Отчет лабораторная 3

Выполнил Азаров И.С. АСМ-22-04

1. Задание

Преобразовать WSGI-приложение картотеки в клиента базы данных SQLite.

Требования к реализации:

- учесть возможные изменения идентификатора своего модуля;

предусмотреть возможность импорта информации, сохраненной первой лабораторной;

использовать механизм подстановки параметров DB-API;

- для хранения данных использовать подкаталог data, для размещения шаблонов подкаталог app/templates, при необходимости — app/static.

1. Реализация

2.1 Основной модуль приложения

**from** **CompanyModule** **import** \*

**from** **flask** **import** Flask, g

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

**def** **getCompany**():

**if** 'company' **not** **in** g:

g.company = Company()

**return** g.company

**@app**.route("/")

**def** **companyindex**():

**return** getCompany().show()

**@app**.route("/showform/<id>")

**def** **showform**(id):

**return** getCompany().showFormId(id)

**@app**.route("/delete/<id>")

**def** **delete**(id):

**return** getCompany().delete(id)

**@app**.route("/add/", methods=['POST'])

**def** **add**():

**return** getCompany().add()

**@app**.teardown\_appcontext

**def** **teardown**(ctx):

getCompany().storage.save()

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

app.run(debug=**True**)

Основного меню приложения построено с использование фреймворка Flask, расположено по адресу <http://localhost:5000> (по умолчанию). Для построения меню используются декораторы (функции в которые передается метод стоящий после них как аргумент и возвращающие также функцию), с указанием адреса после основного.

@app.route("/showform/<id>") вернет результат выполнения метода showform с переданным в него аргументом id. Результатом выполнения будет форма, построенная на шаблонах tpl (п. 2.6). Декоратор привязан к ссылке «добавить сотрудника» (используется с 0 id) и ссылке «Изменить» (используется с guid). Пример выполнения на рисунке 1.

@app.route("/") используется для отображения всех элементов организации. @app.route("/delete/<id>") используется для удаления элемента с указанным id, привязан к ссылке «Удалить».

@app.route("/add/", methods=['POST']) используется для добавления нового сотрудника, привязан к кнопке «Добавить/изменить».

@app.teardown\_appcontext сериализует созданный класс Company в папку проекта.

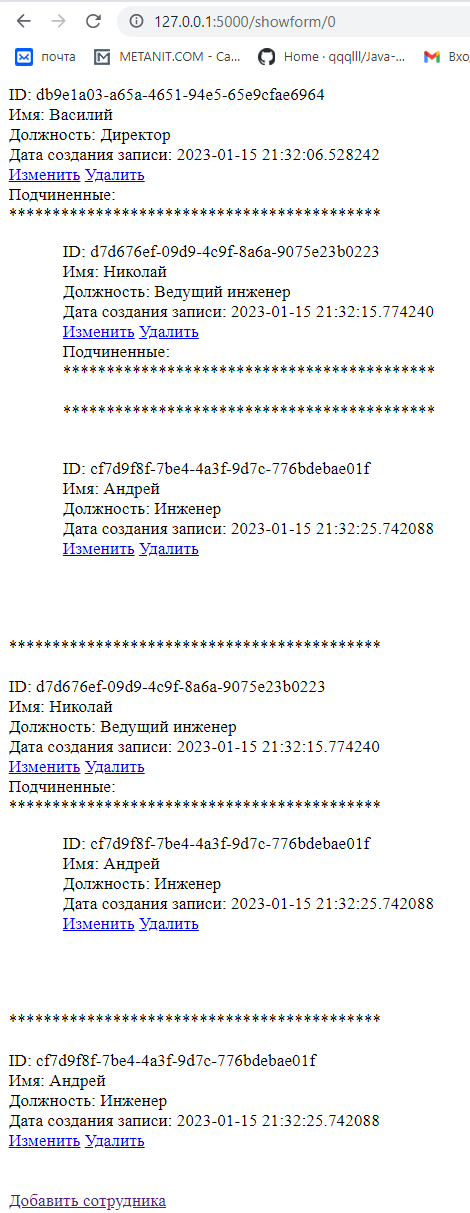


Рисунок 1 – Результат возврата формы ввода данных по ссылке

2.2 Модуль для создания базы данных и таблиц в SQLite

**import** **sqlite3**

**def** **createDB**():

**with** sqlite3.connect('data/Companies.db') **as** con:

cursor = con.cursor()

