Пояснительная записка по проекту “Rest-api приложение на Flask”.

Автор проекта:

Идея проекта: создать веб-сервис для работ по всему миру, на котором пользователь мог предложить вакансию или связаться с представителем вакансии и договориться о найме на работу

Требования к идее: сервис должен позволять пользователю регистрироваться, входить в свой аккаунт, изменять данные аккаунта, добавлять работу

Ссылка на приложение в облаке: <https://yl-flask-alice.herokuapp.com/>

Папки и файлы проекта:

* alembic – папка управления миграций
* data – в ней находится файл создания движка sqlaclhemy и файл предоставляющий модели для добавления в БД при создании
* db – папка с БД
* models – папка с моделями пользователей, работ, сообщений и категорий работ
* rest-api – папка с ресурсами и файлом парсеров аргументов
* static: img – папка с аватарками пользователей. Загрузка с помощью сохранение с помощью file.save(path);

css – файлы со стилями css

* templates – папка с шаблонами. Главный шаблон – main\_template.html, остальные его дополняют
* alembic.ini – файл конфигурации alembic
* forms\_for\_page.py – файл с классами форм для шаблонов
* main\_app.py – файл с приложением и обработчиками страниц
* requirements.txt – файл с нужными библиотеками
* Rest-api приложение на Flask.pptx – презентация проекта
* tests.py – файл с тестирующими функциями

Модели:

* users
  + id – id пользователе
  + surname – фамилия пользователя
  + name – имя пользователя
  + age – возраст пользователя(число)
  + speciality – специализация
  + about – пользователь о себе
  + email – почта(уникальна)
  + hashed\_password - захешированный пароль(добавляется с помощью метода set\_password)
  + avatar\_url – url к аватарке пользователя
  + created\_date - время создания аккаунта
  + chats – чаты. Чаты разделяются “;”, а id получателя и id сообщения “, ”(“2, 1; 4, 6”)
  + job – список связанных Jobs элементов
  + message – список связанных Message элементов

функция set\_password – добавляет захешированный пароль к пользователю

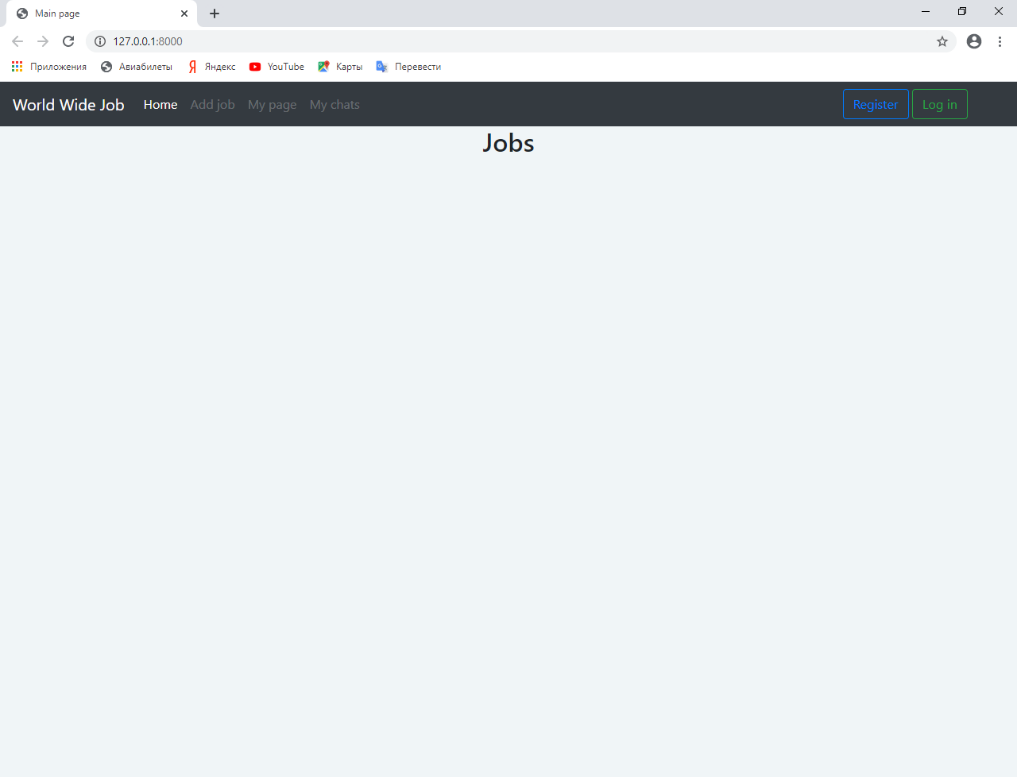
функция check\_password – возвращает сравнение переданного пароля, с паролем пользователя

