Отчет по второй промежуточной аттестации

В течение второй аттестации участниками группы была проделана следующая работа:

**Кудрявцева Настя:**

* Диаграмма активности
* Диаграмма состояний для роли “тренер”
* Диаграмма состояний для роли “клиент”
* ER диаграмма
* Составлена и заполнена база данных для проекта
* Добавлен рабочий механизм авторизации пользователей на сайте и выхода с сайта (прописана flask форма авторизации, написан шаблон login.html, добавлена логика входа/выхода пользователей, так же реализована система сессий)
* Модуль вывода списка тренировок для тренера

**Малюгин Алексей:**

* Диаграмма развертывания
* Диаграмма прецедентов (USE CASE)
* Физическая диаграмма для создания БД
* Написана общая логика запуска WEB сайта, подключения к БД, инициализации основных модулей
* ORM модели для работы с БД
* Реализованы модули регистрации клиента и тренера
* Мелкие наработки по будущему сайту (например, отображение в меню тех пунктов, которые относятся к текущему пользователю)

**Стацура Диана:**

* Диаграмма классов
* Диаграмма последовательности
* Диаграмма объектов
* ТЗ приведено в нормальный вид, исправлены ошибки
* Реализованы модули добавления тренировок клиенту и вывода статуса абонемента клиента.

Далее представлен подробный отчет по проделанной работе:

Оглавление

[Отчет по второй промежуточной аттестации 1](#_Toc8056654)

[Диаграмма активности 3](#_Toc8056655)

[Диаграммы состояний 4](#_Toc8056656)

[Диаграмма развертывания 4](#_Toc8056657)

[ER диаграмма 5](#_Toc8056658)

[USE CASE диаграмма 6](#_Toc8056659)

[Физическая диаграмма 7](#_Toc8056660)

[Диаграмма классов 8](#_Toc8056661)

[Диаграмма объектов 9](#_Toc8056662)

[Диаграмма последовательности 9](#_Toc8056663)

[ORM модели 10](#_Toc8056664)

[Модуль регистрации клиента и тренера 12](#_Toc8056665)

[Модуль авторизации пользователей: 13](#_Toc8056666)

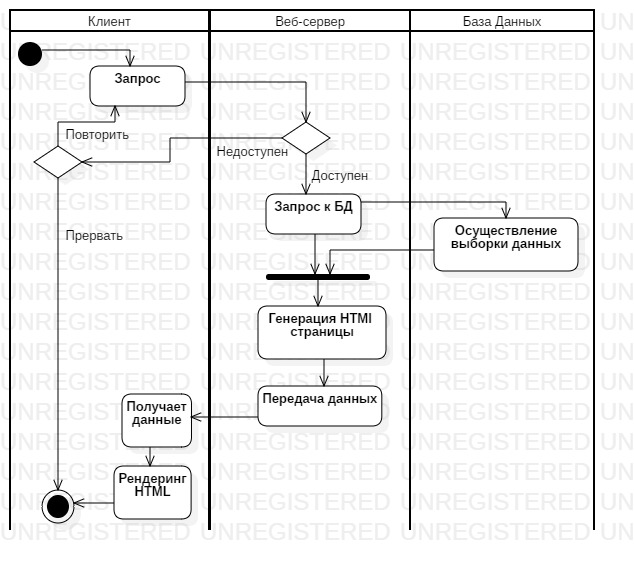
[Модуль вывода списка тренировок 14](#_Toc8056667)

[Модуль добавления тренировок клиенту 14](#_Toc8056668)

[Модуль просмотра статуса абонемента клиента 15](#_Toc8056669)