cursor.execute("""

create table Companies

(

Id text not null primary key,

CreationDate datetime null,

Name text not null

)"""

)

cursor.execute("""

create table Employees

(

Id text not null primary key,

FullName text not null,

Position text not null,

CreationDate datetime null,

CompanyId text null,

FOREIGN KEY (CompanyId) REFERENCES Companies(id)

)"""

)

cursor.execute("""insert into "Companies"

values

('35779ccf-a666-4596-9f44-15445f77e0e0',

date('now'),

'ООО Тестовая'

)""")

Создаются 2 таблицы в базе данных Companies: Companies и Employees. Companies состоит из 3 столбцов:

- id, идентификатор записи, имеет тип данных текст (тип guid в SQLite отсутствует) и является первичным ключом;

- CreationDate, дата создания записи, имеет тип datetime;

- Name, наименование организации.

Employees состоит из 5 столбцов, содержит данные о сотрудниках

- id, идентификатор записи, имеет тип данных текст (тип guid в SQLite отсутствует) и является первичным ключом;

- FullName, имя или ФИО сотрудника, тип данных текст;

- Position, должность, тип данных текст;

- CreationDate, дата создания записи, имеет тип datetime;

- CompanyId, идентификатор организации, к которой принадлежат сотрудники.

Пример созданных таблиц и содержимое одной из них на рисунке 2 (из SQLiteStudio).

