Федеральное агентство связи

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

«Московский технический университет

связи и информатики»

Кафедра математической кибернетики и

информационных технологий

Отчет по учебной практике

«Разработка клиент-серверных приложений (клиент)»

Выполнил:

студент группы БФИ1701

Глушков И.В.

Научный руководитель:

зав. кафедры МКиИТ Городничев М. Г.

Москва 2019 г.

**Оглавление**

[Цель 3](#_Toc19732619)

[Задачи 3](#_Toc19732620)

[Анализ предметной области 3](#_Toc19732621)

[Проектирование 6](#_Toc19732622)

[Разработка 6](#_Toc19732623)

[Вывод 13](#_Toc19732624)

[Список источников 14](#_Toc19732625)

# **Цель**

Цель работы – разработать клиент-серверное приложение со стороны клиента, позволяющее пользователю осуществлять регистрацию, авторизацию, а также взаимодействовать с картой.

# **Задачи**

Поставленная цель определила следующие основные задачи:

1) Изучить принципы взаимодействия клиент-серверных приложений.

2) Разработать интерфейс приложения.

3) Разработать функцию авторизации пользователя.

4) Разработать функцию регистрации пользователя.

5) Разработать функцию взаимодействия с картой.

# **Анализ предметной области**

Клиент-сервер – это сетевая архитектура, в которой сетевая нагрузка распределена между поставщиком услуг (сервером) и заказчиком услуг (клиентом). Как правило, клиент и сервер какого-либо ресурса находятся на разных компьютерах, однако, они также могут находиться в пределах одной вычислительной системы.

Программа-клиент выполняет функции интерфейса, с помощью которого пользователь может отправлять запросы на сервер для получения необходимых данных.

Также стоит отметить, что к одному серверу могут обращаться сразу несколько клиентов. Как правило, количество подключаемых клиентов зависит от вычислительной мощности сервера.

В данной архитектуре взаимодействие между поставщиком и заказчиком услуг происходит в сети интернет или в других компьютерных сетях при помощи сетевых протоколов, например, с помощью IP протокола, FTP протокола, HTTP протокола и других. Например, при помощи HTTP протокола клиент отправляет на сервер сообщение о том, какую информацию и в каком виде он хочет ее получить.

Данные сообщения, посылаемые клиентом, получили название HTTP запросов. В запросах указаны специальные методы, говорящие серверу как обработать сообщение. Сообщения, посылаемые сервером, получили название HTTP ответы. В их структуре присутствуют специальные коды состояния, позволяющие браузеру проанализировать то, как сервер принял запрос от клиента.

Клиент-серверная архитектура имеет видимые плюсы и минусы. К плюсам можно отнести то, что все вычислительные операции производятся на сервере и поэтому нагрузка на компьютер клиента будет минимальной, что способствует снижению требований к вычислительной мощности клиента. Однако стоит отметить, что неработоспособность сервера может вывести из строя всю вычислительную сеть. Также обеспечение постоянной работы сервера требует дополнительной рабочей силы – системного администратора, который будет следить за состоянием сети.

Выделяют два типа работы компьютеров во взаимодействии клиента и сервера: классическая двухуровневая архитектура и трехуровневая архитектура.

Пример двухуровневой архитектуры приведен на рисунке 1.



Рисунок 1 – Пример взаимодействия компьютеров в двухуровневой архитектуре

Данная архитектура используется в клиент-серверных системах, где сервер самостоятельно отвечает на клиентские запросы, используя при этом только собственные вычислительные ресурсы.

Трехуровневая архитектура отличается от двухуровневой тем, что в ней отдельно выносится сервер баз данных (рис. 2).



Рисунок 2 – Пример взаимодействия компьютеров в трехуровневой архитектуре

Для реализации клиент-серверной архитектуры также широко используется стиль взаимодействия REST (Representational State Transfer). В основе REST лежит несколько базовых операций, таких как извлечение данных (GET), сохранение (POST), изменение (PUT) и удаление (DELETE). Обязательными условиями для реализации данного подхода являются следующие требования:

– модель клиент-сервер;

– отсутствие состояния;

– кэширование;

– единообразие интерфейса;

– слои;

– код по требованию (необязательное требование).

