Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт фундаментального образования

Кафедра интеллектуальных информационных технологий

**БАЗА ДАННЫХ**

|  |  |
| --- | --- |
| Преподователь | Алферьева Т.И. |
| Студенты гр. ФО-350005 | Савинова М.А. |

Екатеринбург - 2017

**Базы данных (Models)**

Сам фреймворк Pyramid не имеет встроенных возможностей работы с базами данных, в отличии от таких фреймворков как Django . Поэтому для реляционных БД можно использовать Алхимию

**SQLAlchemy**

Организация БД в пирамиде не зависит от фреймворка, поэтому можно использовать любую структуру, которая вам удобна. Ниже я приведу один из вариантов, более подробно про SQLAlchemy можно прочитать в разделе [SQLAlchemy ORM](http://lectures.uralbash.ru/6.www.sync/2.codding/9.databases/2.sqlalchemy/index.html" \l "sqlalchemy).

**from** **sqlalchemy** **import** Column, Integer, Text, create\_engine

**from** **sqlalchemy.ext.declarative** **import** declarative\_base

**from** **sqlalchemy.orm** **import** sessionmaker

engine = create\_engine('sqlite:///foo.db')

Session = sessionmaker()

Base = declarative\_base(bind=engine)

**class** **User**(Base):

\_\_tablename\_\_ = 'models'

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

name = Column(Text)

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

**return** self.name

В представлениях мы просто создаем объект [sqlalchemy.orm.session.Session](http://docs.sqlalchemy.org/en/latest/orm/session_api.html" \l "sqlalchemy.orm.session.Session" \o "(в SQLAlchemy v1.2)) и работаем с объектами, как описано в документации [SQLAlchemy](http://www.sqlalchemy.org/). При этом в каждом представлении нам необходимо создавать новую SQLAlchemy сессию, а если были изменения подтверждать их при помощи метода [sqlalchemy.orm.session.Session.commit()](http://docs.sqlalchemy.org/en/latest/orm/session_api.html" \l "sqlalchemy.orm.session.Session.commit" \o "(в SQLAlchemy v1.2)).

Данный пример при каждом обновлении делает новую запись в БД и отдает их браузеру.

**from** **wsgiref.simple\_server** **import** make\_server

**from** **pyramid.config** **import** Configurator

**from** **pyramid.response** **import** Response

**from** **models** **import** User, Session, Base, engine

**def** hello(request):

DBSession = Session(bind=engine)

result = DBSession.query(User).all()

**import** **time**

timestamp = int(time.time())

new\_user = User(name=str(timestamp))

DBSession.add(new\_user)

DBSession.commit()

**return** Response(str(result))

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

Base.metadata.create\_all()

DBSession = Session(bind=engine)

DBSession.add(User(name='Vasya'))

DBSession.add(User(name='Petya'))

DBSession.commit()

config = Configurator()

config.add\_route('hello\_world', '/')

config.add\_view(hello, route\_name='hello\_world')

app = config.make\_wsgi\_app()

server = make\_server('0.0.0.0', 8000, app)

server.serve\_forever()

**ZopeTransactionExtension**

**transaction**

ZopeTransactionExtension это расширение для SQLAlchemy, которое привязывает сессии к универсальному менеджеру транзакций [transaction](http://zodb.readthedocs.org/en/latest/transactions.html).

Добавим его в пример:

*# models.py*

**from** **sqlalchemy** **import** Column, Integer, Text, create\_engine

**from** **sqlalchemy.ext.declarative** **import** declarative\_base

**from** **sqlalchemy.orm** **import** sessionmaker

**from** **zope.sqlalchemy** **import** ZopeTransactionExtension

engine = create\_engine('sqlite:///foo.db')

Session = sessionmaker(bind=engine,

extension=ZopeTransactionExtension())

Base = declarative\_base(bind=engine)

**class** **User**(Base):

\_\_tablename\_\_ = 'models'

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

name = Column(Text)

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

**return** self.name

Теперь вместо DBSession.commit, нужно использовать transaction.commit().

*# \_\_init\_\_.py*

**from** **wsgiref.simple\_server** **import** make\_server

**from** **pyramid.config** **import** Configurator

**from** **pyramid.response** **import** Response

**from** **models** **import** User, Session, Base, engine

**import** **transaction**

**def** hello(request):

DBSession = Session(bind=engine)

result = str(DBSession.query(User).all())

**import** **time**

timestamp = int(time.time())

new\_user = User(name=str(timestamp))

DBSession.add(new\_user)

transaction.commit()

**return** Response(result)

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

Base.metadata.create\_all()

DBSession = Session(bind=engine)

DBSession.add(User(name='Vasya'))

DBSession.add(User(name='Petya'))

transaction.commit()

config = Configurator()

config.add\_route('hello\_world', '/')

config.add\_view(hello, route\_name='hello\_world')

app = config.make\_wsgi\_app()

server = make\_server('0.0.0.0', 8000, app)

server.serve\_forever()

**transaction.abort**

Теперь мы используем общий, глобальный менеджер транзакций, который работает не только с SQLAlchemy но и со всеми модулями которые его поддерживают. Ниже пример сессии в которой одновременно участвуют [SQLAlchemy](http://www.sqlalchemy.org/) и [pyramid\_mailer](http://pyramid-mailer.readthedocs.org/en/latest/).