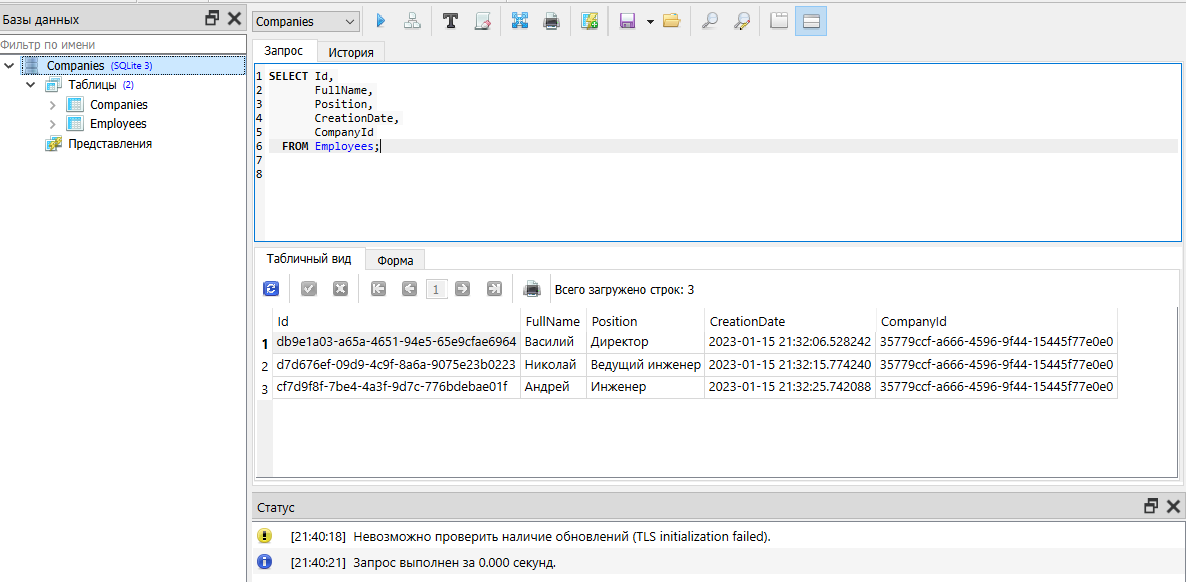


Рисунок 2 – Пример вида базы данных и содержимого таблицы Employees

2.3 Модуль, реализующий основную логику работы приложения

**from** **dataclasses** **import** dataclass

**from** **StorageModule** **import** \*

**from** **abc** **import** ABC, abstractmethod

**from** **flask** **import** render\_template, request

**import** **datetime**

**from** **StrategyModule** **import** IStrategy, SQLIOStrategy

**@dataclass**

**class** **Company**:

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

self.name = "ООО **\"**Тестовая**\"**"

self.storage = DBStorage(self)

self.io:IStrategy = SQLIOStrategy(self.storage)

self.director: ConcreteEmployee = ConcreteEmployee()

self.leadEngineers: list[ConcreteEmployee] = []

self.engineers: list[LeafEmployee]= []

**def** **add**(self):

self.setData(request.form)

**return** self.io.output()

**def** **delete**(self, id):

self.storage.delete(id)

**return** self.io.output()

**def** **show**(self):

**return** self.io.output()

**def** **setData**(self, form):

id = form.get('Id')

fullName = form.get('fullName')

position = form.get('position')

self.storage.add(id, fullName, position)

**def** **showFormId**(self, id):

**return** self.storage.getItem(id).showForm()

**def** **showForm**(self):

**return** render\_template('form.tpl', it=self.storage.getItems())

#абстрактный класс для сотрудников фирмы

**class** **Employee**(ABC):

**@abstractmethod**

**def** **addChild**(self, employee):

**pass**

**@abstractmethod**

**def** **removeChild**(self, employee):

**pass**

**@abstractmethod**

**def** **showForm**(self):

**pass**

**@abstractmethod**

**def** **show**(self):

**pass**

**@dataclass**

**class** **ConcreteEmployee**(Employee):

fullName: str = ""

position: str = ""

creationTime = **0**

childs = []

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

self.id = str(uuid.uuid4())

self.creationTime = datetime.datetime.now()

**def** **addChild**(self, employee):

self.childs.append(employee)

**def** **removeChild**(self, employee):

self.childs.remove(employee)

**def** **showForm**(self):

**return** render\_template('form.tpl', it=self)

**def** **show**(self):

**return** "Редактировать (руководители)"

**def** **mapFields**(self, data):

**if** data **is** **not** **None**:

self.id = data[**0**]

self.fullName = data[**1**]

self.position = data[**2**]

self.creationTime = data[**3**]

self.childs = []

**@dataclass**

**class** **LeafEmployee**(Employee):

fullName: str = ""

position: str = ""

creationTime = **0**

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

self.id = str(uuid.uuid4())

self.creationTime = datetime.datetime.now()

**def** **addChild**(self, employee):

**raise** **Exception**("Подчиненных нет")

**def** **removeChild**(self, employee):

**raise** **Exception**("Подчиненных нет")

**def** **showForm**(self):

**return** render\_template('form.tpl', it=self)

**def** **show**(self):

**return** "Редактировать (младший персонал)"

**def** **mapFields**(self, data):

**if** data **is** **not** **None**:

self.id = data[**0**]

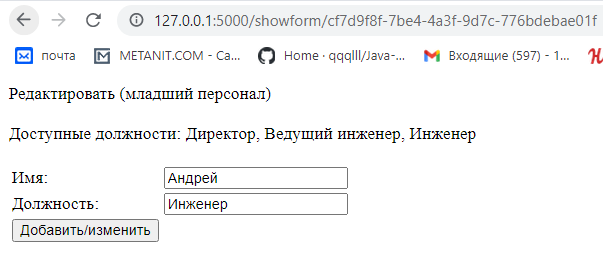
self.fullName = data[**1**]

self.position = data[**2**]

self.creationTime = data[**3**]

Основной класс Company. Содержит списки сотрудников, методы для добавления или удаления сотрудников из списков, сериализации списков. Добавлен метод showForm для вызова формы ввода данных в web-представлении с использованием файла шаблона form.tpl (описан в п. 2.6).