У архитектурного стиля взаимодействия REST можно выделить следующие преимущества: надежность, высокая производительность, масштабируемость, легкость внесения изменений и простота интерфейсов.

# **Проектирование**

Для разработки клиент-серверного приложения был использован язык программирования Python 3.7.3, фреймворк Django 2.2.3, среда программирования PyCharm 2019.1.3, библиотеки django-allauth 0.39.1, virtualenv 16.6.1.

Языком разработки был выбран именно Python из-за возможности подключения большого количества разнообразных библиотек и взаимодействия с фреймворками, подходящих для реализации поставленных задач.

Среда разработки PyCharm уже долгое время считается лучшей интегрированной средой для языка программирования Python. PyCharm предоставляет средства для анализа кода, отладчик и поддерживает веб-разработку на Django.

Django является свободным фреймворком для веб-приложений на языке Python, который использует шаблон проектирования MVC. Для данного фреймворка разработано множество подключаемых модулей, с помощью которых было проще разработать приложение, соответствующее поставленным задачам.

# **Разработка**

Текст settings.py

**import** os  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
SECRET\_KEY = **'dazuv%&-z&nnd84z!j3&2=mj^n7et#ek%q$%@di^y!pzr\_#p2o'**DEBUG = **True**ALLOWED\_HOSTS = []  
INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'django.contrib.sites'**,  
 **'registr'**,  
 **'allauth'**,  
 **'allauth.account'**,  
]  
  
MIDDLEWARE = [  
 **'django.middleware.security.SecurityMiddleware'**,  
 **'django.contrib.sessions.middleware.SessionMiddleware'**,  
 **'django.middleware.common.CommonMiddleware'**,  
 **'django.middleware.csrf.CsrfViewMiddleware'**,  
 **'django.contrib.auth.middleware.AuthenticationMiddleware'**,  
 **'django.contrib.messages.middleware.MessageMiddleware'**,  
 **'django.middleware.clickjacking.XFrameOptionsMiddleware'**,  
]  
  
ROOT\_URLCONF = **'practice.urls'**TEMPLATES = [  
 {  
 **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  
 **'DIRS'**: [**'./templates'**,],  
 **'APP\_DIRS'**: **True**,  
 **'OPTIONS'**: {  
 **'context\_processors'**: [  
 **'django.template.context\_processors.debug'**,  
 **'django.template.context\_processors.request'**,  
 **'django.contrib.auth.context\_processors.auth'**,  
 **'django.contrib.messages.context\_processors.messages'**,  
 **'django.template.context\_processors.request'**,  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = **'practice.wsgi.application'**CACHES = {  
 **'default'**: {  
 **'BACKEND'**: **'django.core.cache.backends.memcached.MemcachedCache'**,  
 **'LOCATION'**: **'127.0.0.1:8000'**,  
 }  
}  
  
AUTHENTICATION\_BACKENDS = (  
**'django.contrib.auth.backends.ModelBackend'**,  
  
**'allauth.account.auth\_backends.AuthenticationBackend'**,  
)  
DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.sqlite3'**,  
 **'NAME'**: os.path.join(BASE\_DIR, **'db.sqlite3'**),  
 }  
}  
  
EMAIL\_BACKEND = **'django.core.mail.backends.smtp.EmailBackend'**AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,  
 },  
]LANGUAGE\_CODE = **'ru-ru'**TIME\_ZONE = **'Europe/Moscow'**USE\_I18N = **True**USE\_L10N = **True**USE\_TZ = **True**STATIC\_URL = **'/static/'**LOGIN\_REDIRECT\_URL = **'/'**SITE\_ID = 1

Текст practice/urls.py

**from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** django.urls **import** include  
**from** django.views.generic **import** RedirectView  
**from** django.conf **import** settings  
**from** django.conf.urls.static **import** static  
**from** django.conf.urls **import** url  
  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
]  
  
urlpatterns += [  
 path(**'home/'**, include(**'registr.urls'**)),  
]  
  
urlpatterns += [  
 path(**''**, RedirectView.as\_view(url=**'/home/'**, permanent=**True**)),  
]  
  
urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)  
  
urlpatterns += [  
 url(**r'^accounts/'**, include(**'django.contrib.auth.urls'**)),  
 url(**r'^accounts/'**, include(**'allauth.urls'**)),  
]