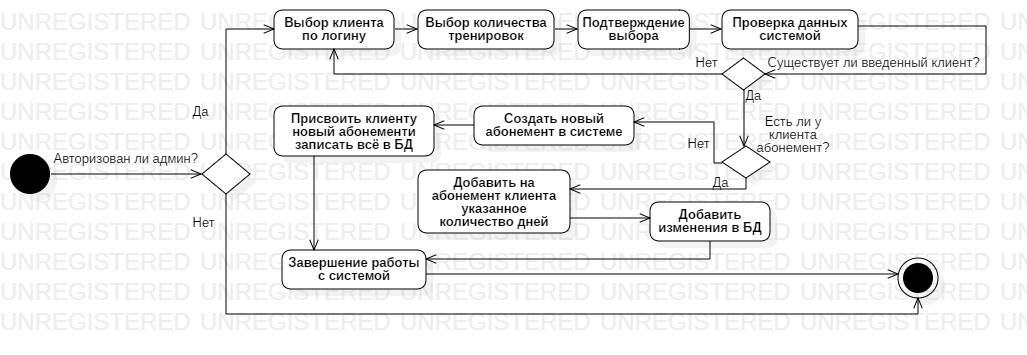
Диаграмма активности

Диаграмма описывает процесс запроса какой-либо информации из базы данных.



Диаграммы состояний

Первая диаграмма описывает процесс добавления тренировок на абонемент клиента. Тут описаны варианты, когда у клиента еще нет абонемента, и когда клиента нет вовсе. Когда абонемента нет – создается запись в БД о новом абонементе, когда он есть – просто введенное количество занятий добавляется к текущему абонементу.



Вторая диаграмма описывает процесс записи на тренировку клиентом. Действие выбор тренировки означает выбор филиала, тренера, дисциплины и времени.

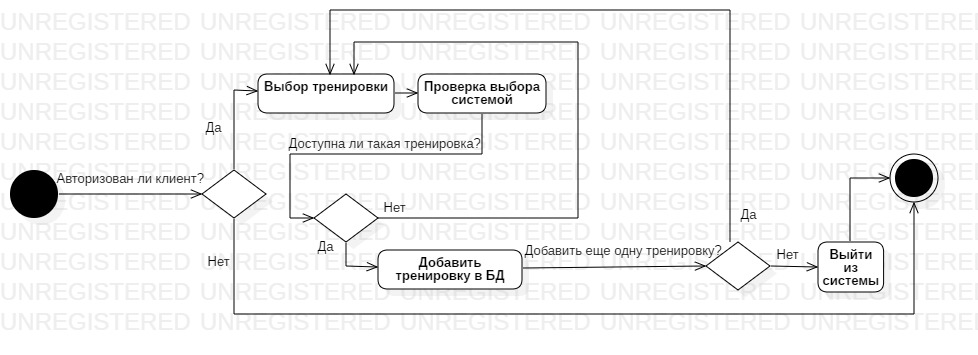
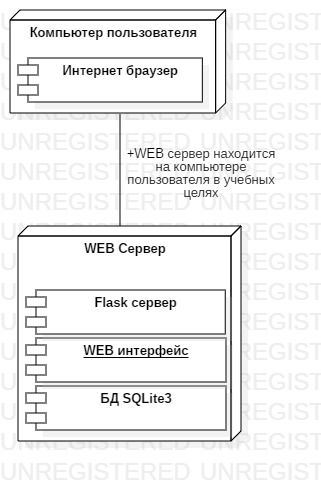


Диаграмма развертывания

Показывает какие узлы существуют в системе.



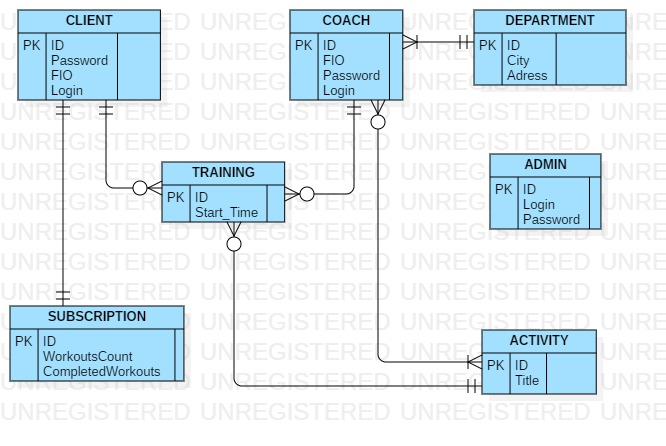
ER диаграмма

ER диаграмма состоит из 7 сущностей. 3 сущности обозначают пользователей системы (CLIENT, COACH, ADMIN). Так как каждый пользователь должен иметь возможность авторизации на сайте, то соответственно необходимы атрибуты логина и пароля. Так же у каждой сущности есть атрибут ID, выступающий первичным ключом.