**from** **wsgiref.simple\_server** **import** make\_server

**from** **pyramid.config** **import** Configurator

**from** **pyramid.response** **import** Response

**from** **models** **import** User, Session, Base, engine

**import** **transaction**

**from** **pyramid\_mailer.message** **import** Message

message = Message(subject="hello world",

sender="example@yandex.ru",

recipients=["me@mail.ru"],

body="hello, me")

**def** hello(request):

DBSession = Session(bind=engine)

result = str(DBSession.query(User).all())

**import** **time**

timestamp = int(time.time())

new\_user = User(name=str(timestamp),

id=100500)

DBSession.add(new\_user)

**from** **pyramid\_mailer** **import** get\_mailer

mailer = get\_mailer(request)

mailer.send(message)

**try**:

transaction.commit()

**except** Exception **as** e:

transaction.abort()

**return** Response(str(e))

**return** Response(result)

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

Base.metadata.create\_all()

DBSession = Session(bind=engine)

DBSession.add(User(name='Vasya'))

DBSession.add(User(name='Petya'))

transaction.commit()

settings = {'mail.host': 'smtp.yandex.ru',

'mail.port': '465',

'mail.ssl': True,

'pyramid\_mailer.prefix': 'mail.',

'mail.username': 'example@yandex.ru',

'mail.password': 'example password'}

config = Configurator(settings=settings)

config.include('pyramid\_mailer')

config.add\_route('hello\_world', '/')

config.add\_view(hello, route\_name='hello\_world')

app = config.make\_wsgi\_app()

server = make\_server('0.0.0.0', 8000, app)

server.serve\_forever()

В этом примере hello записывает нового пользователя с id=100500 в базу данных и отправляет письмо на адрес. При первом обновлении страницы пользователь добавится в БД и отправится письмо. Однако при дальнейшем запуске будет выводить ошибку, так как пользователь с таким id уже существует.

**pyramid\_tm**

[pyramid\_tm](http://pyramid-tm.readthedocs.org/en/latest/) автоматически подтверждает транзакцию в каждом запросе. Если мы забыли написать transaction.commit(), то он все равно вызовется, при этом мы также можем вызывать его явно.

*# \_\_init\_\_.py*

**from** **wsgiref.simple\_server** **import** make\_server

**from** **pyramid.config** **import** Configurator

**from** **pyramid.response** **import** Response

**from** **models** **import** User, Session, Base, engine

**def** hello(request):

DBSession = Session(bind=engine)

result = str(DBSession.query(User).all())

**import** **time**

timestamp = int(time.time())

new\_user = User(name=str(timestamp))

DBSession.add(new\_user)

**return** Response(result)

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

Base.metadata.create\_all()

DBSession = Session(bind=engine)

DBSession.add(User(name='Vasya'))

DBSession.add(User(name='Petya'))

config = Configurator()

config.include('pyramid\_tm')

config.add\_route('hello\_world', '/')

config.add\_view(hello, route\_name='hello\_world')

app = config.make\_wsgi\_app()

server = make\_server('0.0.0.0', 8000, app)

server.serve\_forever()

**pyramid\_sqlalchemy**

[pyramid\_sqlalchemy](http://pyramid-sqlalchemy.readthedocs.org/) создает объект базового класса Base и сессии Session автоматически. Мы просто указываем строку подключения к БД в настройках и включаем модуль pyramid\_sqlalchemy в проект.

**from** **wsgiref.simple\_server** **import** make\_server

**from** **pyramid.config** **import** Configurator

**from** **pyramid.response** **import** Response

**from** **models** **import** User

**from** **pyramid\_sqlalchemy** **import** BaseObject, Session

**import** **transaction**

**def** hello(request):

result = str(Session.query(User).all())

**import** **time**

timestamp = int(time.time())

new\_user = User(name=str(timestamp))

Session.add(new\_user)

**return** Response(result)

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

settings = {'sqlalchemy.url': 'sqlite:///:memory:'}

config = Configurator(settings=settings)

config.include('pyramid\_tm')

config.include('pyramid\_sqlalchemy')

BaseObject.metadata.create\_all()

Session.add(User(name='Vasya'))

Session.add(User(name='Petya'))

transaction.commit()

config.add\_route('hello\_world', '/')

config.add\_view(hello, route\_name='hello\_world')

app = config.make\_wsgi\_app()

server = make\_server('0.0.0.0', 8000, app)

server.serve\_forever()

Файл с моделями теперь выглядит значительно проще.

**from** **sqlalchemy** **import** Column, Integer, Text

**from** **pyramid\_sqlalchemy** **import** BaseObject

**class** **User**(BaseObject):

\_\_tablename\_\_ = 'models'

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

name = Column(Text)

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

**return** self.name