Текст styles.css

.sidebar-nav  
{  
 margin-top: 20px;  
 padding: 0;  
 list-style: none;  
}  
.logreg  
{  
 display: block;  
 float: left;  
 margin-right: 30px;  
 padding: 10px 50px;  
 color: black;  
 text-decoration: none;  
 border-radius: 15px;  
 border: 1px solid black;  
 transition: color 0.5s, background 0.5s;  
}  
  
.logreg:hover  
{  
 text-decoration: none;  
 color: white;  
 background: blue;  
 border: 1px solid blue;  
}

Текст base\_generic.html

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
  
 {% block title %}<**title**>Page</**title**>{% endblock %}  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"**>  
 <**script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"**></**script**>  
 <**script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"**></**script**>  
 {% load static %}  
 <**link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}"**>  
  
</**head**>  
  
<**body**>  
  
 <**div class="container-fluid"**>  
  
 <**div class="row"**>  
 <**div class="col-sm-2"**>  
 {% block sidebar %}  
 <**ul class="sidebar-nav"**>  
 <**li**><**a href="{% url 'index' %}"**>Главная</**a**></**li**>  
 {% if user.is\_authenticated %}  
 <**li**>Пользователь: {{ user.get\_username }}</**li**>  
 <**li**><**a href="{% url 'logout'%}?next={{request.path}}"**>Выйти</**a**></**li**>  
 {% else %}  
 <**li**><**a href="{% url 'login'%}?next={{request.path}}"**>Авторизация</**a**></**li**>  
 {% endif %}  
 </**ul**>  
 {% endblock %}  
 </**div**>  
 <**div class="col-sm-10 "**>  
 {% block content %}{% endblock %}  
 </**div**>  
 </**div**>  
  
 </**div**>  
</**body**>  
</**html**>

Текст index.html

{% extends "base\_generic.html" %}  
  
{% block content %}  
<**h1**>Домашняя страница</**h1**>  
{% if user.is\_authenticated %}  
<**h3**>Добро пожаловать, {{ user.get\_username }}</**h3**>  
<**iframe src="https://www.google.com/maps/embed?pb=!1m10!1m8!1m3!1d141646.2946730079!2d37.63123165814494!3d55.72821710552029!3m2!1i1024!2i768!4f13.1!5e0!3m2!1sru!2sru!4v1568545840189!5m2!1sru!2sru" width="600" height="450" frameborder="0" style="**border:0;**" allowfullscreen=""**></**iframe**>  
{% else %}  
<**a href="{% url 'login'%}?next={{request.path}}" class="logreg"**>Авторизация</**a**>  
<**a href="{% url 'account\_signup' %}" class="logreg"**>Регистрация</**a**>  
{% endif %}  
{% endblock %}

Текст apps.py

**from** django.apps **import** AppConfig  
  
  
**class** RegistrConfig(AppConfig):  
 name = **'registr'**

Текст registr/urls.py

**from** django.urls **import** path  
**from** . **import** views  
  
  
urlpatterns = [  
 path(**''**, views.index, name=**'index'**),  
]

Текст views.py

**from** django.shortcuts **import** render  
  
*# Create your views here.***def** index(request):  
 **return** render(request, **'index.html'**)

Текст logged\_out.html

{% extends "base\_generic.html" %}  
  
{% block content %}  
<**p**>Вы вышли из аккаунта</**p**>  
  
<**a href="{% url 'login'%}"**>Кликните здесь, чтобы авторизоваться.</**a**>  
{% endblock %}

Текст login.html

{% extends "base\_generic.html" %}  
  
{% block content %}  
  
<**form method="post" action="{% url 'login' %}"**>  
{% csrf\_token %}  
  
<**h1**>Авторизация</**h1**>  
  
<**div**>  
 <**td**>{{ form.username.label\_tag }}</**td**>  
 <**td**>{{ form.username }}</**td**>  
</**div**>  
<**div**>  
 <**td**>{{ form.password.label\_tag }}</**td**>  
 <**td**>{{ form.password }}</**td**>  
</**div**>  
  