Сущность DEPARTMENT (филиал) – необходима при регистрации нового тренера (так как тренер привязан к какому-то филиалу).

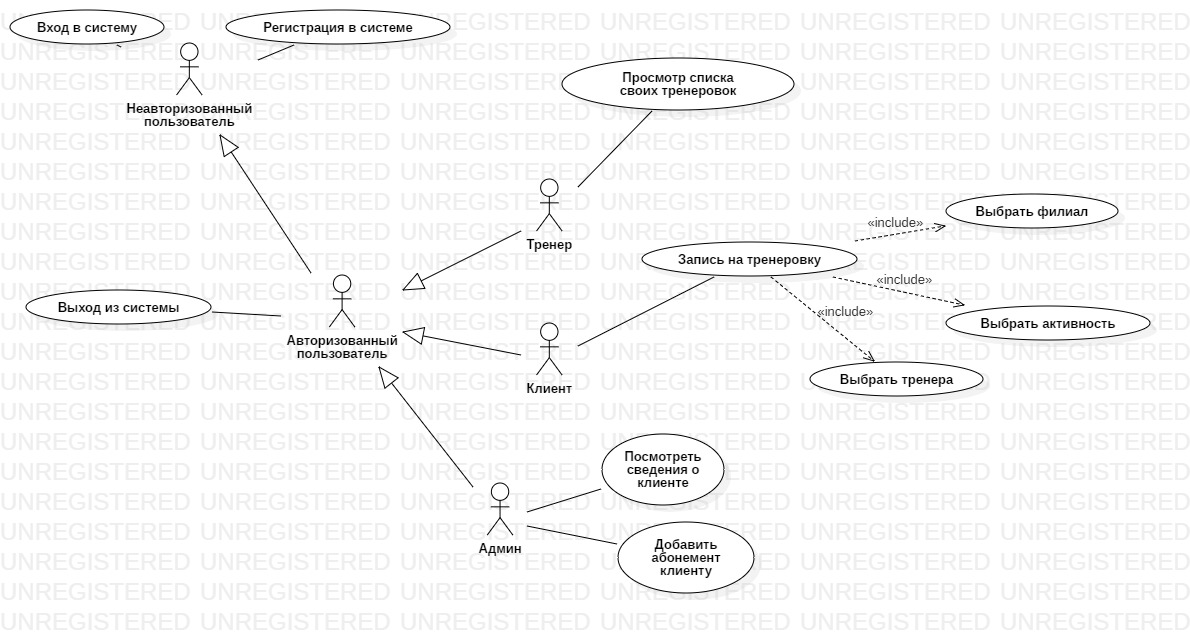
Следующая наиболее важная сущность – TRAINING (тренировка). Она необходима так как в системе должна быть возможность создания тренировки клиентом. Сущность тренировки включает в себя поле START\_TIME, обозначающее время и дату начала тренировки, а также ссылки на клиента, тренера и дисциплину тренировки. Отношения многие к одному, так как на одной тренировке может быть только один тренер, один клиент (в нашем проекте рассматриваются только персональные тренировки) и одна дисциплина.

Следующая сущность – SUBSCRIPTION, то есть абонемент. Атрибуты – WorkoutsCount (количество тренировок на абонементе), CompletedWorkouts (Количество тренировок, которые клиент уже отходил). Зная эти два параметра, мы можем вычислить сколько клиенту осталось отходить тренировок.



USE CASE диаграмма

Диаграмма объясняет функционал системы. У нас есть 3 основных актера – тренер, клиент и администратор. Каждый может выполнять какую-либо функцию. Так же все они являются авторизованными пользователями, которые могут выйти из системы. Еще есть актер неавторизованный пользователь, который может зарегистрироваться в системе или войти в неё.



