**Проектная документация**

**Разработчики**

Сергеев Михаил Дмитриевич

Колтовой Илья Максимович

Карпухин Кирилл Давидович

Сысоев Дмитрий Алексеевич

**Цель работы:** Разработать программный продукт, позволяющий производить работу с документами компании «ИДИС ИБ».

**Средства:** Python 3.11.0, PyCharm 2022.2.3, MongoDB, Flask. Сысоев Дмитрий Алексеевич

**Дополнительные средства:**

|  |  |
| --- | --- |
| **Встроенные библиотеки** | **Сторонние библиотеки** |
| **-** | **Flask** |
|  | **MongoDB** |

**СУБД -**Система управления базами данных.

**Use-case-**Диаграмма использования программного продукта.

**ПП-**Программный продукт.

**MongoDB**-Документно-ориентированная база данных.

**Backend-** написание логики какого либо программного продукта или же сервера.

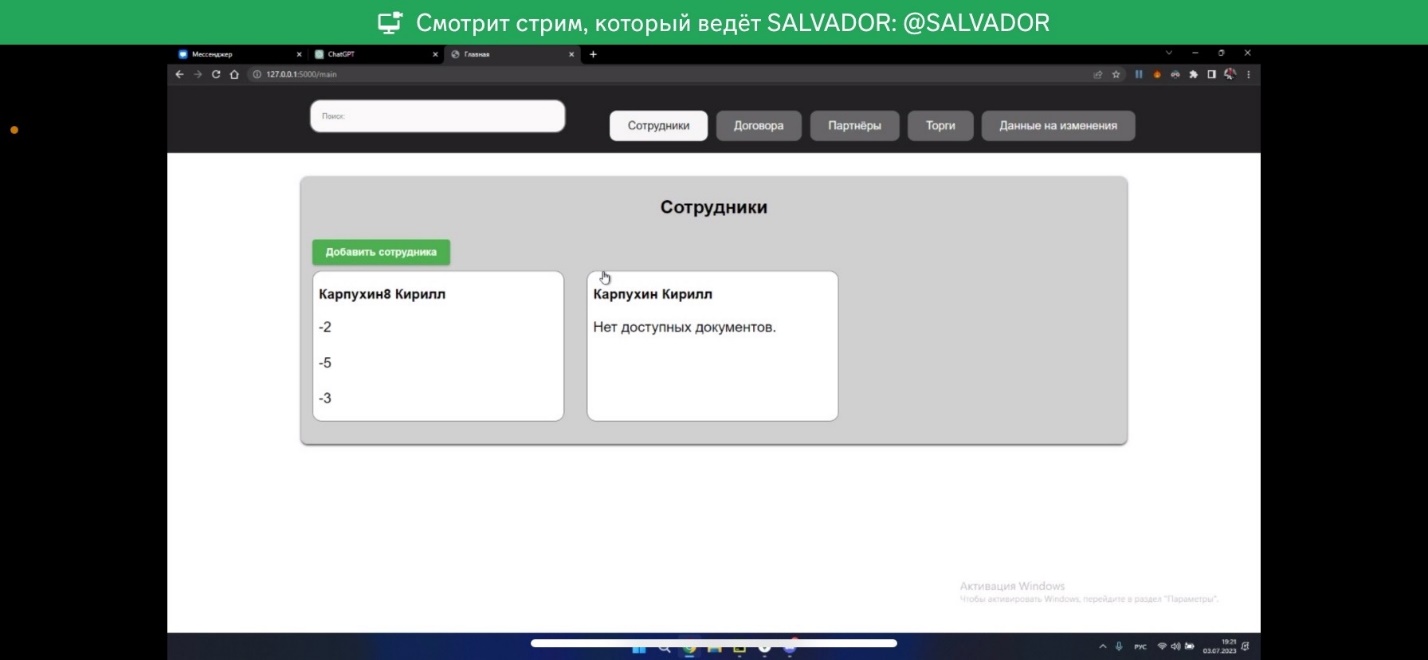
**web-приложение-** программным продукт построенный на языке программирования и реализованный с помощью web-технологий (протоколов, адресов и запросах).

