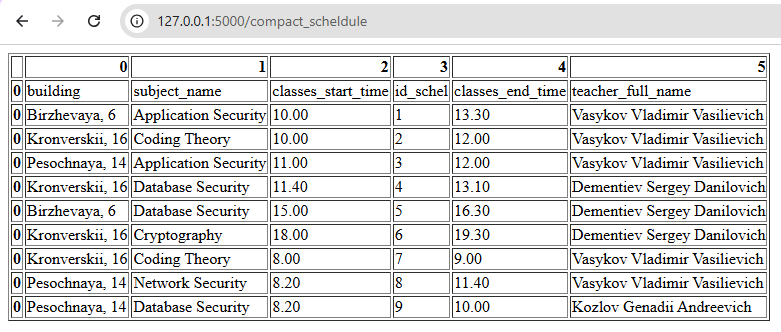


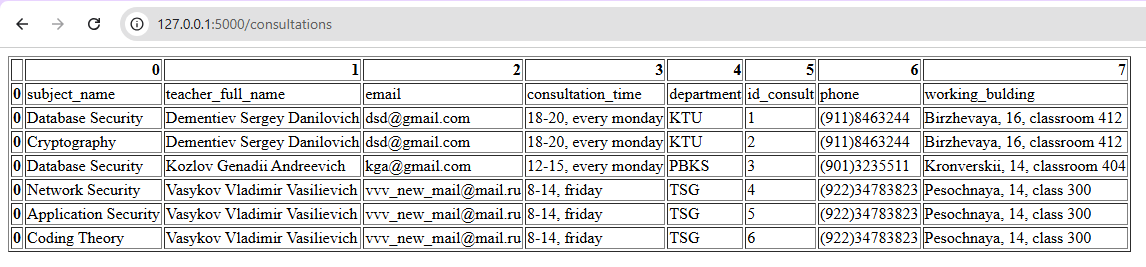
Продемонстрируем доступ со стороны каждой из ролей.