Для ввода/вывода данных используется модуль стратегии 2.4. Для сериализации модуль 2.5. Паттерн «компоновщик» имеет общий «интерфейс» Employee, и конкретные классы, реализующие этот интерфейс. Интерфейс имеет абстрактные методы для добавления или удаления потомков, метод для вывода данных в консоль. Добавлен метод для вызова формы ввода данных. Класс ConcreteEmployee для создания руководителей организации и LeafEmployee для создания подчиненных. Отличаются наличием или отсутствием потомков. Классы имеют 4 информационных поля, id, имя, должность, дата создания записи (используется текущая дата). Для маппинга полей классов из массива кортежей, полученного от SQLite-сервера используется метод mapFields. Объекты этих классов имеют каждый свою форму редактирования (отличается заголовком), рисунок 3.



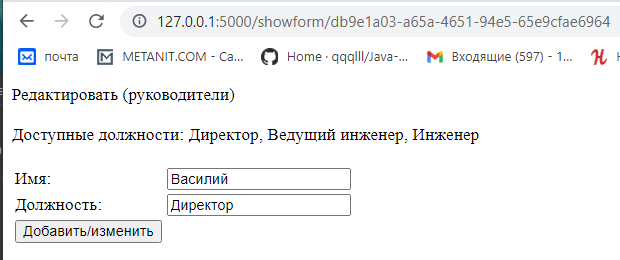


Рисунок 3 – Форма для редактирования

2.4 Модуль для паттерна «стратегия»

**from** **abc** **import** ABC, abstractmethod

**from** **flask** **import** render\_template

# абстрактный класс для стратегии

**class** **IStrategy**(ABC):

**@abstractmethod**

**def** **input**(self, name ):

**pass**

**@abstractmethod**

**def** **output**(self, title, field):

**pass**

#конкретная стратегия для ввода вывода с консоли

**class** **ConsoleIOStrategy**(IStrategy):

**def** **input**(self, field=**None**):

**return** input(f"{field}: ")

**def** **output**(self, title=**None**, field=**None**):

print(f"{title}: {field}")

#конкретная стратегия для ввода вывода через веб форму

**class** **WebIOStrategy**(IStrategy):

**def** **\_\_init\_\_**(self, storage):

self.storage = storage

**def** **input**(self, field=**None**):

**return** render\_template('form.tpl', it=self)

**def** **output**(self, title=**None**, field=**None**):

company = self.storage.getItems()

**if** company **is** **not** **None**:

**return** render\_template('company.tpl', it=company.director, leads=company.leadEngineers, engs=company.engineers)

**else**:

**return** render\_template('company.tpl', it="", leads="", engs="")

# конкретная стратегия для ввода вывода через веб форму

**class** **SQLIOStrategy**(IStrategy):

**def** **\_\_init\_\_**(self, storage):

self.storage = storage

**def** **input**(self, field=**None**):

**return** render\_template('form.tpl', it=self)

**def** **output**(self, title=**None**, field=**None**):

company = self.storage.getItems()

**if** company **is** **not** **None**:

**return** render\_template('company.tpl', it=company.director, leads=company.leadEngineers,

engs=company.engineers)

**else**:

**return** render\_template('company.tpl', it="", leads="", engs="")

Паттерн реализован через «интерфейс» IStrategy, использующей для создания абстрактных методов библиотеку ABC. Конкретную стратегию для ввода/вывода через консоль реализует класс ConsoleIOStrategy. Конкретную стратегию для ввода/вывода через web интерфейс реализует класс WebIOStrategy. Метод input вызывает форму для ввода данных, output выводит всю информацию о сотрудниках организации. Конкретную стратегию для ввода/вывода в таблицу Employees базы данных SQLite через web интерфейс реализует класс SQLIOStrategy.