* jobs
  + id – id работы
  + author – id автора(Внешний ключ)
  + header – заголовок
  + requirements – требования
  + description – описание
  + creation\_date – время создания работы
  + user – список User обьектов
  + categories– список категорий работы
* messages
  + id – id работы
  + sender – id отправителя(Внешний ключ)
  + receiver – id получателя
  + text – текст сообщения
  + date – время обновления сообщения
  + user – список User объектов
* categories
  + id – id категории
  + name – наименование категории

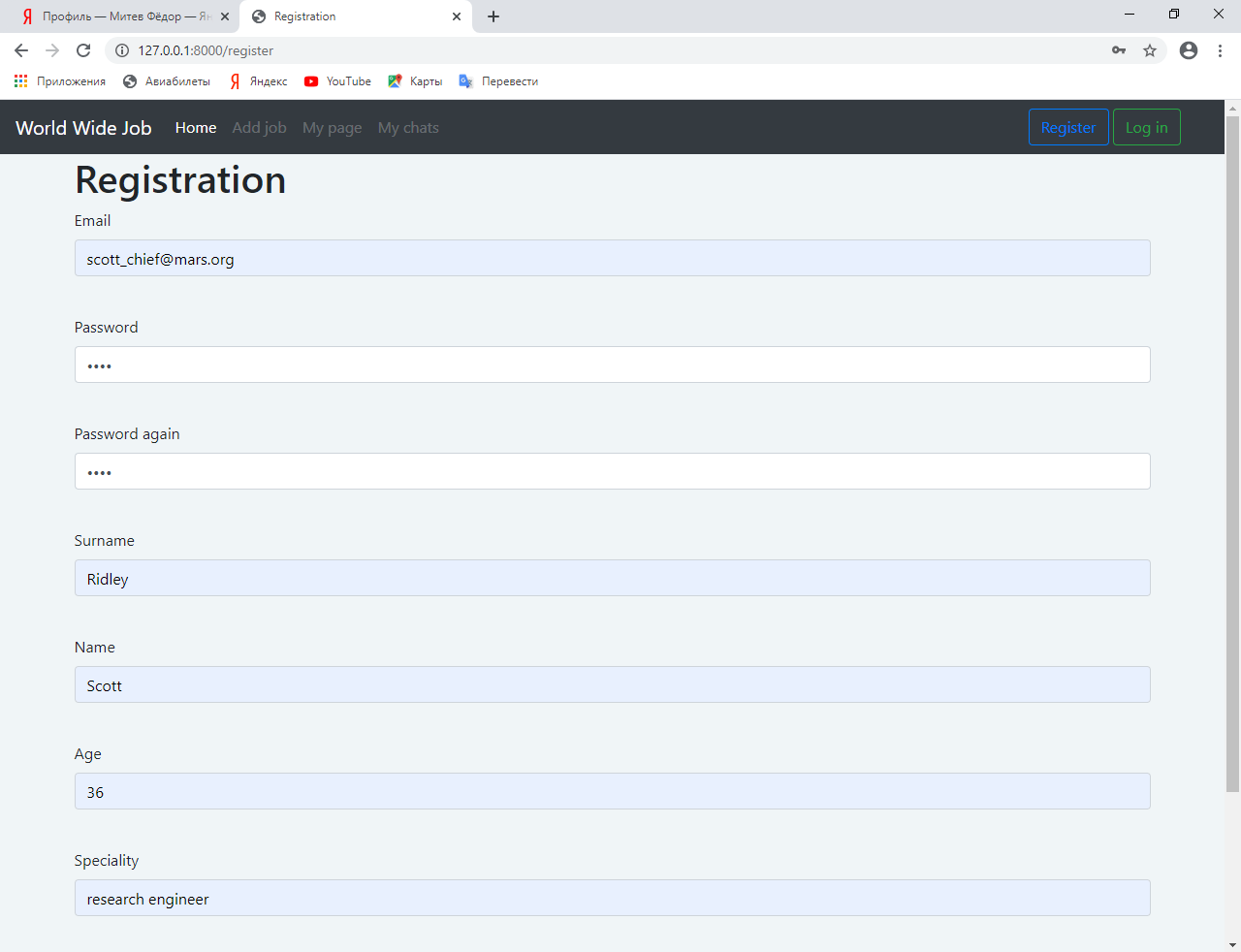
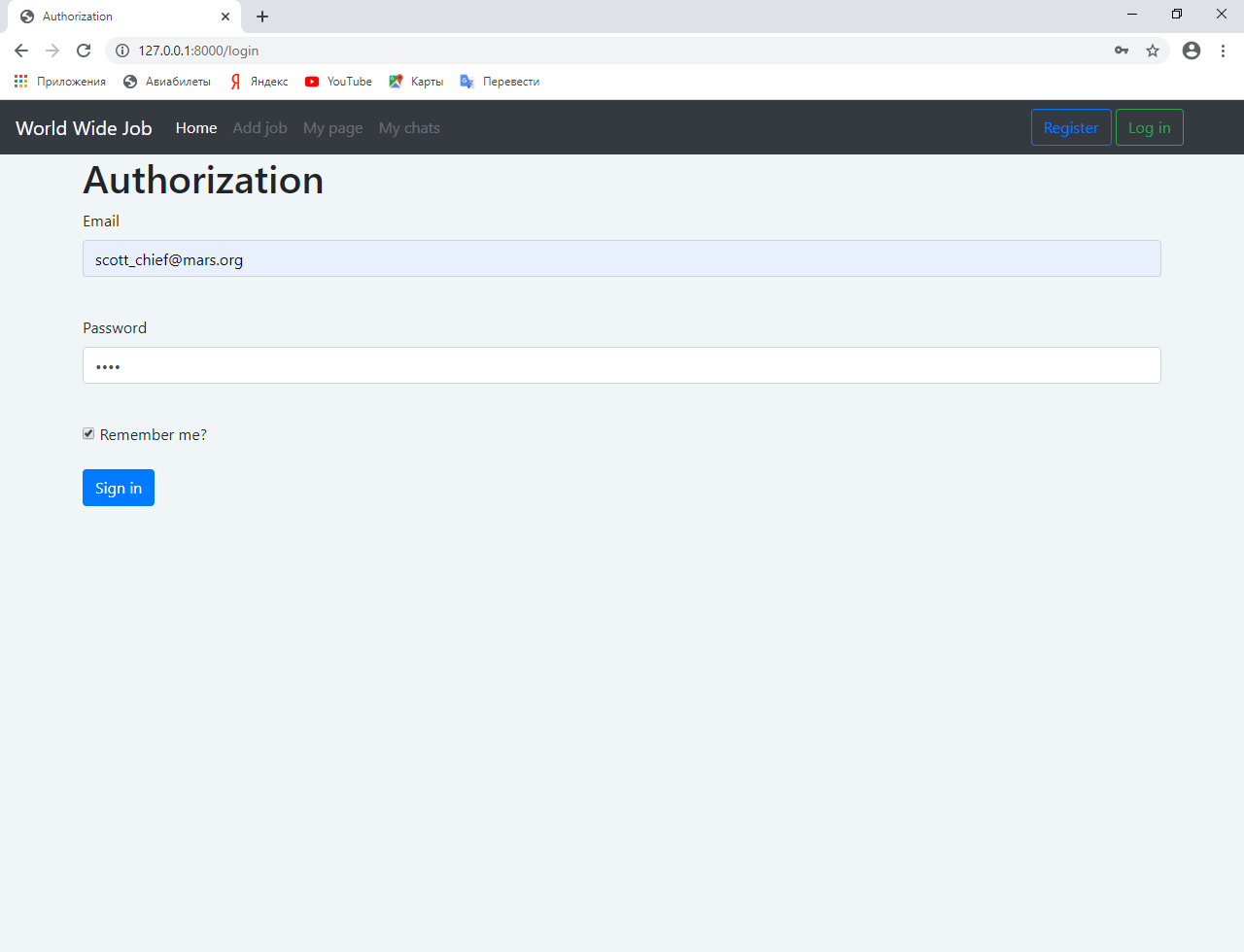
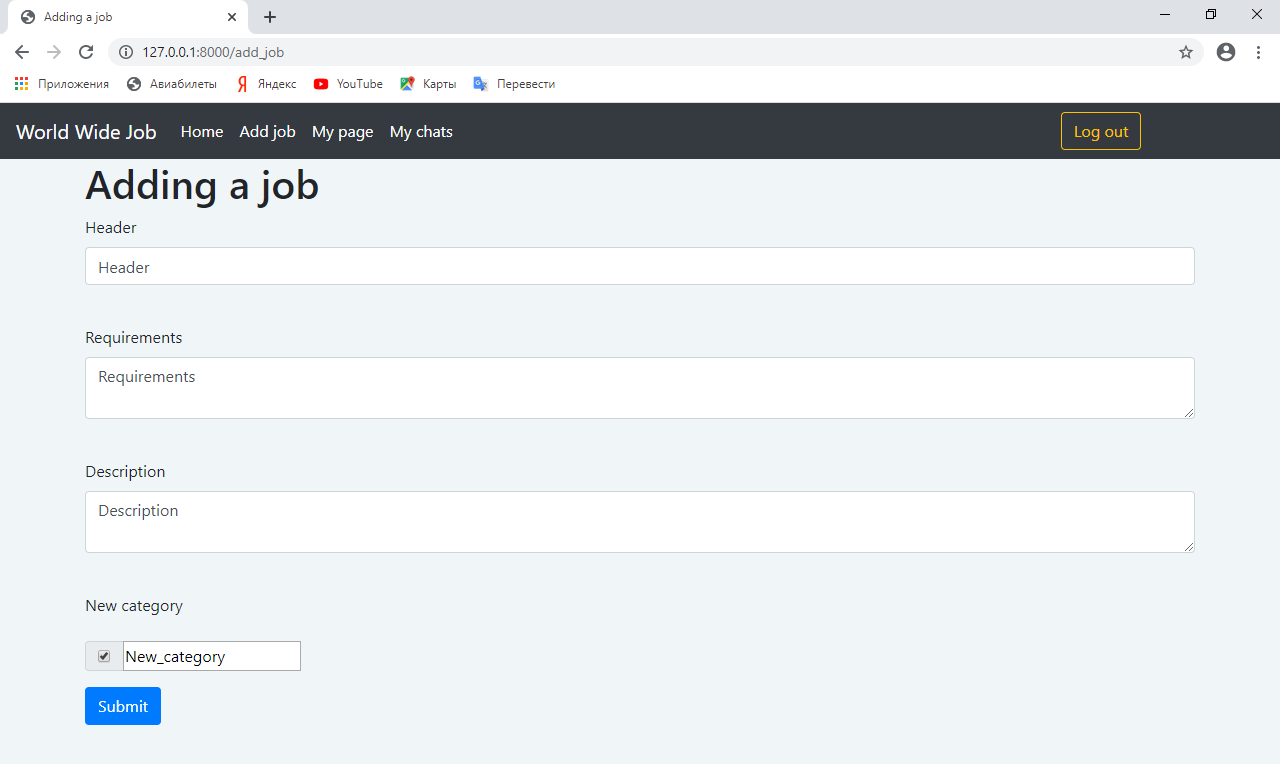
создаётся промежуточная таблица association:

* jobs – id работы(Внешний ключ)
* category - id категории(Внешний ключ)

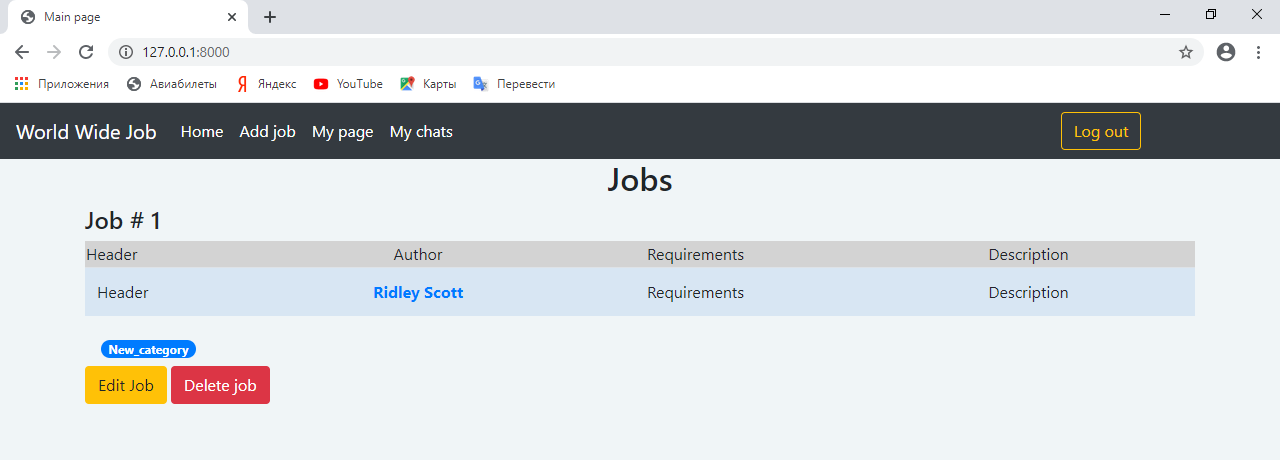
Как происходит взаимодействие с пользователем:

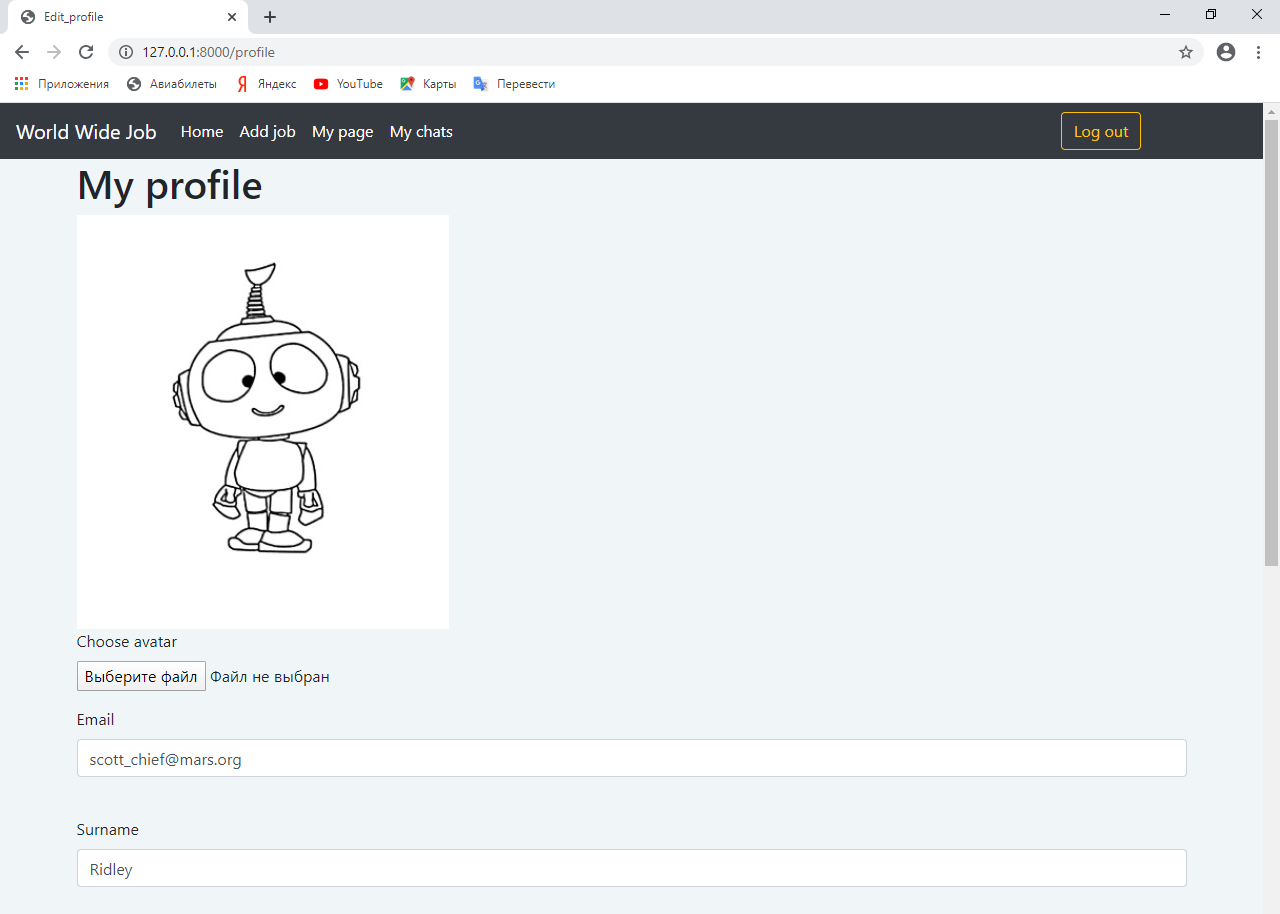
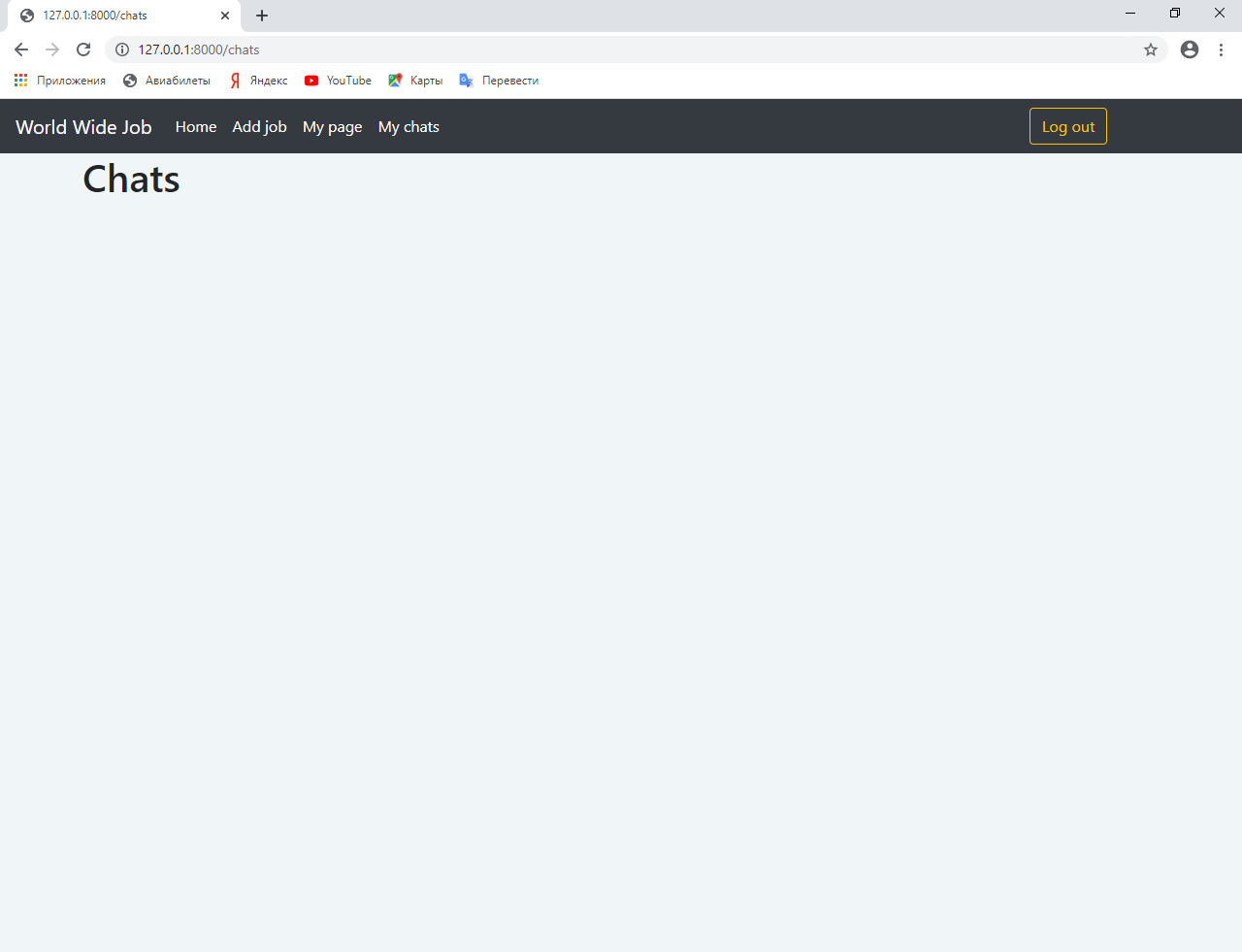
* При переходе по ссылке приложения пользователь попадает на главную страницу main\_page(“/”) – на ней отображаются работы. Не авторизованным пользователям доступны только функции просмотра работы и профилей пользователей.

