Flask vol.2

В первой части

- Виртуальные окружения в Python
- Декораторы в Python
- Простейшее Flask-приложение
- Настройка маршрутов
- request, session, g
- Возможности шаблонизатора Jinja2
- Работа с ссылками и статичными файлами
- Организация каталога приложения

Flask-WTF

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired


class LoginForm(FlaskForm):
    user_login = StringField('Login')
    user_pass = PasswordField('Password')
    submit = SubmitField('Send')
```

```
@app.route('/', methods = ['GET', 'POST'])
def index():
        loginForm = LoginForm()
        if loginForm.validate_on_submit():
                return loginForm.user_login.data
        return render_template('index.html', loginForm = loginForm)
```

```
<form method="POST" action="/">
        {{loginForm.csrf_token }}
        <div class="form-field">{{ loginForm.user_login.label }} {{ loginForm.user_login }}</div>
        <div class="form-field">{{ loginForm.user_pass.label }} {{ loginForm.user_pass }}</div>
        {{ loginForm.submit }}
</form>
```

Типы полей

- BooleanField(default field arguments, false_values=None)
- DateField(default field arguments, format='%Y-%m-%d')
- DateTimeField(default field arguments, format='%Y-%m-%d %H:%M:%S')
- DateTimeLocalField(default field arguments, format='%Y-%m-%d %H:%M:%S')
- DecimalField(default field arguments, places=2, rounding=None, use_locale=False, number_format=None)
- EmailField(default field arguments)
- FileField(default field arguments)
- MultipleFileField(default field arguments)
- FloatField(default field arguments)
- IntegerField(default field arguments)
- IntegerRangeField(default field arguments)
- RecaptchaField(default field arguments)

- RadioField(default field arguments, choices=[], coerce=str)
- SelectField(default field arguments, choices=[], coerce=str, option_widget=None, validate_choice=True)
- SearchField(default field arguments)
- SelectMultipleField(default field arguments, choices=[], coerce=str, option_widget=None)
- SubmitField(default field arguments)
- StringField(default field arguments)
- TelField(default field arguments)
- TimeField(default field arguments, format='%H:%M')
- URLField(default field arguments)
- HiddenField(default field arguments)
- PasswordField(default field arguments)
- TextAreaField(default field arguments)
- FormField(form_class, default field arguments, separator='-')
- FieldList(unbound_field, default field arguments, min_entries=0, max_entries=None)

RecaptchaField

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired
from flask_wtf.recaptcha import RecaptchaField

app.config['RECAPTCHA_PUBLIC_KEY'] = '6LeYIbsSAAAAACRPIllxA7wvXjIE411PfdB2gt2J'
app.config['RECAPTCHA_PRIVATE_KEY'] = '6LeYIbsSAAAAAJezaIq3Ft_hSTo0YtyeFG-JgRtu'

class LoginForm(FlaskForm):
    user_login = StringField('Login')
    user_pass = PasswordField('Pssword')
    recaptcha = RecaptchaField()
    submit = SubmitField('Send')
```

default field arguments

- label – лейбл
- validators – валидаторы
- filters – фильтры
- description – описание поля
- id – идентификатор
- default – значение по умолчанию
- widget – виджет, используемый при рендеренге
- render_kw (dict) – словарь значений, передаваемых виджету при рендеренге

Валидаторы

- DataRequired(message=None)
- Email(message=None, granular_message=False, check_deliverability=False, allow_smtputf8=True, allow_empty_local=False)
- InputRequired(message=None)
- EqualTo(fieldname, message=None)
- IPAddress(ipv4=True, ipv6=False, message=None)
- Length(min=- 1, max=- 1, message=None)
- MacAddress(message=None)
- NumberRange(min=None, max=None, message=None)
- Optional(strip_whitespace=True)
- Regexp(regex, flags=0, message=None)
- URL(require_tld=True, message=None)
- UUID(message=None)
- AnyOf(values, message=None, values_formatter=None)
- NoneOf(values, message=None, values_formatter=None)