2.5 Модуль для хранения данных

**import** **pickle**

**import** **sqlite3**

**import** **uuid**

**from** **datetime** **import** datetime

**from** **sqlite3** **import** Connection

**import** **CompanyModule**

**import** **DB**

**class** **Storage**:

**def** **\_\_init\_\_**(self, company):

self.company = company

**try**:

self.load()

**except**:

self.company.director = CompanyModule.ConcreteEmployee()

self.company.leadEngineers:list[CompanyModule.ConcreteEmployee] = []

self.company.engineers:list[CompanyModule.LeafEmployee] = []

**def** **save**(self):

pickle.dump((self.company.director, self.company.leadEngineers, self.company.engineers), open("data.dat", "wb"))

**def** **load**(self):

(self.company.director, self.company.leadEngineers, self.company.engineers) = pickle.load(open("data.dat", "rb"))

**def** **getItems**(self):

company = CompanyModule.Company()

company.director = self.company.director

company.leadsEngineers = self.company.leadEngineers

company.engineers = self.company.engineers

**return** company

**def** **delete**(self, id):

**if** self.company.director.id == id:

**import** **CompanyModule**

self.company.director = CompanyModule.ConcreteEmployee()

**if** len(self.company.director.childs) > **0**:

**for** child **in** self.company.director.childs:

**if** child.id == id:

self.company.director.removeChild(child)

**if** len(self.company.leadEngineers) > **0**:

**for** emp **in** self.company.leadEngineers:

**if** emp.id == id:

self.company.leadEngineers.remove(emp)

**if** len(emp.childs) > **0**:

**for** child **in** emp.childs:

**if** child.id == id:

emp.removeChild(child)

**if** len(self.company.engineers) > **0**:

**for** emp **in** self.company.engineers:

**if** emp.id == id:

self.company.engineers.remove(emp)

**def** **getItem**(self, id):

**if** id == '0':

**return** CompanyModule.Company()

**if** self.company.director.id == id:

**return** self.company.director

**if** len(self.company.leadEngineers) > **0**:

**for** emp **in** self.company.leadEngineers:

**if** emp.id == id:

**return** emp

**if** len(self.company.engineers) > **0**:

**for** emp **in** self.company.engineers:

**if** emp.id == id:

**return** emp

**raise** **Exception**("Сотрудника с таким Id не существует")

**def** **add**(self, fullName, position):

**if** position == "Директор":

self.company.director.fullName = fullName

self.company.director.position = "Директор"

**if** len( self.company.leadEngineers) > **0**:

**for** emp **in** self.company.leadEngineers:

self.company.director.addChild(emp)

**if** len( self.company.engineers) > **0**:

**for** emp **in** self.company.engineers:

self.company.director.addChild(emp)

**elif** position == "Ведущий инженер":

leadEngineer = CompanyModule.ConcreteEmployee()

leadEngineer.fullName = fullName

leadEngineer.position = "Ведущий инженер"

**if** len( self.company.engineers) > **0**:

**for** emp **in** self.company.engineers:

leadEngineer.addChild(emp)

self.company.leadEngineers.append(leadEngineer)

**elif** position == "Инженер":

engineer = CompanyModule.LeafEmployee()

engineer.fullName = fullName

engineer.position = "Инженер"

self.company.engineers.append(engineer)

**class** **DBStorage**:

con: Connection

**def** **\_\_init\_\_**(self, company):

self.company = company

self.companyId = '35779ccf-a666-4596-9f44-15445f77e0e0'

self.connect()

**def** **connect**(self):

self.con = sqlite3.connect('data/Companies.db', detect\_types=sqlite3.PARSE\_DECLTYPES)

self.cursor = self.con.cursor()