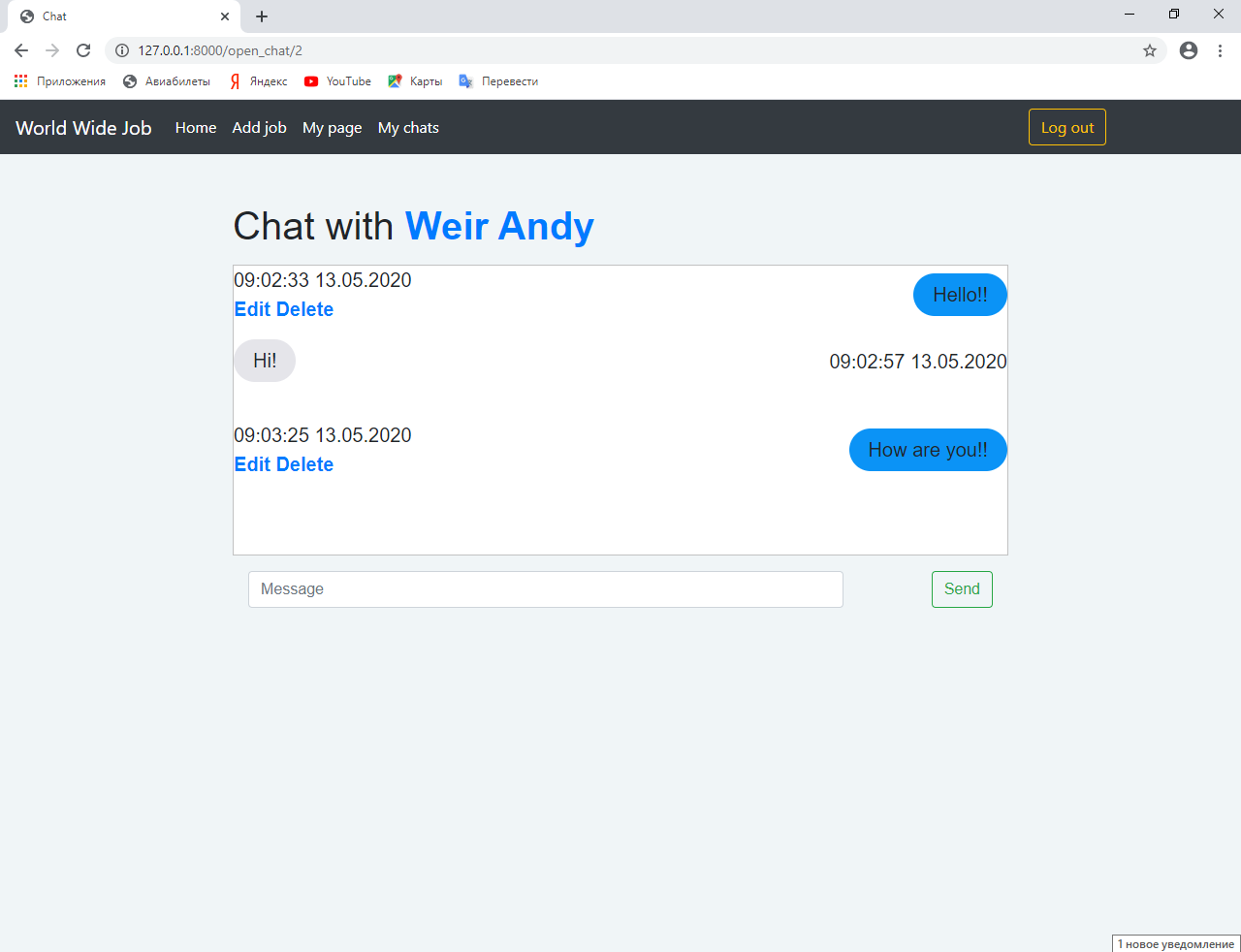
Верхняя часть страницы – это шаблон main\_template.html который будут дополнять другие шаблоны. Здесь можно увидеть кнопки для перехода на страницу регистрации или авторизации, а также кнопка Home для возвращения на главную страницу