Физическая диаграмма

Диаграмма была нам нужна для создания таблиц БД. Кроме сущностей из ER диаграммы тут появилась сущность COACH\_ACTIVITY, как связующее звено между тренером и дисциплиной. Такая сущность возникла из-за отношения многие ко многим. Так же в данной диаграмме уже есть атрибуты внешних ключей, которые отражают зависимости между таблицами.

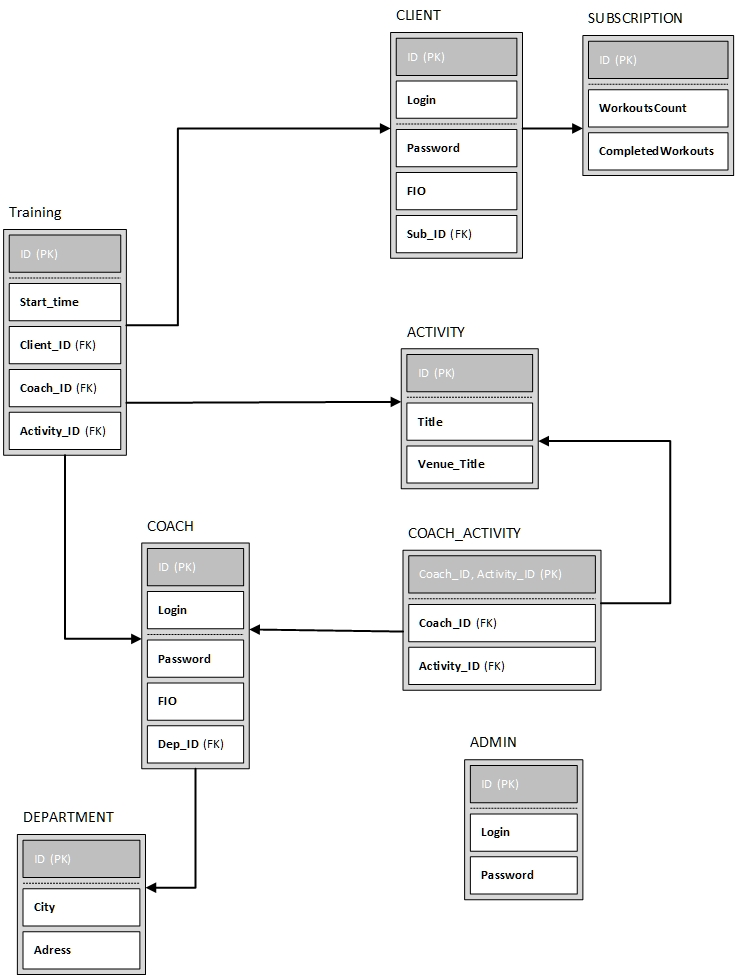


Диаграмма классов

Диаграмма описывает классы системы. Основные элементы такие же, как и в ER диаграмме, но тут добавились некоторые методы у классов. Тренер может посмотреть свой график, администратор добавить тренировок клиенту, а клиент записаться на тренировку.

Так как атрибут ID общий для всех, то он вынесен в отдельный класс IElement и от него наследуются другие классы. Так же пользователи системы имеют общие поля логина и пароля, а также общий метод выйти из системы, так что для них также есть отдельный класс IUser, от которого они наследуются.