**def** **save**(self):

self.con.commit()

self.con.close()

**def** **load**(self):

DB.createDB()

**def** **getItems**(self):

company = CompanyModule.Company()

self.cursor.execute("select \* from Employees where Position = 'Директор'")

data = self.cursor.fetchone()

self.cursor.execute("""select \* from Employees""")

dataChilds = self.cursor.fetchall()

**if** data **is** **not** **None**:

company.director.mapFields(data)

self.setChilds(dataChilds, company.director)

self.cursor.execute("select \* from Employees where Position = 'Ведущий инженер'")

data = self.cursor.fetchall()

**for** item **in** data:

tmpLead = CompanyModule.ConcreteEmployee()

tmpLead.mapFields(item)

self.setChilds(dataChilds, tmpLead)

company.leadEngineers.append(tmpLead)

self.cursor.execute("select \* from Employees where Position = 'Инженер'")

data = self.cursor.fetchall()

**for** item **in** data:

tmpEng = CompanyModule.LeafEmployee()

tmpEng.mapFields(item)

company.engineers.append(tmpEng)

**return** company

**def** **delete**(self, id):

self.cursor.execute("delete from Employees where Id = ?", (id,))

**def** **getItem**(self, id):

self.cursor.execute("""select \* from Employees

where Id =?""", (id,))

data = self.cursor.fetchone()

**if** id == '0':

**return** CompanyModule.Company()

**if** data[**2**] == 'Директор':

director = CompanyModule.ConcreteEmployee()

director.mapFields(data)

self.setChilds(data, director)

**return** director

**if** data[**2**] == 'Ведущий инженер':

leadEngineer = CompanyModule.ConcreteEmployee()

leadEngineer.mapFields(data)

self.setChilds(data, leadEngineer)

**return** leadEngineer

**if** data[**2**] == 'Инженер':

eng = CompanyModule.LeafEmployee()

eng.mapFields(data)

**return** eng

**def** **setChilds**(self, data, employee):

**if** employee.position == 'Директор':

**for** item **in** data:

**if** item[**2**] == 'Ведущий инженер':

child = CompanyModule.ConcreteEmployee()

child.mapFields(item)

employee.addChild(child)

**if** item[**2**] == 'Инженер':

child = CompanyModule.LeafEmployee()

child.mapFields(item)

employee.addChild(child)

**if** employee.position == 'Ведущий инженер':

**for** item **in** data:

**if** item[**2**] == 'Инженер':

child = CompanyModule.LeafEmployee()

child.mapFields(item)

employee.addChild(child)

**def** **add**(self, id, fullName, position):

self.cursor.execute("select \* from Employees where Id=?", (id,))

addedEmployee = self.cursor.fetchone()

**if** addedEmployee **is** **not** **None**:

self.update(id, fullName, position)

**else**:

self.cursor.execute("""insert into Employees

values(?,?,?,?,?)""", (uuid.uuid4().\_\_str\_\_(), fullName, position, datetime.now(), self.companyId))

**def** **update**(self,id, fullName, position):

self.cursor.execute("update Employees set FullName=?, Position=? where id=?", (fullName, position, id))

Класс Storage используется для сериализации/десериализации данных в каталог операционной системы (каталог проекта) с использованием библиотеки pickle (методы save и load).

Класс DBStorage используется для добавления, удаления, обновления данных в таблице Employees базы данных Companies на SQLite-сервере. Для добавления новой записи в таблицу в методе add используется запрос insert c параметрами. Для получения записи(ей) используются методы getItem и getItems, использующие запрос select c фильтром where. Результат выполнения запроса возвращается в виде массива кортежей, где элементы массива значения строк, элементы кортежа значения столбцов, рисунок 4. Для обновления данных используется метод update и соответствующий запрос update с параметрами и фильтром where.