* регистрация происходит в функции register(“/register”), которая берёт заполненную пользователем форму и отправляет её данные для регистрации пользователя в POST запрос. Форма также проверяет уникальность введенной почты.
* Дальше происходит авторизация пользователя в функции login(“/login”) и сверяются данные формы: почта и пароль должны соответствовать одному пользователю
* При успешной авторизации пользователь входит в систему и она видит его как действительного пользователя – за это отвечает функция load\_user. Также меняется шапка страницы: вместо регистрации и авторизации появляется кнопка выхода и становятся доступны функции добавления работ, переписки и профиль. За выход из системы отвечает функция logout(“/logout”)
* Чтобы опубликовать свою вакансию, то надо чтобы пользователь обязательно заполнил поля заголовка работы(“header”) и требований(“requirements”). Обработка это страницы происходит в функции add\_job(“add\_job”). Также можно указать категорию работы. Работу можно изменять и удалять.

После публикации работы она появляется в списки работ на главной странице



* Изменение профиля происходит в функции edit\_profile(“/profile”). Вы можете изменять любые данные вашей страницы, даже автарку. Для подтверждения любых изменений обязателен ввод старого пароля.
* Чаты можно увидеть при нажатии на кнопку My chats. За эту страницу отвечает функция chats (“/chats ”). При нажатии на имя собеседника можно будет перейти в окно чата(или из его профиля для первого контакта)
* За отображение чатов отвечает функция messaging, которая берёт переписку из БД и передаёт её шаблону , а также отправляет сообщения.