Custom validators

```
def length(min=-1, max=-1):
    message = 'Must be between %d and %d characters long.' % (min, max)

    def _length(form, field):
        l = field.data and len(field.data) or 0
        if l < min or max != -1 and l > max:
            raise ValidationError(message)

    return _length

class MyForm(Form):
    name = StringField('Name', [InputRequired(), length(max=50)])
```

Отправка писем

```
from flask_mail import Mail, Message

app = Flask(__name__)

app.config['MAIL_SERVER'] = 'smtp.googlemail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'test@gmail.com'
app.config['MAIL_DEFAULT_SENDER'] = 'test@gmail.com'
app.config['MAIL_PASSWORD'] = 'password'

mail = Mail(app)

msg = Message("Subject", sender="sender@example.com",  recipients=['recipient_1@example.com'])
msg.body = "Mail body"
msg.html = "<p>Mail body</p>"
mail.send(msg)
```

Blueprints (чертежы/схемы/планы/эскизы)

- Эскизы — способ организации приложений. Эскиз может иметь собственные функции представления, шаблоны и статические файлы, для них можно выбрать собственные URI.
- Основная концепция blueprint'ов заключается в том, что они записывают операции для выполнения при регистрации в приложении. Flask связывает функции представлений с blueprint'ами при обработке запросов и генерировании URL'ов от одной конечной точки к другой.

```
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

auth_bp = Blueprint('auth_bp', __name__,
                    template_folder='templates')

@auth_bp.route('/', defaults={'page': 'index'})
@auth_bp.route('/<page>')
def show(page):
    try:
        return render_template('pages/%s.html' % page)
    except TemplateNotFound:
        abort(404)
########################################
from flask import Flask, render_template
from flask import Blueprint, render_template, abort
from auth.auth import auth_bp

app = Flask(__name__)
app.register_blueprint(auth_bp)
app.run(debug = True,  host='127.0.0.1', port='5050')
```

Префиксы URL

- app.register_blueprint(simple_page, url_prefix='/pages')

- Если нужно вставить ссылку с одной страницы на другую, можно воспользоваться функцией url_for(), как обычно: нужно просто добавить к конечной точке URL'а префикс с именем blueprint'а и точкой (.):

  url_for('admin.index')

- Наконец, если в функции представления blueprint'а или в отрисованном шаблоне нужно добавить ссылку на другую конечную точку того же blueprint'а, можно воспользоваться относительным перенаправлением, добавив префикс, состоящий только из точки:

  url_for('.index')

Фабрика приложений

Файл инициализации:

```
def create_app(config):
    app = Flask(__name__)
    app.config.from_object(config)
    mail.init_app(app)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .admin import main as admin_blueprint
    app.register_blueprint(admin_blueprint)

    return app
```

Test Driven Development

- Основные правила:

  1. Пишите тест, который не проходит, до написания кода.

  2. Затем пишите код, который сможет пройти тест.

- Цикл “red, green, refactor”:

  1. Red: вы пишите провальный тест без написания кода.

  2. Green: пишите простейший код, который сможет пройти тест. Даже если код кажется вам наиглупейшим.

  3. Refactor: рефакторинг кода, если необходим. Не беспокойтесь, если вы поменяете код и ваши юнит-тесты сломаются, если что-то пойдет не так.

unittest

```
import unittest
from app import create_app
class Config(object):
    DEBUG = False
    TESTING = False
    SECRET_KEY = 'SecretKeyqwerty12345!!'
class DevelopmentConfig(Config):
    DATABASE_URI = 'mysql://user@localhost/foo2'
    DEBUG = True
class TestCase(unittest.TestCase):
    def setUp(self):
        self.app = create_app(DevelopmentConfig)
    def tearDown(self):
        print('Tear down')
    def test_probe(self):
        expected = 10
        res = 5 + 5
        assert  res == expected


if __name__ == '__main__':
    unittest.main()
```