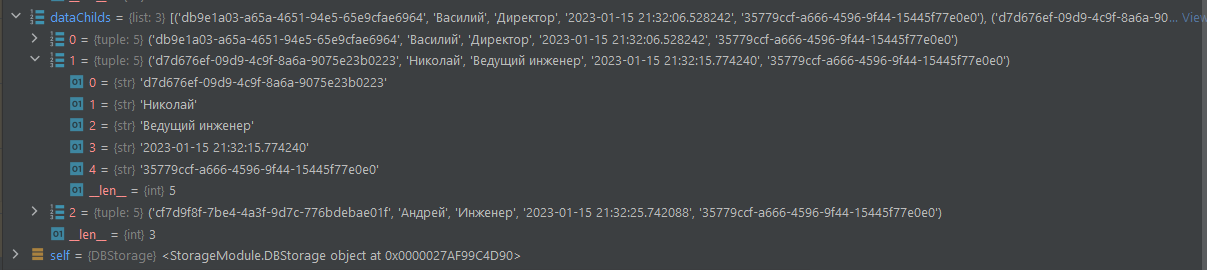


Рисунок 4 – Полученные из таблицы данные

2.6 Шаблоны для web – представлений

**base.tpl** основной шаблон, который используется как базовый для расширения другими шаблонами. Позволяет выводить данные подряд на одну страницу.

{% include "header.tpl" ignore missing %}

{% block content %} {% endblock %}

{% include "footer.tpl" ignore missing %}

**add.tpl** содержит ссылку на добавление сотрудника

<a href='/showform/0'>Добавить сотрудника</a><br>

**company.tpl** шаблон для вывода всех сотрудников компании, содержит шаблон add.tpl и шаблон item.tpl. В шаблоне используются циклы и условные операторы для вывода элементов item.tpl.

{% extends "base.tpl" %}

{% block content %}

{% if it !="" %}

{% include "item.tpl" ignore missing %}

{% else %}

Нет сотрудников<br>

{% endif %}

{% for it in leads %}

{% include "item.tpl" ignore missing %}

{% endfor %}

{% for it in engs %}

{% include "item.tpl" ignore missing %}

{% endfor %}

{% include "add.tpl" ignore missing %}

{% endblock %}

**footer.tpl** пустой шаблон

**form.tpl** шаблон формы ввода данных, расширяет базовый шаблон. В нем используется табличная верстка для выравнивая inputbox и его заголовка. Кнопка вызывает метод add для добавления нового сотрудника или изменения данных уже существующего.

{% extends "base.tpl" %}

{% block content %}

{{it.show()}}

<form action = '/add' method=POST><br>

Доступные должности: Директор, Ведущий инженер, Инженер<br>

<input type=hidden name=Id value={{it.id}}><br>

<table>

<tr>

<td>

Имя:

</td>

<td>

<input type=text name=fullName value={{it.fullName}}>

</td>

</tr>

<tr>

<td>

Должность:

</td>

<td>

<input type=text name=position value={{it.position}}>

</td>

</tr>

<tr>

<td>

<input type=submit value="Добавить/изменить">

</td>

<tr>

</table>

</form>

{% endblock %}

**header.tpl** заголовок, содержит наименование организации

<html><head><title>Организация ООО "Тестовая"</title></head><body>

**item.tpl** шаблон элемента, который выводится в шаблоне company.tpl. Содержит информацию о элементе, имеет ссылки на изменение или удаление элемента по его id. Вызывает соответствующие методы из модуля п. 2.1.

ID: {{it.id}}<br>

Имя: {{it.fullName}}<br>

Должность: {{it.position}}<br>

Дата создания записи: {{it.creationTime}}<br>

<a href="/showform/{{it.id}}">Изменить</a>

<a href="/delete/{{it.id}}">Удалить</a><br>

{%if it.position != "Инженер" %}

Подчиненные: <br>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<br>

{% for it in it.childs %}

<p style="margin-left: 50px;">{% include "item.tpl" ignore missing %}</p>

{% endfor %}

<br>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{%endif%}

<br><br>