* Чаты можно изменять(функция edit\_message (“/edit\_message/<int:message\_id> ”)) и чаты можно удалять(функция delete\_message (“/delete\_message/<int:message\_id> ”))

Rest-api обработчики(в конце каждой get, post, put, delete функции - session.commit())

файл users\_resources.py

* UsersResource:
  + get
    - проверка наличия пользователя
    - получение пользователя по id
    - возвращение атрибутов пользователя
  + put
    - парсинг аргументов
    - проверка наличия пользователя
    - получение пользователя по id
    - если пользователь меняет почту 🡪 проверка на уникальность почты
    - изменение атрибутов пользователя
    - если есть фото: изменяем avatar\_url
    - если есть пароль: изменяем password
* UsersListResource:
  + get
    - возвращается список словарей атрибутов
  + post
    - парсинг аргументов
    - проверка на уникальность почты
    - создание класса пользователя User(avatar\_url при создании ссылается на аватарку анонима)

файл jobs\_resources.py

* JobsResource:
  + get
    - проверка наличия работы
    - получение работы по id
    - возвращение атрибутов работы(атрибут categories в удобной форме для form.select.value: [(value, label), (value1, label1) …])
  + delete
    - получение работы по id
    - удаление работы
  + put
    - парсинг аргументов
    - проверка наличия работы
    - получение работы по id
    - изменение атрибутов работы
    - 2 списка новых и старых категорий
    - удаляем категории которые пропали
    - добавляем категории которые появились
    - если есть чекбокс новой категории, то создаём категорию, присваиваем ей имя и добавляем в список категорий
* JobsListResource:
  + get
    - возвращение списка словарей атрибутов работы(атрибут categories в удобной форме для form.select.value: [(value, label), (value1, label1) …])
  + post
    - парсинг аргументов
    - создание класса работы Jobs
    - из списка категорий добавляются выбранные
    - если есть чекбокс новой категории, то создаём категорию, присваиваем ей имя и добавляем в список категорий

файл messages\_resources.py

* MessagesResource:
  + get
    - проверка наличия сообщения
    - получение сообщения по id
    - возвращение атрибутов сообщения(получателя и текста сообщения)
  + delete
    - получение сообщения по id
    - получение класса отправителя и класса получателя

Здесь будет также изменяться атрибут User.chats для каждого из собеседников – поэтому берём два класса пользователей

* + - представление чатов в более удобном формате(список кортежей: получатель, сообщение)
    - получение списка сообщений
    - итерация по списку сообщений:
      * если сообщение с соответствующим id и оно последнее, то если это единственное сообщение – удаляем эти чаты у обоих пользователей, иначе изменяем значение id последнего сообщения на id предшествующего сообщения у обоих пользователей
      * перевожу списки чатов обратно в строковые форматы для пользователей
    - удаление сообщения
  + put
    - парсинг аргументов
    - если сообщение не пустое
      * получение сообщения по id
      * изменение атрибутов сообщения
* MessagesListResource:
  + get
    - парсинг аргументов
    - получение сообщений где (отправитель = мой id и получатель = id собеседника) или (отправитель = id собеседника и получатель = мой id)
    - переводив в список кортежей атрибутов(id, id отправителя, id получателя, текст, время в строковом формате)
  + post
    - парсинг аргументов
    - если сообщение не пустое
      * создаём класс сообщений Message
      * заполняем атрибуты
      * добавляю класс в сессию чтобы пользоваться атрибутом id
      * получаю отправителя и получателя
      * изменяю их чаты в список кортежей
      * если уже есть переписка с собеседником, то изменяю id последнего сообщения на id нового сообщения и добавляю чат в начало списка и делаю то же самое для получателя.
      * А если нет переписки, то создаю её в начале чатов пользователей
      * перевожу списки чатов обратно в строковые форматы для пользователей

Особенности приложения:

1. Категории можно изменять и добавлять новые категории.
2. Во время изменения профиля, если пользователь добавляет новую аватарку, то путь файла обрабатывается функцией secure\_filename которая убирает лишний путь файла, тем самым делает безопасной нашу веб-систему
3. Список чатов сортируется по обновлённости. Делается это в POST и DELETE запросах к ресурсам сообщений и которые работают с классами получателя и отправителя. В POST запросе проверяется наличие чата с получателем и если его нет, то создаёт в начале списка, иначе обновляет id последнего сообщения и перемещает чат в начало списка чатов(как самый новый). В DELETE запросе если удаляемое сообщение не из конца чата, то оно просто удаляется, иначе рассматривается два случая:
   1. если это сообщение единственное в переписке – чаты для обоих пользователей удаляются
   2. иначе обновляется id последнего сообщения на id предпоследнего