Использование БД

- Конфигурация:

```
MYSQL_DATABASE_USER = 'sysdba'
MYSQL_DATABASE_PASSWORD = 'password'
MYSQL_DATABASE_DB = 'club_db'
MYSQL_DATABASE_HOST = 'localhost'
```

- Инициализация

```
mysql = MySQL()
mysql.init_app(app)
app.app_context().push()
with app.app_context():
        current_app.db_connection = mysql.connect()
```

- Запросы к БД

```
def show(page):
    cursor = current_app.db_connection.cursor()
    cursor.execute('SELECT user_name FROM user')
    users = []
    for user in cursor.fetchall():
        users.append({'name': user[0]})
    return jsonify(users)
```

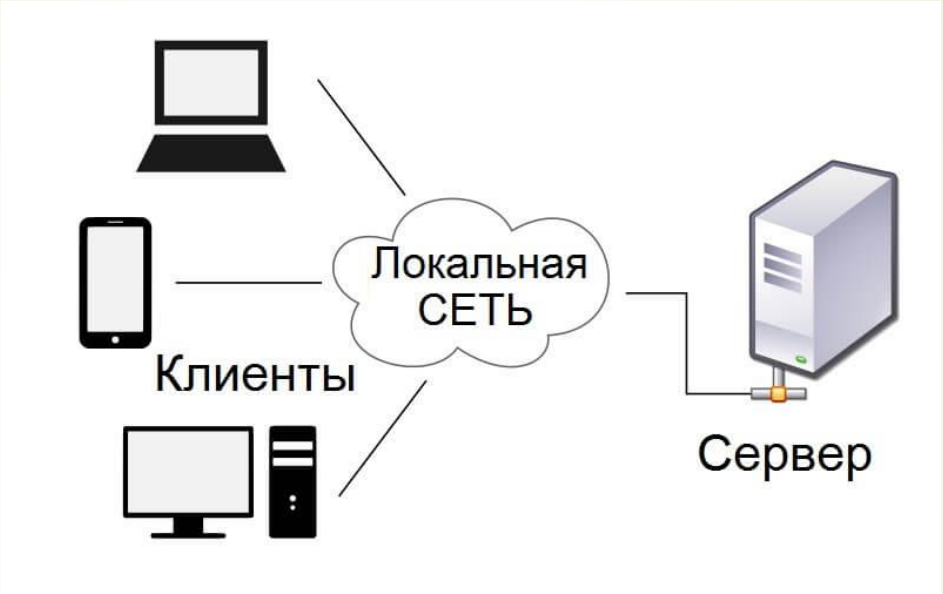
Структура приложения

```
/
    app/
        blueprint1/
            templates/
            static/
            __init__.py
            blueprint1.py
        static/
        templates/
        model.py
        __init__.py
    run.py
    test.py
```

Архитектуры приложений

- Монолитная архитектура
- Клиент-серверная архитектура
- Трехзвенная архитектура
- Peer2Peer
- Микросервисная архитектура
- Serverless
- Брокер сообщений