**Router-** часть программы отвечающая за отображения компонента на форме dock.

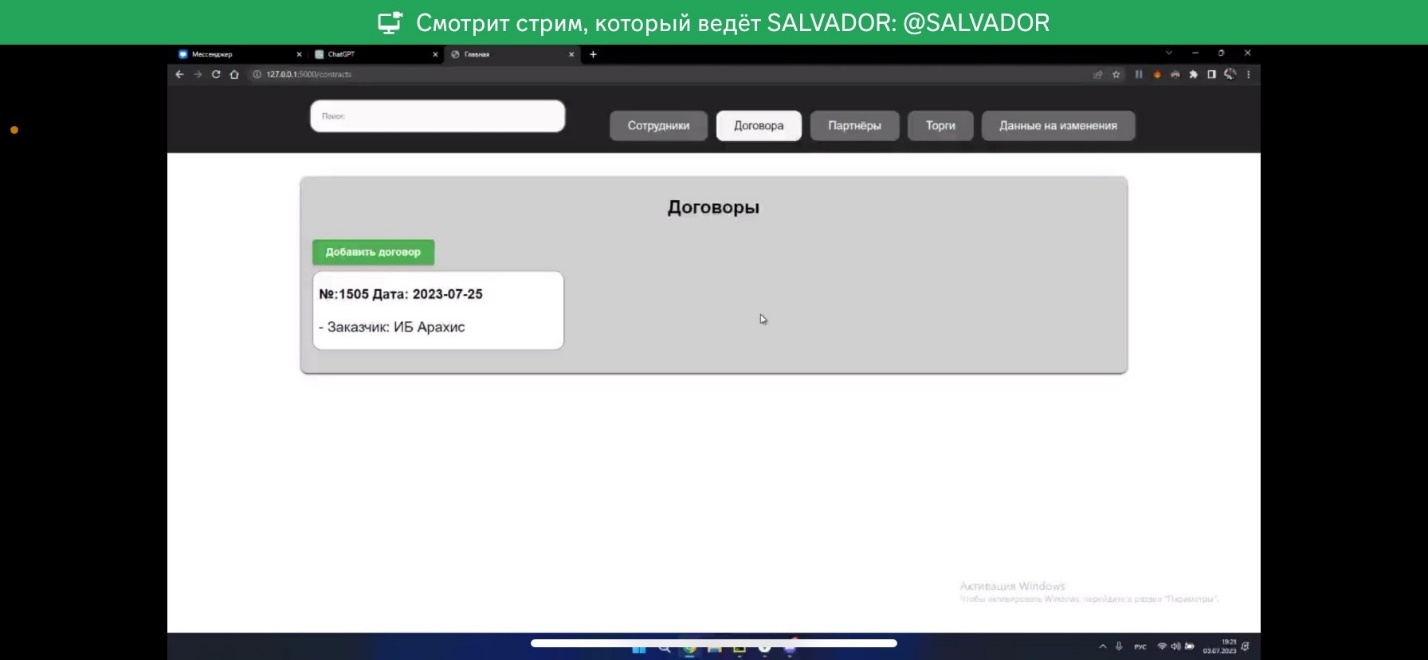
**Секция файла-** это подразделение файлов, в зависимости от их технической принадлежности.

**Описание функционала приложения**

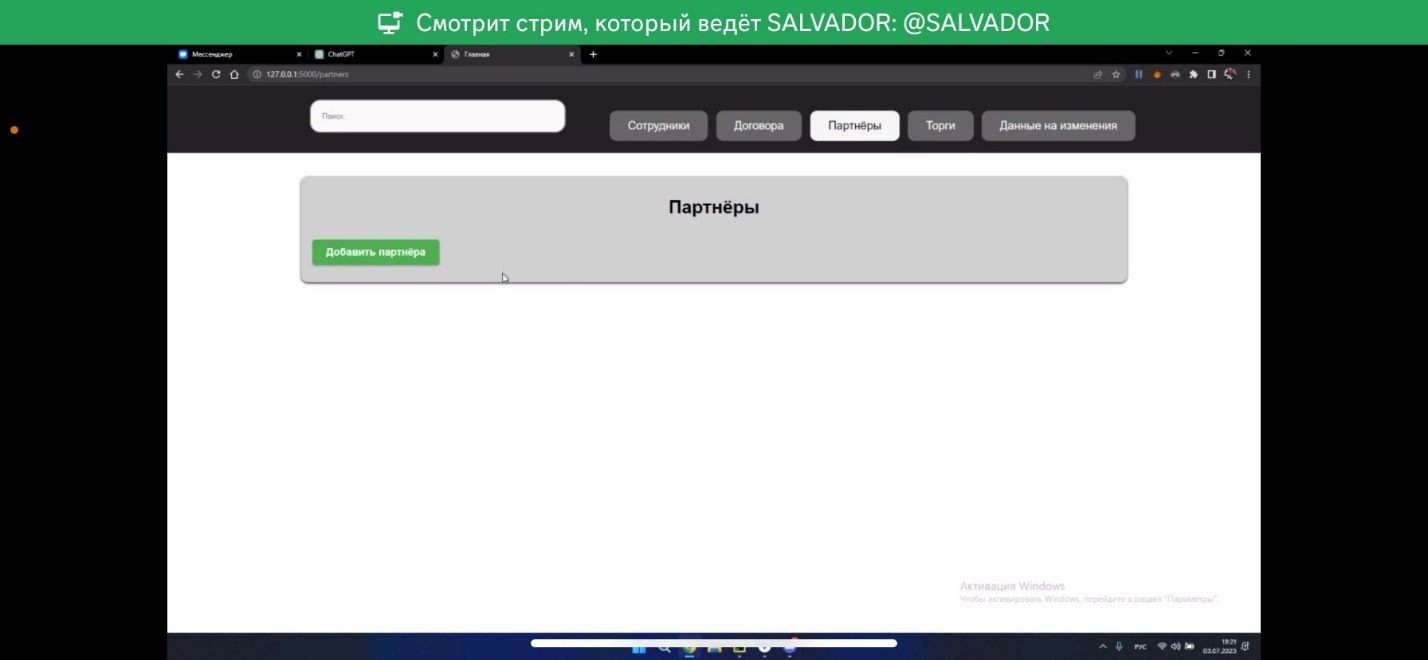
1. Веб-приложение представляет из себя набор вкладок, через которые осуществляется основная работа.