Интерфейс студента

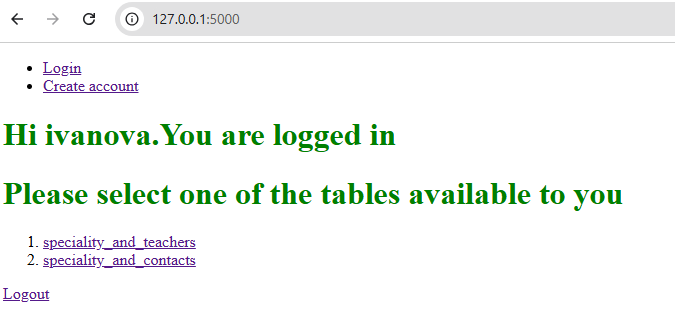


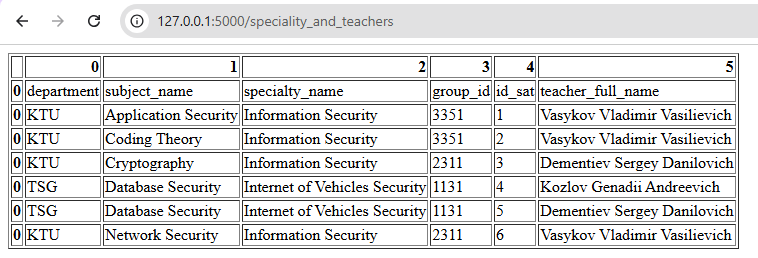
Доступные таблицы

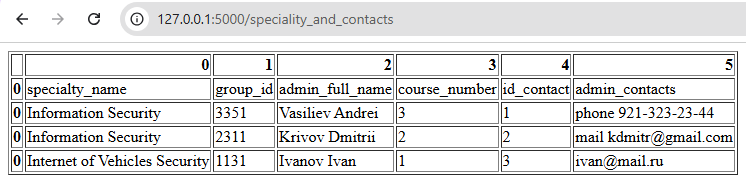




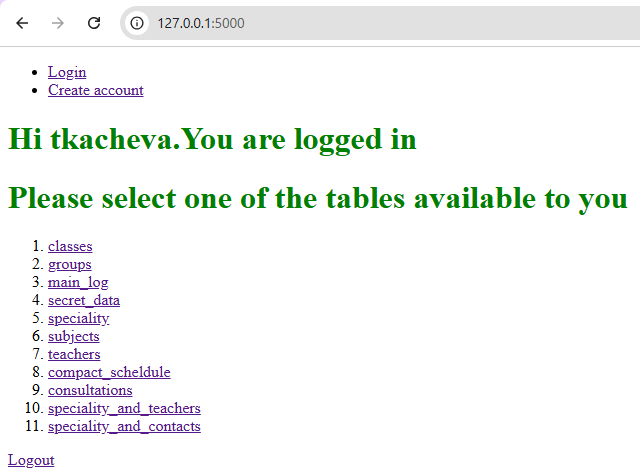
Интерфейс преподавателя:



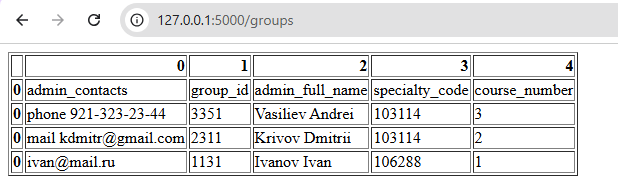


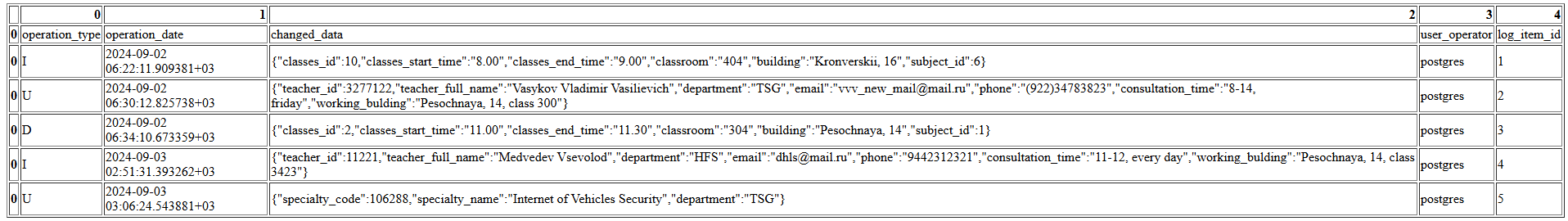


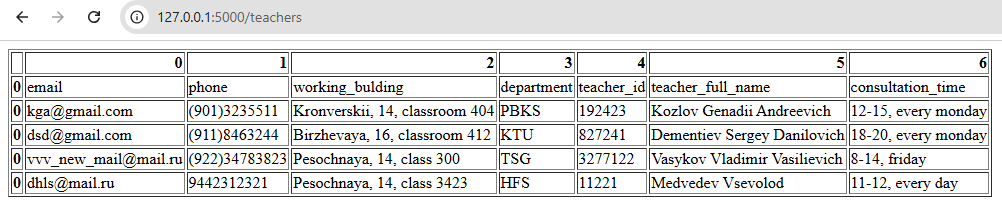
Интерфейс администратора:



Выведем выборочно несколько таблиц, доступных администратору:







Резервную копию БД, а также полные исходники разработанного сервиса и указанных в ЛР скриптов можно найти в приложении <https://drive.google.com/drive/folders/1t8juZJKcoRopHDSzBlHI50O2-Km5W5BH?usp=sharing>

Выделим недостатки разработанного приложения с точки зрения информационной безопасности:

* разработанный алгоритм доступа не использует систему разграничения доступа и определенные в СУБД роли, а только дублирует ее в коде сервиса. В связи с этим при изменении системы разграничения доступа в СУБД, данные изменения необходимо учитывать в самом сервисе причем с изменением исходного кода.
* сервис не поддерживает выгрузку обновленных данных, добавление, изменение и удаление данных в/из субд. Для выгрузки обновленных данных необходим повторный или регулярный запуск скрипта sql\_to\_html.py из полных исходников
* все данные пользователей в том числе пароли хранятся полностью в открытом виде без шифрования и хэширования