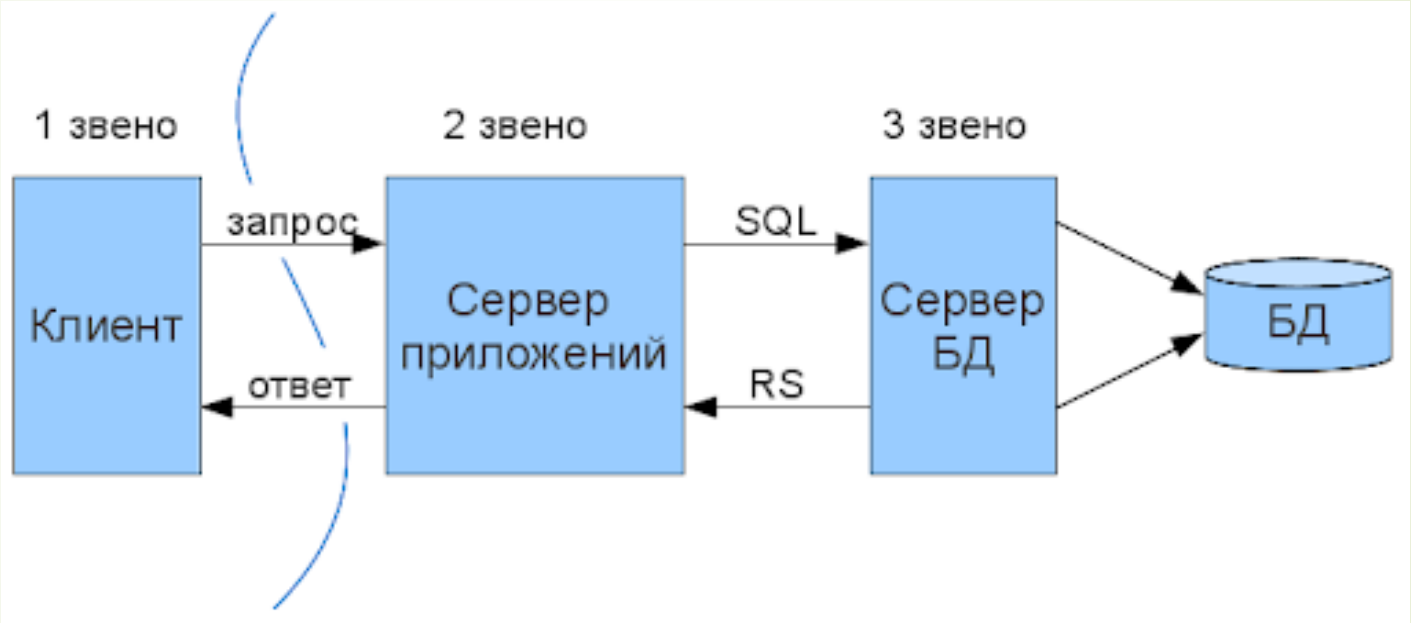
Архитектура клиент-сервер