****

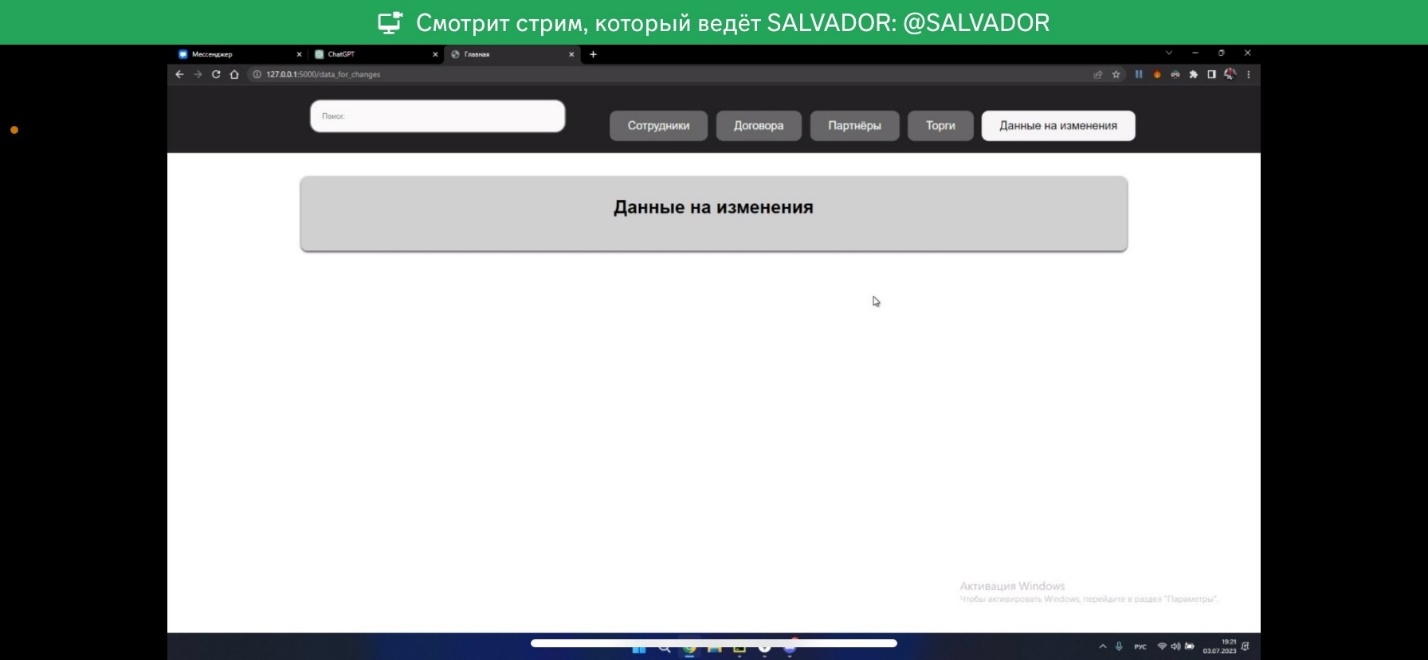
**Рисунок 1- Вкладки сайта**



**Рисунок 2- Вкладки сайта**

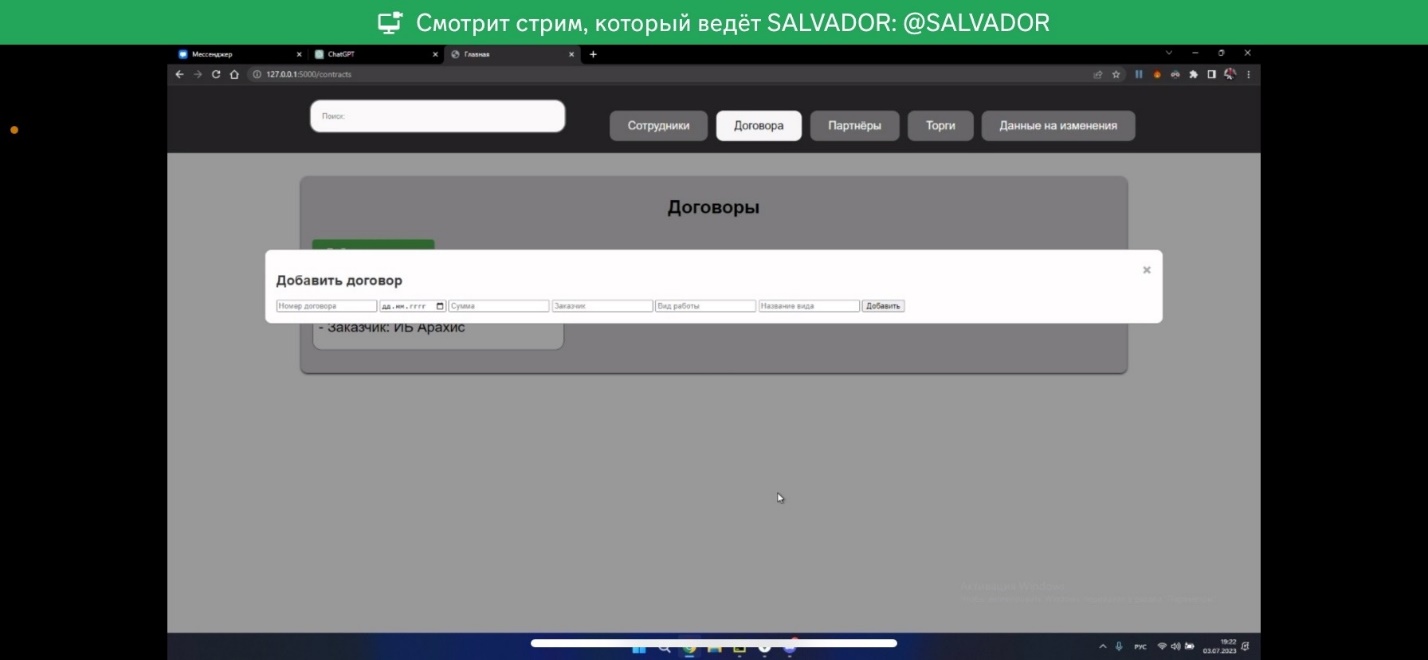


**Рисунок 3- Вкладки сайта**



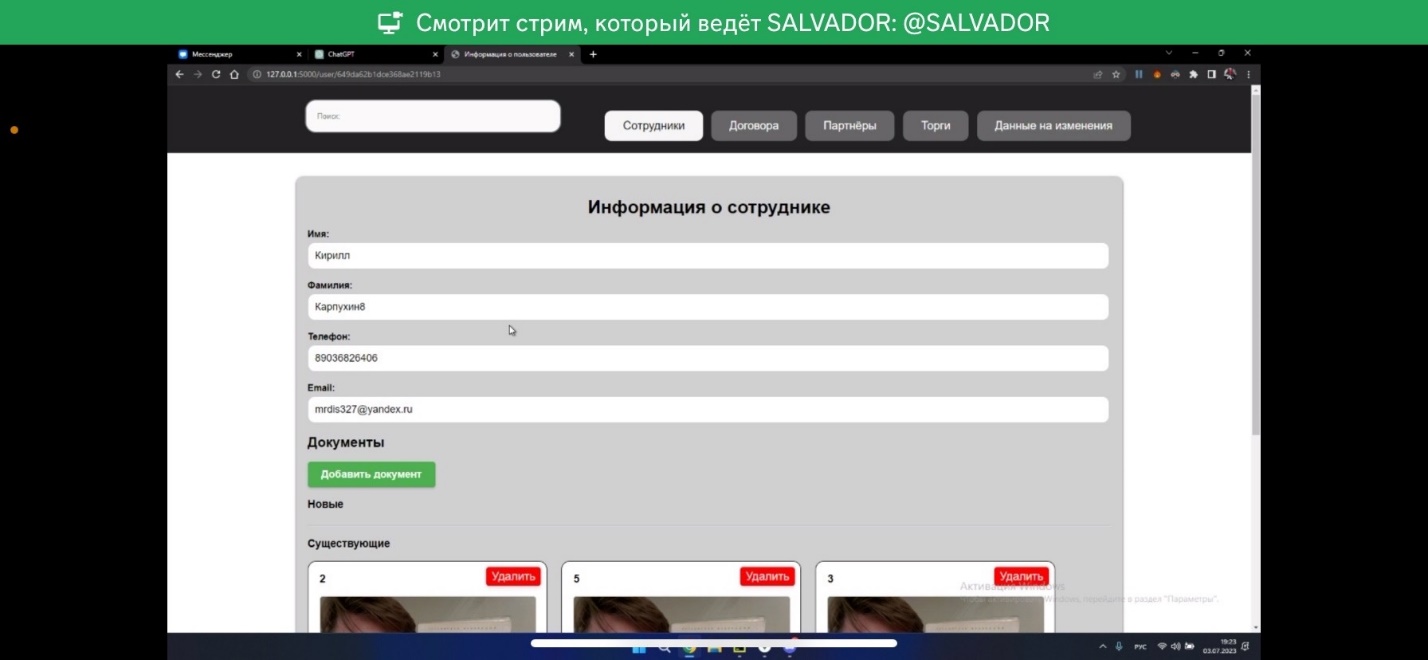
**Рисунок 4- Вкладки сайта**

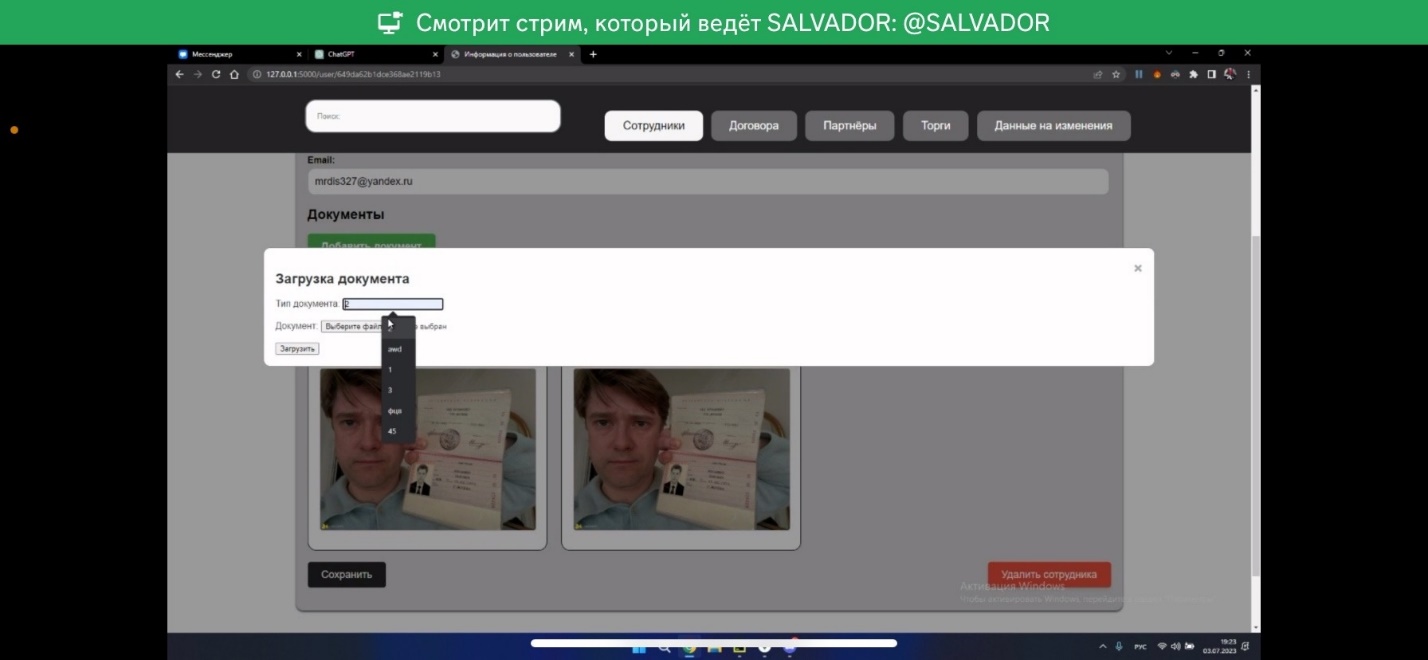
**Добавление Договоров**

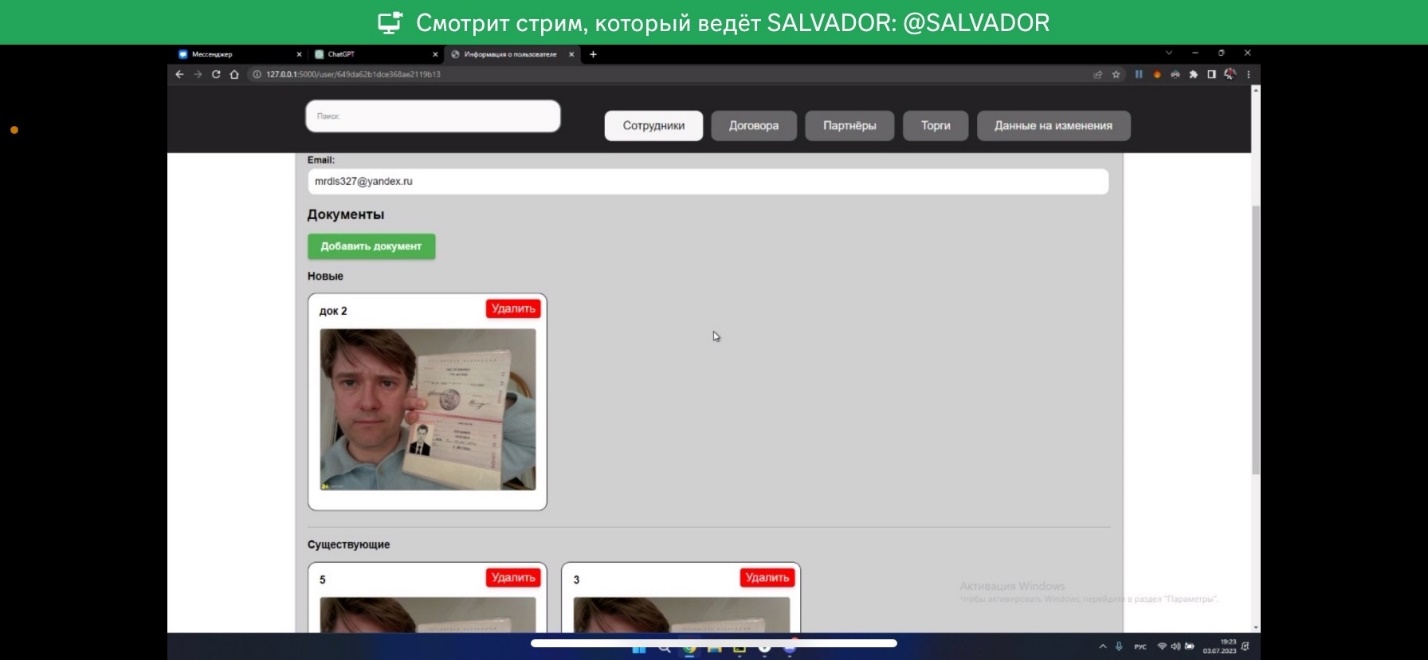


**Рисунок 5- добавление нового договора**

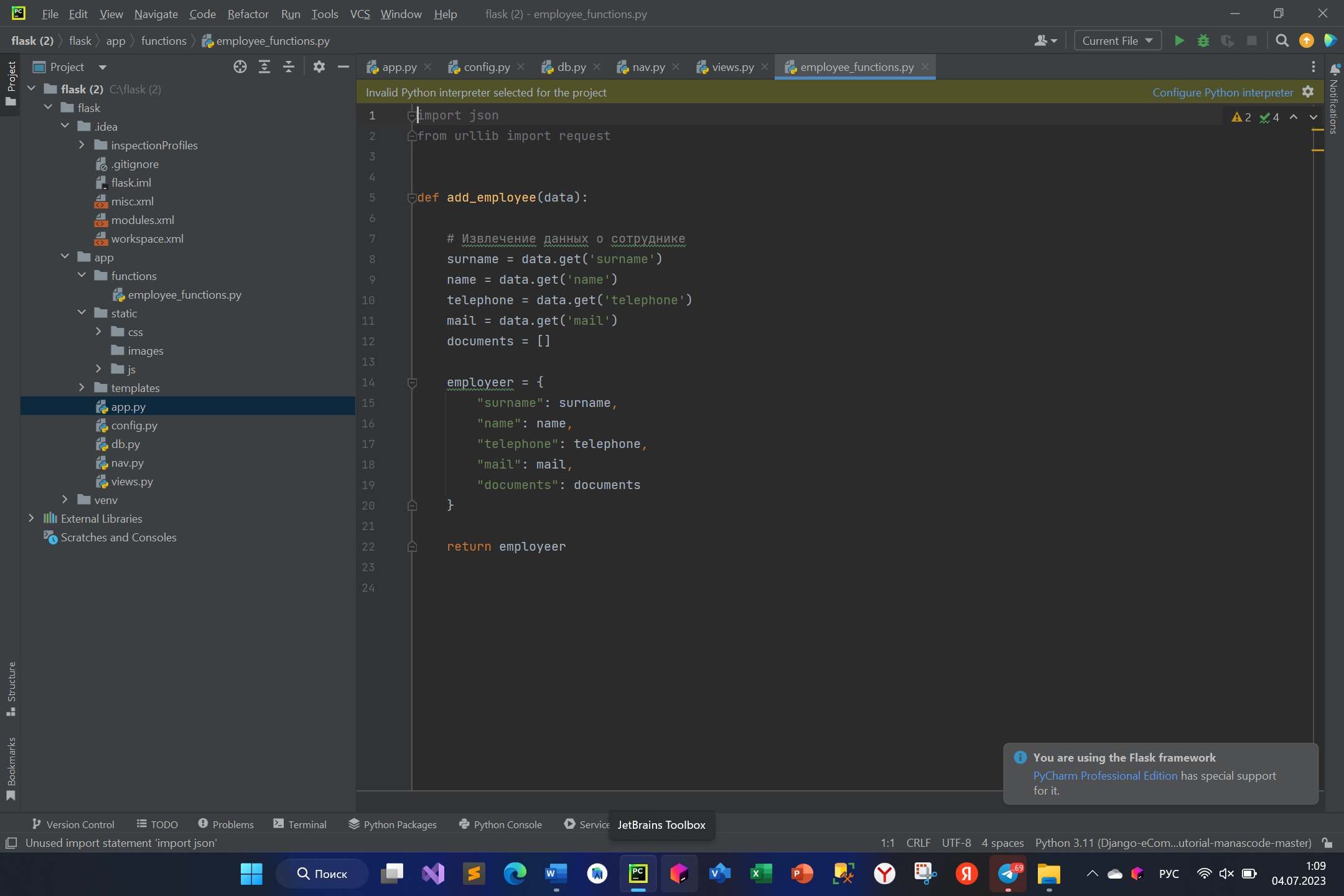
**Работа с данными о сотрудниках**







**Структура приложения**



**Файловая структура приложения**

Разработанное приложение имеет возможность работать с документами, а именно: добавить новый документ к определенному сотруднику, удалить определенный документ у сотрудника.