<**div**>  
 <**input type="submit" value="Войти"** />  
 <**input type="hidden" name="next" value="{{ next }}"** />  
</**div**>  
</**form**>  
  
{% endblock %}

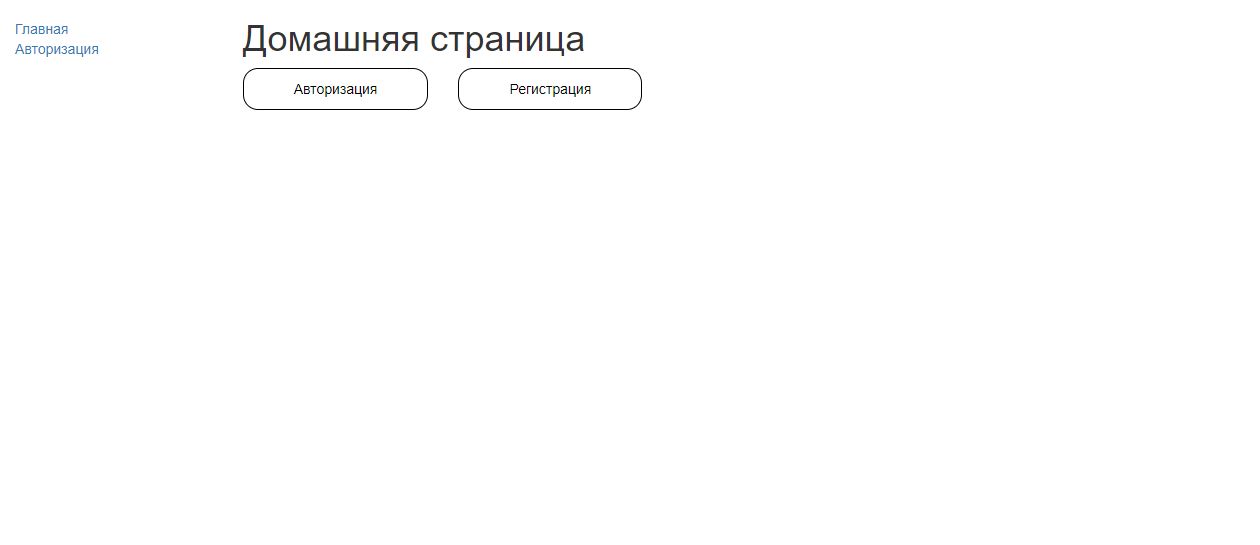


Рисунок 3 – Главная страница

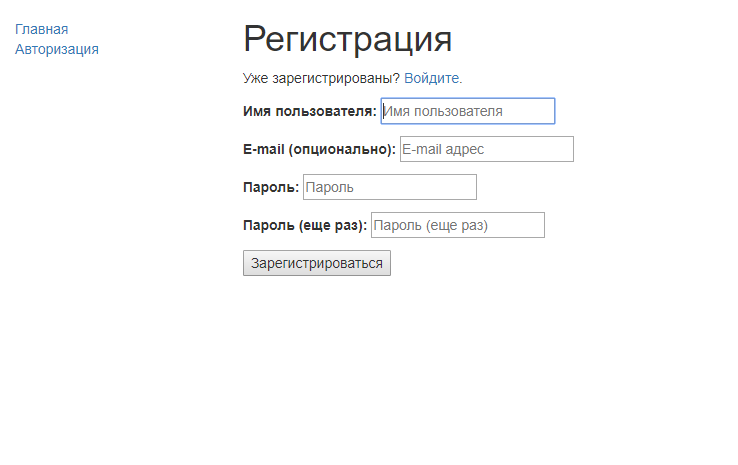


Рисунок 4 – Страница регистрации

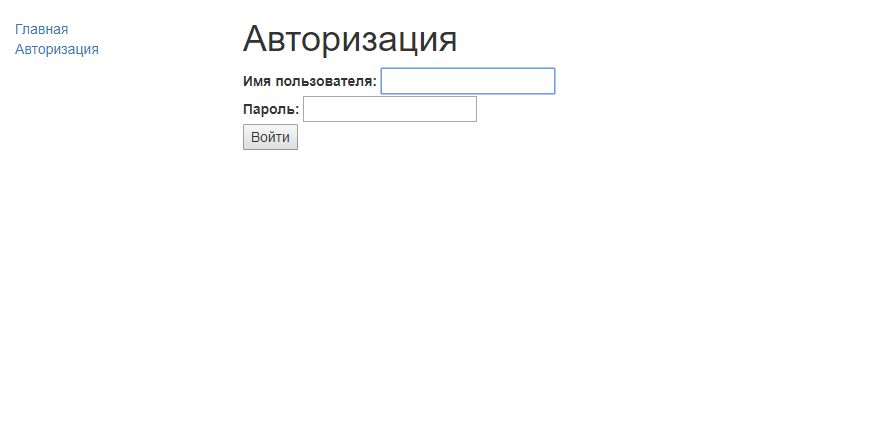


Рисунок 5 – Страница авторизации

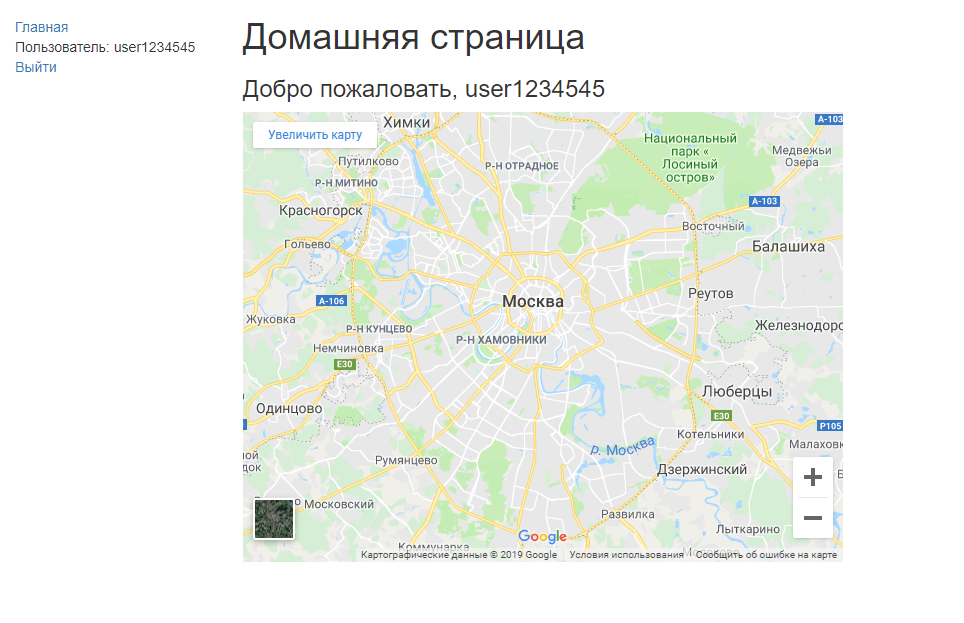


Рисунок 6 – Главная страница с авторизованным пользователем

# **Вывод**

В ходе выполнения учебной практики был кратко изучен материал о структуре клиент-серверной архитектуры, ее типах, взаимодействии компьютеров внутри данной архитектуры, способах реализации, а также было разработано клиент-серверное приложение со стороны клиента.

# **Список источников**

1) В. А. Дронов. – Django 2.1. Практика создания веб-сайтов на Python. – СПб: БХВ-Петербург, 2019. – 672 с. – 978-5-9775-4058-2.

2) М. Лутц. – Изучаем Python, 4-е издание. – Москва: Символ-Плюс, 2011. – 1280 с. – 978-5-93286-159-2.

3) Документация Django. <https://docs.djangoproject.com/en/2.2/>.

4) Руководство по разработке проекта с помощью Django. <https://developer.mozilla.org/ru/docs/Learn/Server-side/Django>.

5) Документация библиотеки django-allauth. <https://django-allauth.readthedocs.io/en/latest/installation.html>