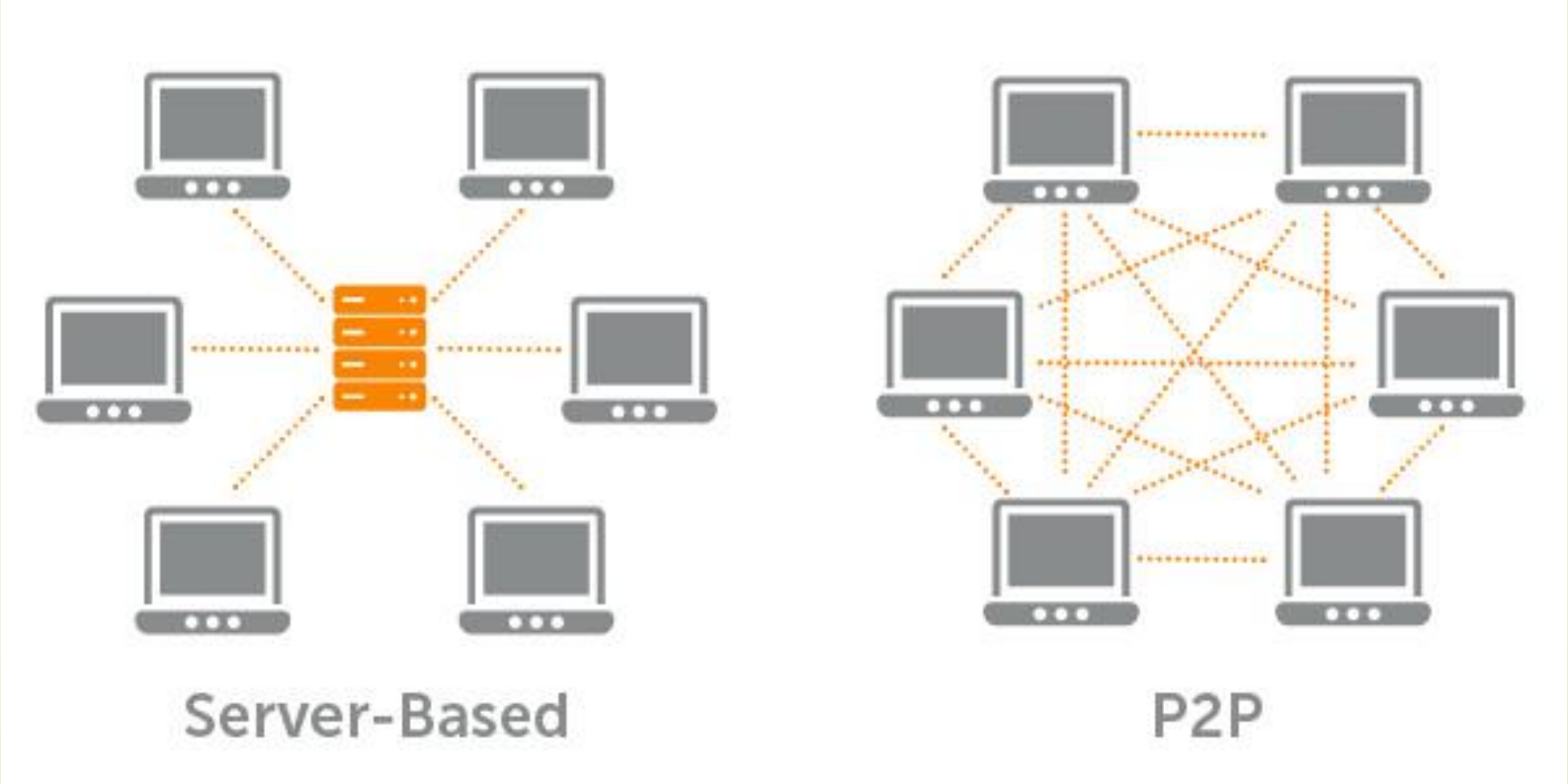
Виды клиентов:

- Толстый клиент
- Тонкий клиент

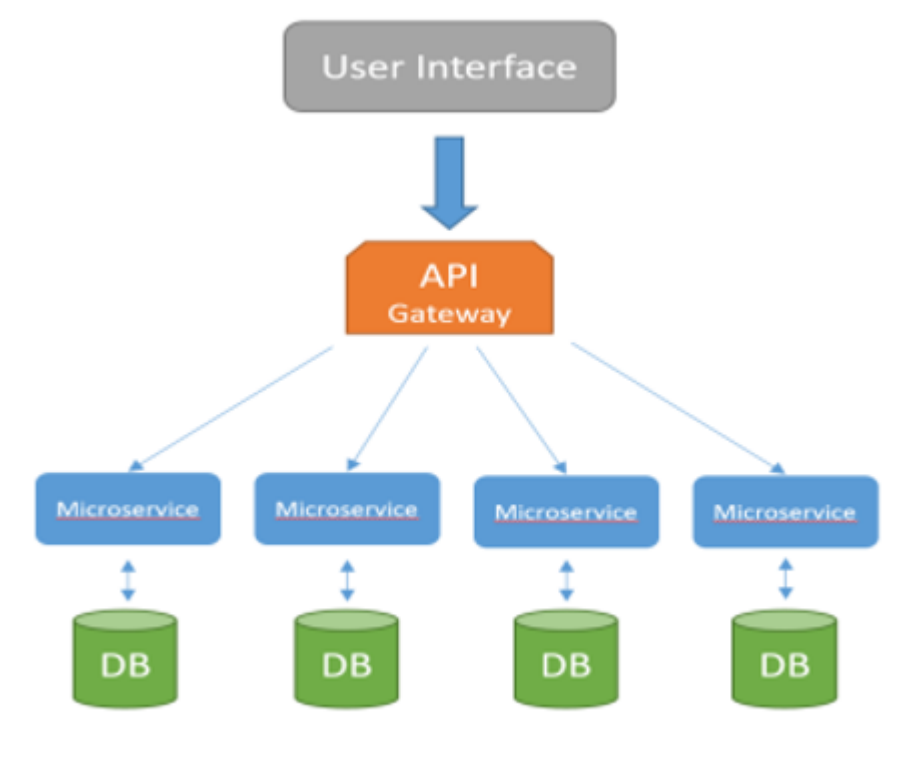
Трехзвенная архитектура



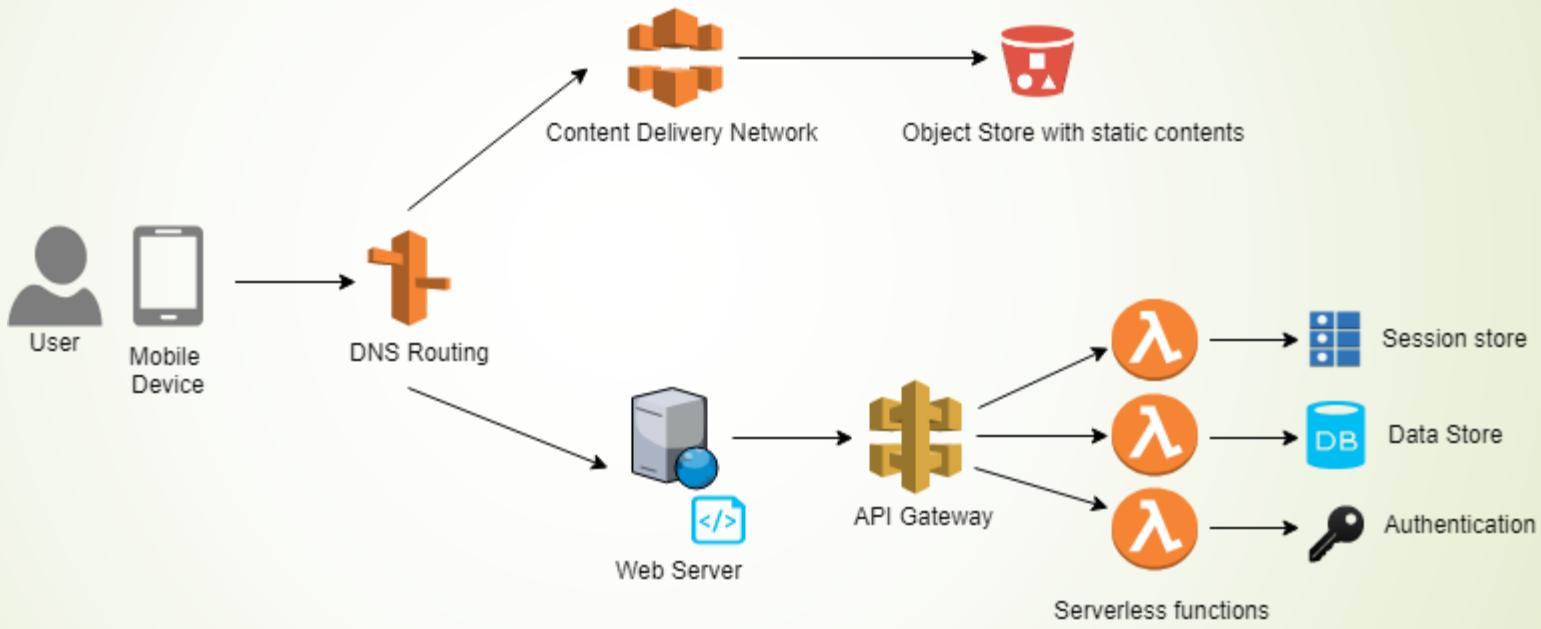
P2P



Архитектура на основе микросервисов



Serverless


Content Delivery Network
Object Store with static contents
User
Mobile Device
DNS Routing
Session store
Data Store
DB
Authentication
Web Server
API Gateway
Serverless functions

Брокер сообщений