Так же работа с данными сотрудников реализована таким образом: добавлением нового сотрудника, удалением нового сотрудника и изменением данных о сотрудниках.

1. **Код app.py**

from views import \*  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

1. **Код Config.py**

import os  
  
  
SECRET\_KEY = ""

1. **Код DB.py**

from pymongo import MongoClient  
  
  
# Подключение к MongoDB  
client = MongoClient('localhost', 27017)  
db = client['Documents']  
  
collection = db['Employees']  
collection\_contracts = db['Contracts']  
collection\_partners = db['Partners']

1. **Код Nav.py**

from bson import ObjectId  
from flask import Flask  
from flask import render\_template  
from db import collection, collection\_contracts  
  
app = Flask(\_\_name\_\_)  
  
  
@app.route('/')  
def index():  
 # Получение данных из базы данных  
 users = collection.find()  
  
 # Передача данных в шаблон для отображения  
 return render\_template('main.html', users=users)  
  
# Страница "Сотрудники"  
@app.route('/main')  
def employees():  
 users = collection.find()  
  
 # Передача данных в шаблон для отображения  
 return render\_template('main.html', users=users)  
  
  
# Страница "Договора"  
@app.route('/contracts')  
def contracts():  
 contracts = collection\_contracts.find()  
  
 return render\_template('contracts.html', contracts=contracts)  
  
  
# Страница "Партнёры"  
@app.route('/partners')  
def partners():  
 # Дополнительная логика для страницы "Партнёры"  
 # ...  
  
 return render\_template('partners.html')  
  
  
# Страница "Торги"  
@app.route('/trading')  
def trading():  
 # Дополнительная логика для страницы "Торги"  
 # ...  
  
 return render\_template('trading.html')  
  
  
# Страница "Данные на изменения"  
@app.route('/data\_for\_changes')  
def data\_for\_changes():  
 # Дополнительная логика для страницы "Данные на изменения"  
 # ...  
  
 return render\_template('data\_for\_changes.html')