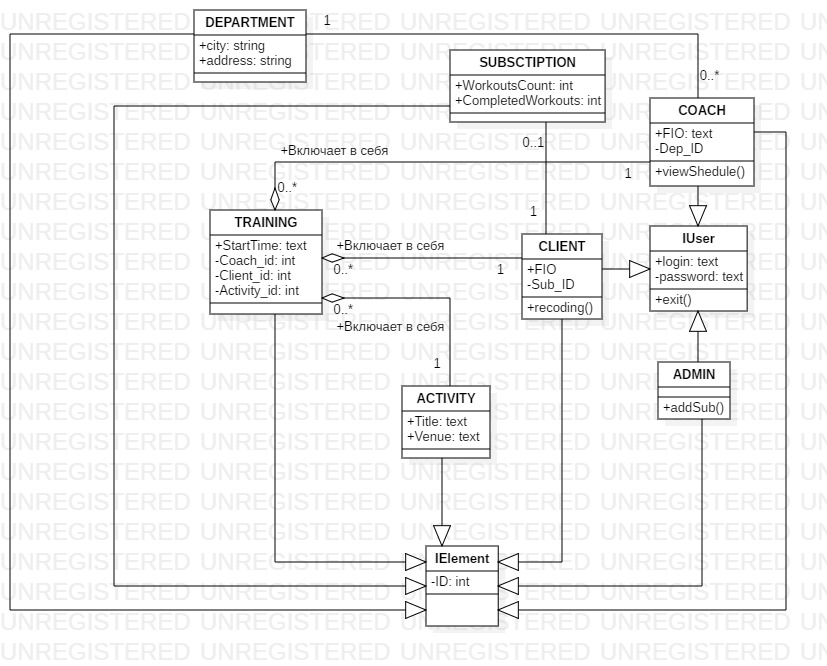


Диаграмма объектов

Отражает состояние системы в определенный момент времени. Тут отображены объекты, соответствующие классам, но уже с заданными конкретными атрибутами.

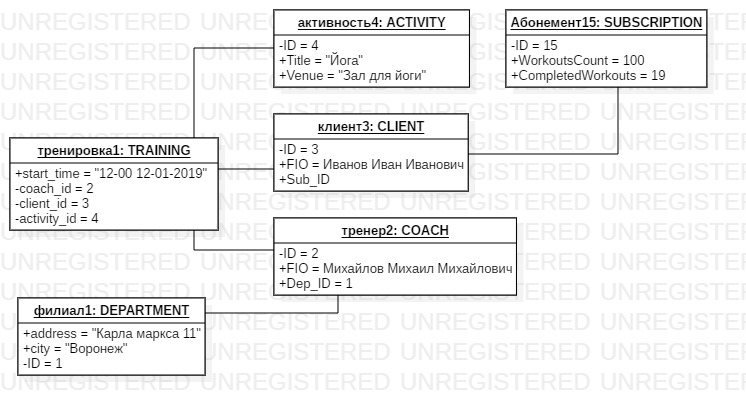
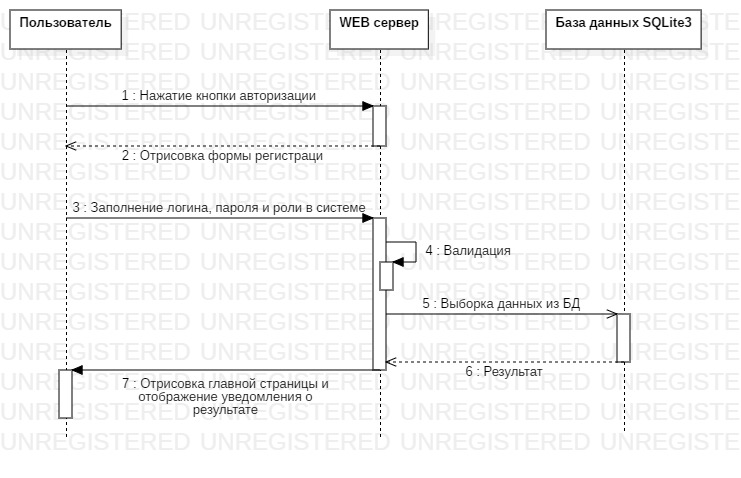


Диаграмма последовательности

Данная диаграмма описывает прецедент авторизации в системе неавторизованного пользователя.



ORM модели

Данный код позволяет взаимодействовать с базой данных с помощью библиотеки peewee. Тут отражены все сущности физической диаграммы. Код из файла models.py:

*# Интрефейс Базовой модели*class BaseModel(Model):  
 class Meta:  
 database = db  
*# Интерфейс сущности пользователь*class IUser(BaseModel):  
 Login = FixedCharField(20)  
 Password = FixedCharField(20)  
 def exit(self):  
 pass  
*# Интерфейс сущности элемент системы*class IElement(BaseModel):  
 ID = AutoField()  
*# Абонемент*class SUBSCRIPTION(IElement):  
 WorkoutsCount = IntegerField(default=30)  
 CompletedWorkouts = IntegerField(default=0)  
*# Филиал клуба*class DEPARTMENT(IElement):  
 City = FixedCharField(45)  
 Address = FixedCharField(250)  
 def getDepList():  
 return 0  
*# Админы*class ADMIN(IUser, IElement):  
 def addSub(client, days):  
 return 0 def viewSub(login):  
 return 0  
*# Клиенты клуба*class CLIENT(IUser, IElement):  
 FIO = FixedCharField(100)  
 Sub\_ID = ForeignKeyField(  
 SUBSCRIPTION,  
 db\_column='Sub\_ID'  
 )  
 def recording(self):  
 pass  
*# Тренера*class COACH(IUser, IElement):  
 FIO = FixedCharField(100)  
 Dep = ForeignKeyField(  
 DEPARTMENT,  
 db\_column='Dep'  
 )  
 def viewShedule(coach):  
 return 0  
*# Активности*class ACTIVITY(IElement):  
 Title = FixedCharField(45)  
 Venue\_Title = FixedCharField(45)  
 def getActivities():

returm 0  
*# Активность тренера*class COACH\_ACTIVITY(BaseModel):  
 Coach\_ID = ForeignKeyField(COACH, db\_column='Coach\_ID')  
 Activity\_ID = ForeignKeyField(ACTIVITY, db\_column='Activity\_ID')  
 class Meta:  
 primary\_key = CompositeKey('Coach\_ID','Activity\_ID')  
*# Тренировки*class TRAINING(IElement):  
 Start\_time = DateTimeField(formats='%d.%m.%Y %H:%M')  
 Client = ForeignKeyField(CLIENT, db\_column='Client\_ID')  
 Coach = ForeignKeyField(COACH, db\_column='Coach\_ID')  
 Activity = ForeignKeyField(ACTIVITY, db\_column='Activity\_ID')

Модуль регистрации клиента и тренера

В файле маршрутов были прописаны маршруты регистрации для тренера и клиента:

@app.route('/regasclient', methods=['GET', 'POST'])  
def regasclient():  
 if 'username' in session:  
 flash('Вы уже авторизовались в системе')  
 return redirect(url\_for('index'))  
 form = RegAsClientForm()  
 if form.validate\_on\_submit():  
 element = addClient(request.form['fio'], request.form['username'], request.form['password'])  
 if(not element):  
 flash('Пользователь с таким логином уже существует')  
 return redirect(url\_for('index'))  
 else:  
 session['username'] = element.Login  
 session['FIO'] = element.FIO  
 session['role'] = 'client'  
 flash('Вы успешно зарегистрировались в системе')  
 return redirect(url\_for('index'))  
 return render\_template(  
 'regasclient.html',  
 form = form,  
 )

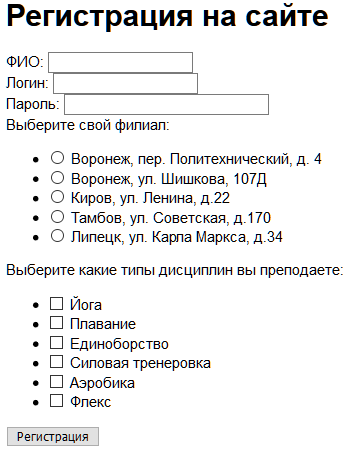
и аналогично для тренера.

Так же были созданы flask формы регистрации, написаны HTML шаблоны для страниц регистрации. Были реализованы методы addCoach и addClient, в которых происходит вставка данных в БД.

Часть кода данных методов, в которой происходит как раз вставка в БД:

COACH.insert(FIO = fio,  
 Login = login,  
 Password = password,  
 Dep = dep).execute()

Пример того как выглядит форма регистрации тренера на данный момент:



Модуль авторизации пользователей:

В маршрутах был прописан маршрут авторизации пользователей:

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

def login():

form = LoginForm()

if 'username' in session:

flash('Вы уже авторизовались в системе')

return redirect(url\_for('index'))

if form.validate\_on\_submit():

element = VerifyAuthData(form)

if(not element):

flash('Вы ввели неверные данные, попробуйте еще раз')