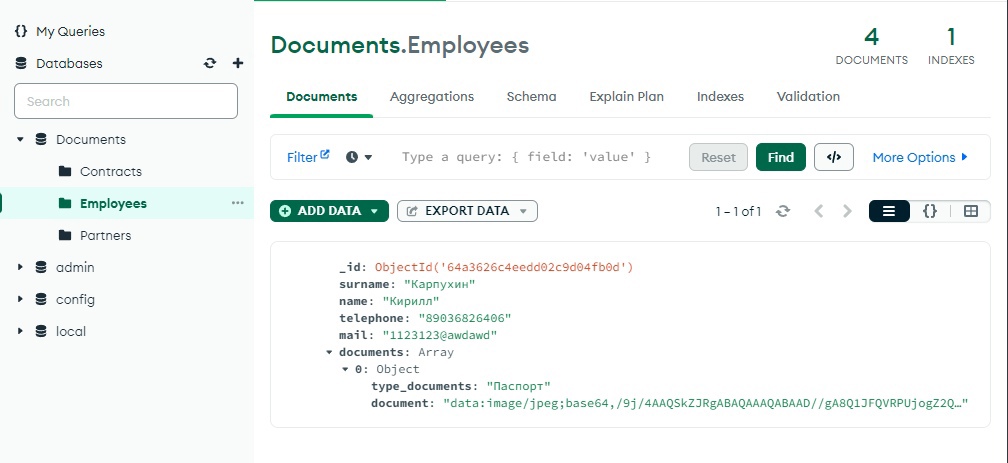
1. **Код Views.py**

from nav import \*  
from db import collection, collection\_contracts, collection\_partners  
from flask import request, redirect, json, render\_template  
from functions import employee\_functions  
  
  
############################## <!----! Пользователи !----!>  
  
@app.route('/user/<user\_id>')  
def user(user\_id):  
 # Поиск пользователя по object \_id  
 user = collection.find\_one({'\_id': ObjectId(user\_id)})  
  
 if user:  
 return render\_template('user.html', user=user)  
 else:  
 return 'Пользователь не найден'  
  
  
@app.route('/user/<user\_id>/remove', methods=['POST'])  
def remove\_employee(user\_id):  
 # Преобразуем строковый user\_id в ObjectId  
 employee\_id = ObjectId(user\_id)  
 # Удаляем пользователя  
 collection.delete\_one({'\_id': employee\_id})  
 users = collection.find()  
 return render\_template('main.html', users=users)  
  
  
@app.route('/add\_employee', methods=['POST'])  
def add\_employee():  
 data = request.form # Получение данных из POST-запроса  
 # Кидаем в базу  
 collection.insert\_one(employee\_functions.add\_employee(data)).inserted\_id  
 users = collection.find()  
 return render\_template('main.html', users=users)  
  
  
@app.route('/user/<user\_id>/edit', methods=['POST'])  
def edit\_user\_save(user\_id):  
  
 name = request.form['name']  
 surname = request.form['surname']  
 telephone = request.form['telephone']  
 mail = request.form['mail']  
  
 # Получение данных о новых документах  
 new\_documents = request.form.getlist('new\_documents')  
 # Декодирование JSON и обработка каждого документа  
 decoded\_documents = []  
 for document\_str in new\_documents:  
 document\_data = json.loads(document\_str)  
 for document in document\_data:  
 decoded\_documents.append(document)  
  
 # Получение данных о существующих документах  
 existing\_documents = request.form.getlist('existing\_documents')  
 for existing\_documents\_str in existing\_documents:  
 document\_data = json.loads(existing\_documents\_str)  
 for document in document\_data:  
 decoded\_documents.append(document)  
  
 # Обновляем пользователя  
 collection.update\_one(  
 {'\_id': ObjectId(user\_id)},  
 {'$set': {  
 'name': name,  
 'surname': surname,  
 'telephone': telephone,  
 'mail': mail,  
 'documents': decoded\_documents  
 }}  
 )  
  
 return redirect('/user/' + user\_id)  
  
  
############################## <!----! Договора !----!>  
  
@app.route('/contract/<contract\_id>')  
def contract(contract\_id):  
 contract = collection\_contracts.find\_one({'\_id': ObjectId(contract\_id)})  
  
 if contract:  
 return render\_template('contract.html', contract=contract)  
 else:  
 return 'Договор не найден'  
  
  
@app.route('/add\_contract', methods=['POST'])  
def add\_contract():  
 data = request.form # Получение данных из POST-запроса  
  
 contractNumber = data.get('contractNumber')  
 date = data.get('date')  
 amount = data.get('amount')  
 customer = data.get('customer')  
 workType = data.get('workType')  
 workTypeName = data.get('workTypeName')  
  
 # Создание объекта сотрудника  
 contract = {  
 "contractNumber": contractNumber,  
 "date": date,  
 "amount": amount,  
 "customer": customer,  
 "workType": workType,  
 "workTypeName": workTypeName  
 }  
  
 collection\_contracts.insert\_one(contract).inserted\_id  
 contracts = collection\_contracts.find()  
  
 return render\_template('contracts.html', contracts=contracts)  
  
  
@app.route('/contract/<contract\_id>/remove', methods=['POST'])  
def remove\_contract(contract\_id):  
 # Преобразуем строковый user\_id в ObjectId  
 contract\_id = ObjectId(contract\_id)  
 # Удаляем пользователя  
 collection\_contracts.delete\_one({'\_id': contract\_id})  
  
 contracts = collection\_contracts.find()  
 return render\_template('contracts.html', contracts=contracts)  
  
  
@app.route('/contract/<contract\_id>/edit', methods=['POST'])  
def edit\_contract\_save(contract\_id):  
  
 contractNumber = request.form['contractNumber']  
 date = request.form['date']  
 amount = request.form['amount']  
 customer = request.form['customer']  
 workType = request.form['workType']  
 workTypeName = request.form['workTypeName']  
  
  
 # Обновляем пользователя  
 collection\_contracts.update\_one(  
 {'\_id': ObjectId(contract\_id)},  
 {'$set': {  
 'contractNumber': contractNumber,  
 'date': date,  
 'amount': amount,  
 'customer': customer,  
 'workType': workType,  
 'workTypeName': workTypeName  
 }}  
 )  
  
 return redirect('/contract/' + contract\_id)  
  
  
############################## <!----! Партнёры !----!>  
  
@app.route('/partner/<partner\_id>')  
def partner(partner\_id):  
 partner = collection\_partners.find\_one({'\_id': ObjectId(partner\_id)})  
  
 if partner:  
 return render\_template('partner.html', partner=partner)  
 else:  
 return 'Договор не найден'  
  
  
@app.route('/add\_partner', methods=['POST'])  
def add\_partner():  
 data = request.form # Получение данных из POST-запроса  
  
 contractNumber = data.get('contractNumber')  
 date = data.get('date')  
 amount = data.get('amount')  
 customer = data.get('customer')  
 workType = data.get('workType')  
 workTypeName = data.get('workTypeName')  
  
 # Создание объекта сотрудника  
 partner = {  
 "contractNumber": contractNumber,  
 "date": date,  
 "amount": amount,  
 "customer": customer,  
 "workType": workType,  
 "workTypeName": workTypeName  
 }  
  
 collection\_partners.insert\_one(partner).inserted\_id  
 partners = collection\_partners.find()  
  
 return render\_template('partners.html', partners=partners)  
  
  
@app.route('/partner/<partner\_id>/remove', methods=['POST'])  
def remove\_partner(partner\_id):  
 # Преобразуем строковый user\_id в ObjectId  
 partner\_id = ObjectId(partner\_id)  
 # Удаляем пользователя  
 collection\_contracts.delete\_one({'\_id': partner\_id})  
  
 partners = collection\_partners.find()  
 return render\_template('partners.html', partners=partners)  
  
  
@app.route('/partner/<partner\_id>/edit', methods=['POST'])  
def edit\_partner\_save(partner\_id):  
  
 contractNumber = request.form['contractNumber']  
 date = request.form['date']  
 amount = request.form['amount']  
 customer = request.form['customer']  
 workType = request.form['workType']  
 workTypeName = request.form['workTypeName']  
  
  
 # Обновляем пользователя  
 collection\_partners.update\_one(  
 {'\_id': ObjectId(partner\_id)},  
 {'$set': {  
 'contractNumber': contractNumber,  
 'date': date,  
 'amount': amount,  
 'customer': customer,  
 'workType': workType,  
 'workTypeName': workTypeName  
 }}  
 )  
  
 return redirect('/partner/' + partner\_id)

1. **Код Empolye\_funcuon.py**

import json  
from urllib import request  
  
  
def add\_employee(data):  
  
 # Извлечение данных о сотруднике  
 surname = data.get('surname')  
 name = data.get('name')  
 telephone = data.get('telephone')  
 mail = data.get('mail')  
 documents = []  
  
 employeer = {  
 "surname": surname,  
 "name": name,  
 "telephone": telephone,  
 "mail": mail,  
 "documents": documents  
 }  
  
 return employeer

**Работа базы данных**



**Сопутствующие модели**



**Рисунок 6- Use-case**

**Тестирование проекта**

Тестирование программного обеспечения — это исследование, проводимое с целью предоставления заинтересованным сторонам информации о качестве тестируемого программного продукта или услуги. Тестирование программного обеспечения также может обеспечить объективное, независимое представление о программном обеспечении, позволяющее бизнесу оценить и понять риски, связанные с внедрением программного обеспечения. Методы тестирования включают процесс выполнения программы или приложения с целью обнаружения сбоев и проверки пригодности программного продукта для использования.

Тестирование стабильности или надежности — один из видов нефункционального тестирования ПО, целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.

Перед тем как подвергать ПО экстремальным нагрузкам стоит провести проверку стабильности в предполагаемых условиях работы, то есть погрузить продукт в полную рабочую атмосферу. При тестировании, длительность его проведения не имеет первостепенного значения, основная задача — наблюдая за потреблением ресурсов, выявить утечки памяти и проследить чтобы скорость обработки данных и/или время отклика приложения в начале теста и с течением времени не уменьшалась. В противном случае вероятны сбои в работе продукта и перезагрузки системы.

Проверка эргономичности — исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования взаимодействия человек-компьютер в целом — формулируют универсальные принципы.

Под «чёрным ящиком» понимается объект исследования, внутреннее устройство которого неизвестно. Специальные знания кода приложения, внутренней структуры и знания программирования в целом не требуются. Тестировщик знает, что должно делать программное обеспечение, но не знает, как оно это делает. Например, тестировщик знает, что конкретный ввод возвращает определенный неизменный результат, но не знает, как программное обеспечение в первую очередь выдает результат.

Типичные методы разработки тестов черного ящика включают:

* Анализ граничных значений;
* Угадывание ошибок;
* Тестирование перехода в состояние;
* Тестирование вариантов использования;
* Анализ предметной области;
* Тестирование синтаксиса.

Лучшим методом тестирования, будет юзабилити-тестирование, так как мобильно приложение в первую очередь должно быть удобным и интуитивно понятным для пользования в полной мере.

При проведении тестирования было выявлено, что все вкладки, представленные на сайте, работают без ошибочно. Особых неисправностей и не стабильной работы не выявлено. Все фотографии и описание находятся на одном уровне.

**Рекомендации к дальнейшей разработке**

1. *Реализовать поиск по секциям*
2. *Добавить разделение на роли*
3. *Реализовать авторизацию*
4. *Разработать работу с торгами компании ООО «ИДИС ИБ»*
5. *Реализовать функционал администратора по добавлению новых пользователей и принятия изменение данных.*
6. *Реализовать отправку заявки на изменение.*
7. *Добавить расширенную валидацию.*