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

else:

session['username'] = element[0].Login

if request.form['roles'] != 'admin':

session['FIO'] = element[0].FIO

else:

session['FIO'] = 'Администратор'

session['role'] = request.form['roles']

flash('Вы успешно авторизовались в системе')

return redirect(url\_for('index'))

return render\_template(

'login.html',

form = form,

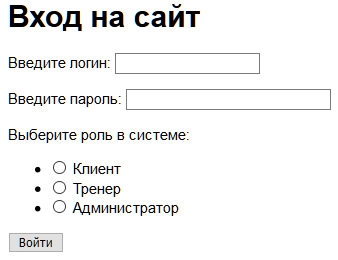
)

Был создан HTML шаблон и форма Flask входа авторизации в системе. Так же написан метод VerifyAuthData, проверяющий возможность войти в систему.

Часть кода из этого метода, проверяющая наличие соответствующего пользователя (в данном случае тренера):

coach = COACH.get((COACH.Login == login) & (COACH.Password == password))

Так на данный момент выглядит форма авторизации на сайте:



Модуль вывода списка тренировок

Был написан рабочий модуль вывода списка тренировок для тренера:

def viewShedule(coach):

data = []

for el in TRAINING.select().where(TRAINING.Coach == coach.ID).order\_by(TRAINING.Start\_time):

date\_obj = datetime.strptime(el.Start\_time, '%d.%m.%Y %H:%M')

fio = CLIENT.get(CLIENT.id == el.Client\_ID).FIO

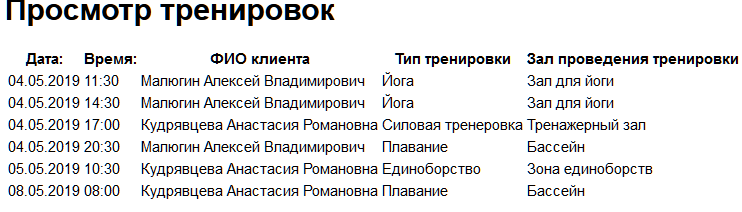
activity = ACTIVITY.get(ACTIVITY.ID == el.Activity\_ID).Title

venue = ACTIVITY.get(ACTIVITY.ID == el.Activity\_ID).Venue\_Title

data.append((date\_obj.strftime('%d.%m.%Y'),date\_obj.strftime('%H:%M'),fio, activity,venue ))

return data

Пример работы модуля:



Соответственно также был написан HTML шаблон.

Модуль добавления тренировок клиенту

Был реализован модуль добавления тренировок. Это метод администратора. Сам код:

def addSub(client, days):

flag = False

sub = 0

try:

sub = SUBSCRIPTION.get(SUBSCRIPTION.ID == client.Sub\_ID)

except:

flag = True

if flag:

sub\_id = SUBSCRIPTION.insert(WorkoutsCount = days, CompletedWorkouts = 0).execute()

CLIENT.update(Sub\_ID = sub\_id).where(CLIENT.Login == client.Login).execute()

return 1

else:

newdays = sub.WorkoutsCount + days

SUBSCRIPTION.update(WorkoutsCount = newdays).where(SUBSCRIPTION.ID == client.Sub\_ID).execute()

return 1

Если у клиента нет абонемента, то создадим его. Если есть, то просто добавим заданное количество дней на абонемент (newdays = sub.WorkoutsCount + days).

Модуль просмотра статуса абонемента клиента

Был также разработан модуль просмотра статуса абонемента клиент администратором.

def viewSub(login):

try:

CLIENT.get(CLIENT.Login == login)

except:

return 0

isExistsSub = True

client = CLIENT.get(CLIENT.Login == login)

FIO = client.FIO

sub = None

try:

sub = SUBSCRIPTION.get(SUBSCRIPTION.ID == client.Sub\_ID)

except:

isExistsSub = False

if isExistsSub:

days = sub.WorkoutsCount - sub.CompletedWorkouts

return list([isExistsSub, FIO, days])

else:

return list([isExistsSub, FIO])

Если абонемент у клиента имеется, то возвращаются значения, кроме ФИО еще и количество оставшихся на абонементе тренировок.

Пример результатов работы данного модуля:

