Задание 5 Разработка ПО. Часть1

**1. Тестирование юзабилити:**

Ситуация №1:

**Категория пользователя:**преподаватель, впервые использующий систему

**Затраченное время на задачу:** 10 минут​

**Сложность достижения результата:**лёгкая​

**Задача:**зарегистрироваться и войти в чат-систему, после чего в раздел «Задачи» сформулировать и отправить задание, необходимое для выполнения студентом, будь то к примеру задание по формированию тура, в случае эксплуатации системы группой «Туризм» при интеграции с CRM-системой.

**Итог**: у студента в чат-разделе «Задачи» появится задание от преподавателя, необходимое для выполнения. При выполнении данной задачи у преподавателя может возникнуть ситуация №8, когда он предложит внести какие-то изменения в интерфейс или поток событий, чтобы улучшить его. Эта ситуация покажет, насколько просто или нет работать в системе преподавателю, одному из ключевых конечных пользователей.​

Ситуация №2:

**Категория пользователя:**студент, впервые использующий ситсему ​

**Затраченное время на задачу:** 25 минут​

**Сложность достижения результата:**средняя​

**Задача:**студенту необходимо осуществить полный цикл входа в систему, после чего в чат-разделе «Задачи» получить задание, сделать его и отчитаться о выполнении.

**Итог**: студентом будет выполнено задание преподавателя, которое может оценить преподаватель. При выполнении данной задачи у студента может возникнуть ситуация №1, когда студент понимает задачу, но не может выполнить её в срок, но маловероятно, так как система довольно интуитивна и запутаться в ней сложно, помимо этого в подобных ситуациях поможет чат-раздел «Обсуждение», в котором происходит обычное общение, в котором в том числе можно попросить о помощи или консультации. Эта ситуация покажет, насколько система понятна для студентов, впервые работающих в чат-системе.

Ситуация №3:

**Категория пользователя:**человек, который будет обслуживать систему после передачи продукта разработчиками.​

**Затраченное время на задачу:** 60 минут​

**Сложность достижения результата:**сложная​

**Задача:**так как чат-системы могут расширятся в функционале и возможностях, может возникнуть необходимость в создании дополнительного модуля/функции, либо же обновлении /совершенствовании уже существующего. Для этого необходимо будет произвести работу с кодом системы.​

**Итог**: человеком, обслуживающим и поддерживающим систему будет произведено обновление/добавление модуля/функции системы. В ходе выполнения данного задания могут возникнуть следующие ситуации: №1 - понимает задачу, но не может выполнить её в срок; №2 - понимает цель, но вынужден испробовать несколько разных подходов её достижения; №7 - утверждает, что что-то неправильно или непонятно. Данная ситуация необходима для понимания, насколько хорошо подготовлена система для передачи её в обслуживание другому человеку, удобочитаемость кода, его сбалансированность и эффективное использование, которое бы не запутало.​

Ситуация №4:

**Категория пользователя:**разработчик ​

**Затраченное время на задачу:** 30 минут​

**Сложность достижения результата:**средняя​

**Задача:**необходимо осуществить интеграцию чат-системы с каким-либо более крупным проектом.

**Итог**: разработчиком будет выполнена интеграция системы с более крупной системой в качестве модуля. В ходе выполнения может возникнуть ситуации: №1 - понимает задачу, но не может выполнить её в срок; №2 - понимает цель, но вынужден испробовать несколько разных подходов её достижения. Возникнуть они могут в случае, если интеграция трудновыполнима и возникают ошибки, которые в свою очередь могут разными способами решиться, но на это уйдёт больше времени. Ситуация необходима для того, чтобы понять, насколько качественна система для быстрой и успешной интеграции с различными крупными проектами.

Ситуация №5:

**Категория пользователей**: разработчик.

**Сложность**: сложная.

**Время**: 50 минут.

Задача: разработчику нужно создать скрипт-программу, которая будет выполнять выборку из базы данных, с которой, в дальнейшем будет производиться какая-то работа, на момент задания выборка должна поместиться либо в обычный .txt файл, либо в Excel-файл.

Результат: разработчик получает готовый файл с данными, которые можно использовать при работе по переносу или видоизменению основной платформы проекта, или же любой задачи, которая требует большей внимательности к сохранности данных пользователей.

**2. Разработка:**

В ходе начала создания чат-системы, мною были установлены необходимые инструментарии, в которые входит Python, SQLite, а также Flask.

Была создана папка, в которой будет производиться разработка проекта, в неё добавлен файл app.py, в котором содержится код Python проекта, папка templates и static. В файле app.py произведена начальная работа и протестирована работа библиотека Flask на отсутсвие ошибок и успешный запуск.

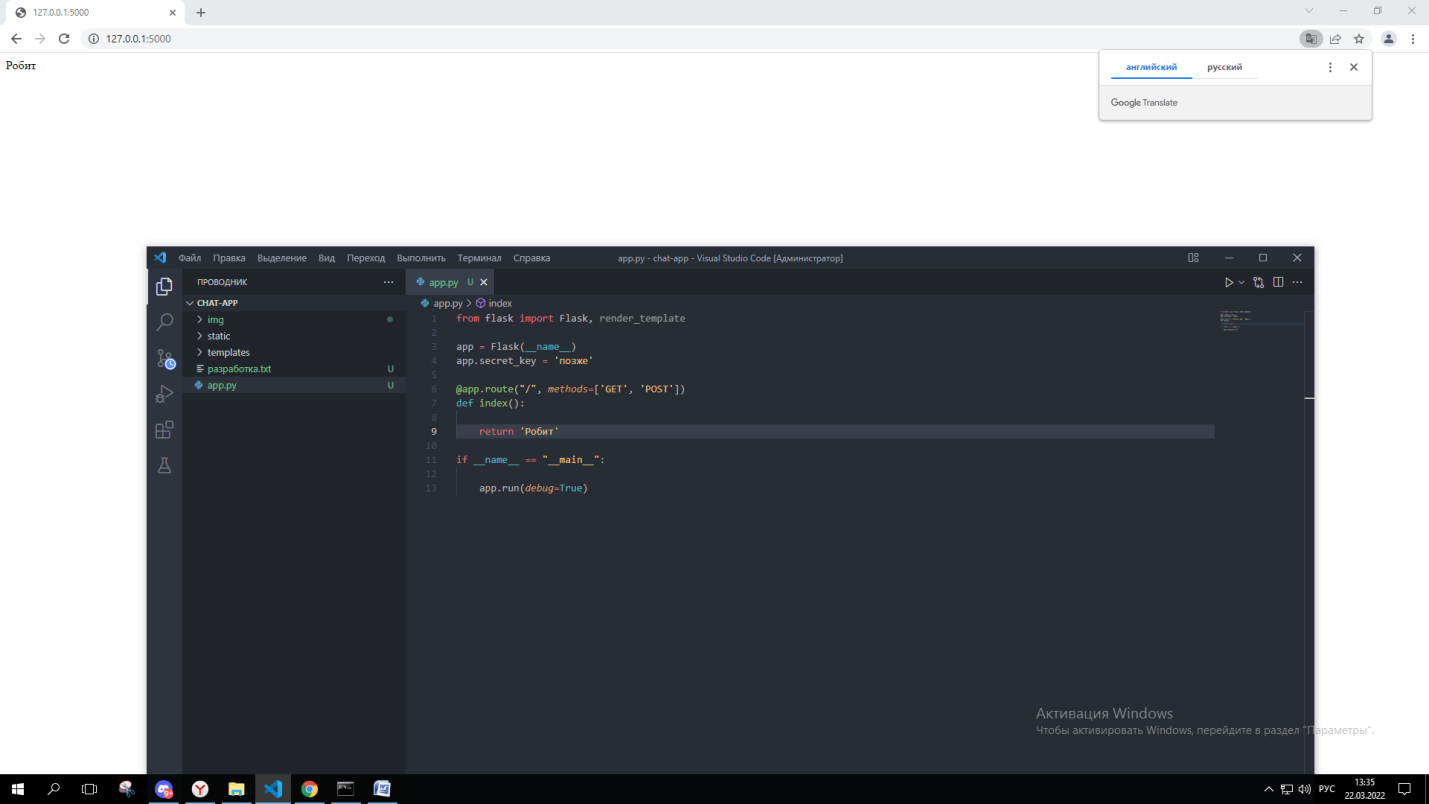


Рисунок 1 – начало проекта

После с помощью инструментария Flask, в который входит Jinja, был дан старт созданию шаблона html структуры для будущих страниц. Это позволяет использовать общую структуру сайта для всех страниц системы, меняя лишь контент, находящийся в блоках, написанных с помощью синтаксиса Jinja.

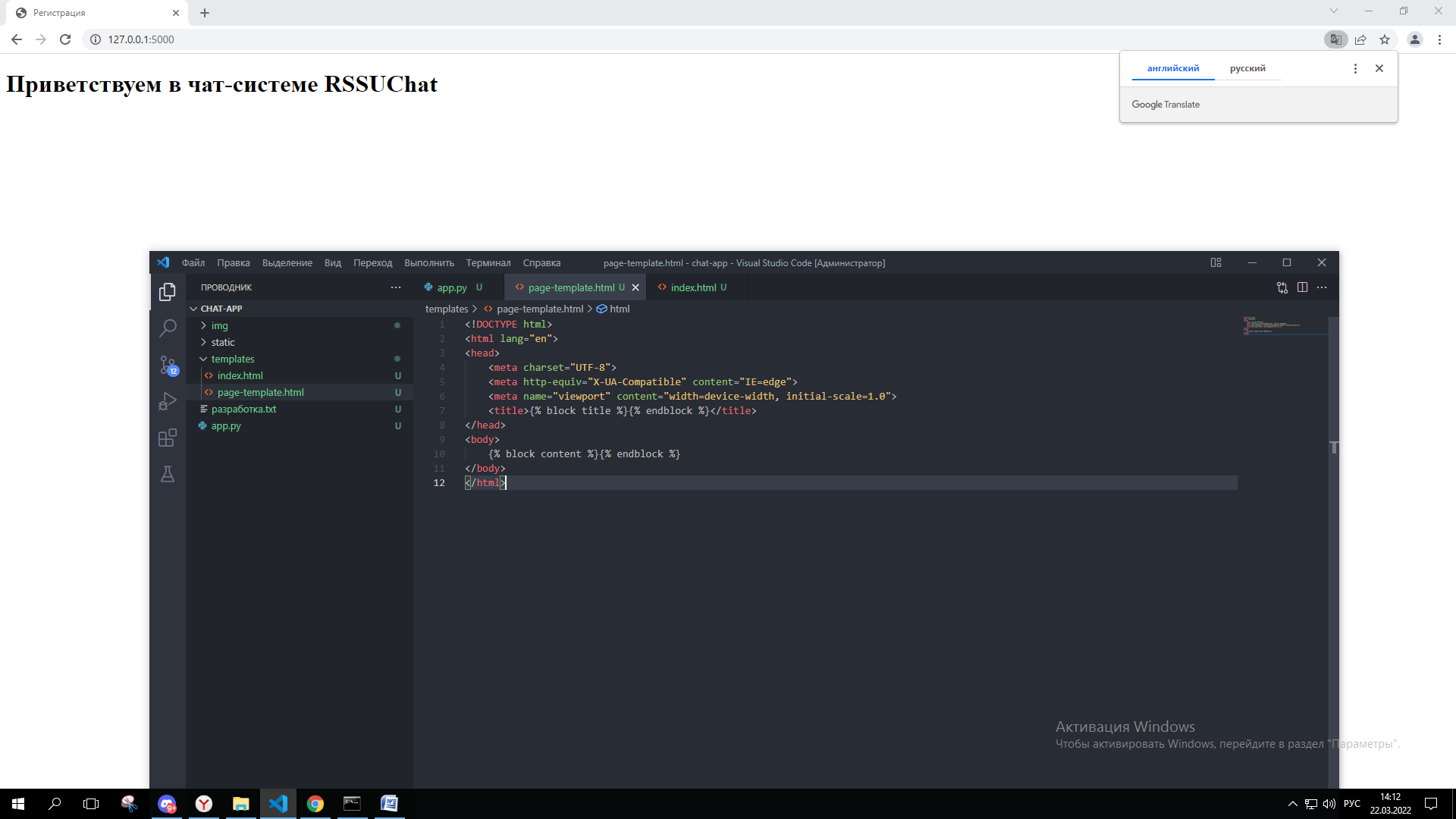


Рисунок 2 – шаблон страницы

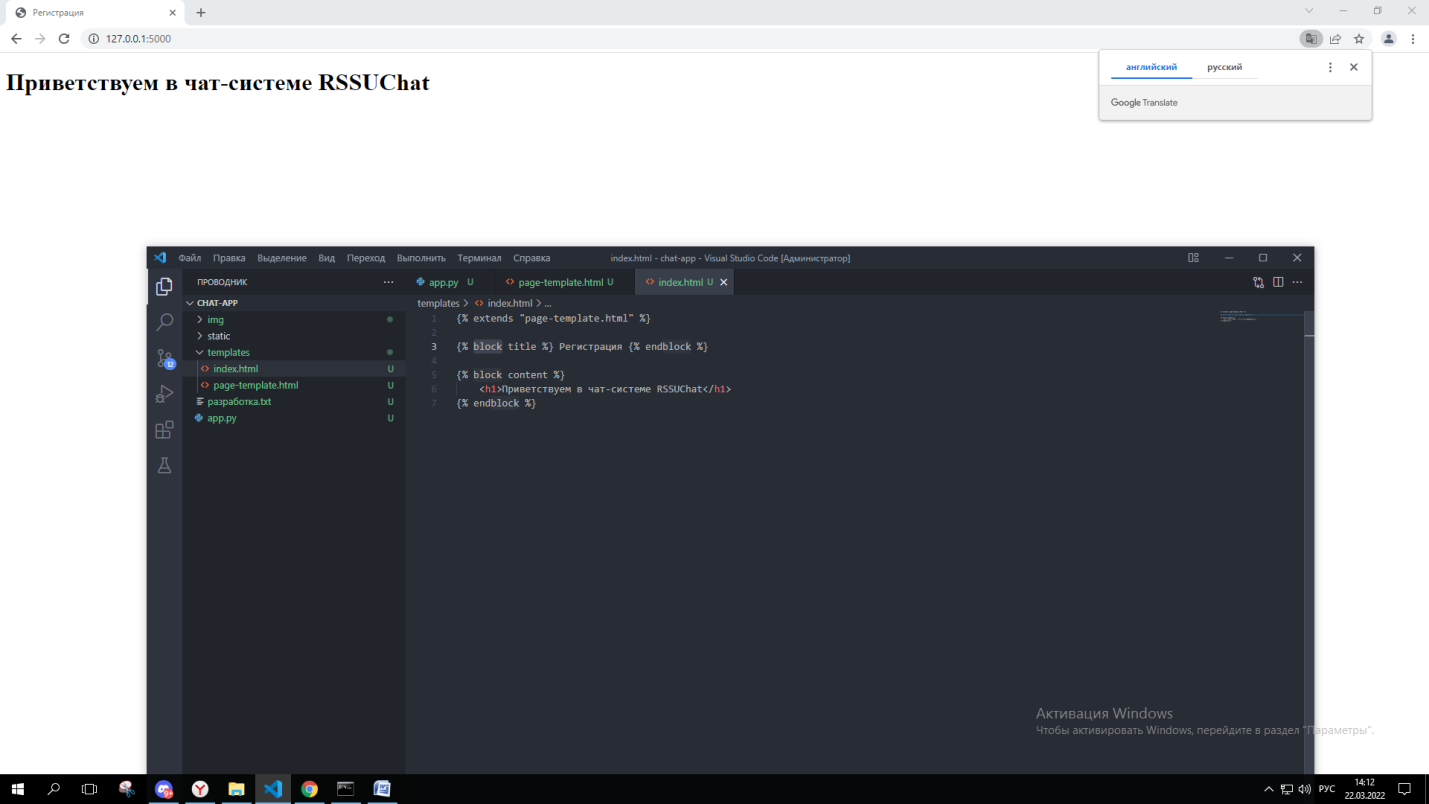


Рисунок 3 – страница index.html

После этого дан старт созданию страницы регистрации, для этого, после изучения материалов по Flask, было принято решение установить пару дополнений к нему, которые делают более удобной работу с формами. Был создан Питон файл, который будет содержать в себе обработчик формы и требования к ней, создано было с помощью специальных библиотек.

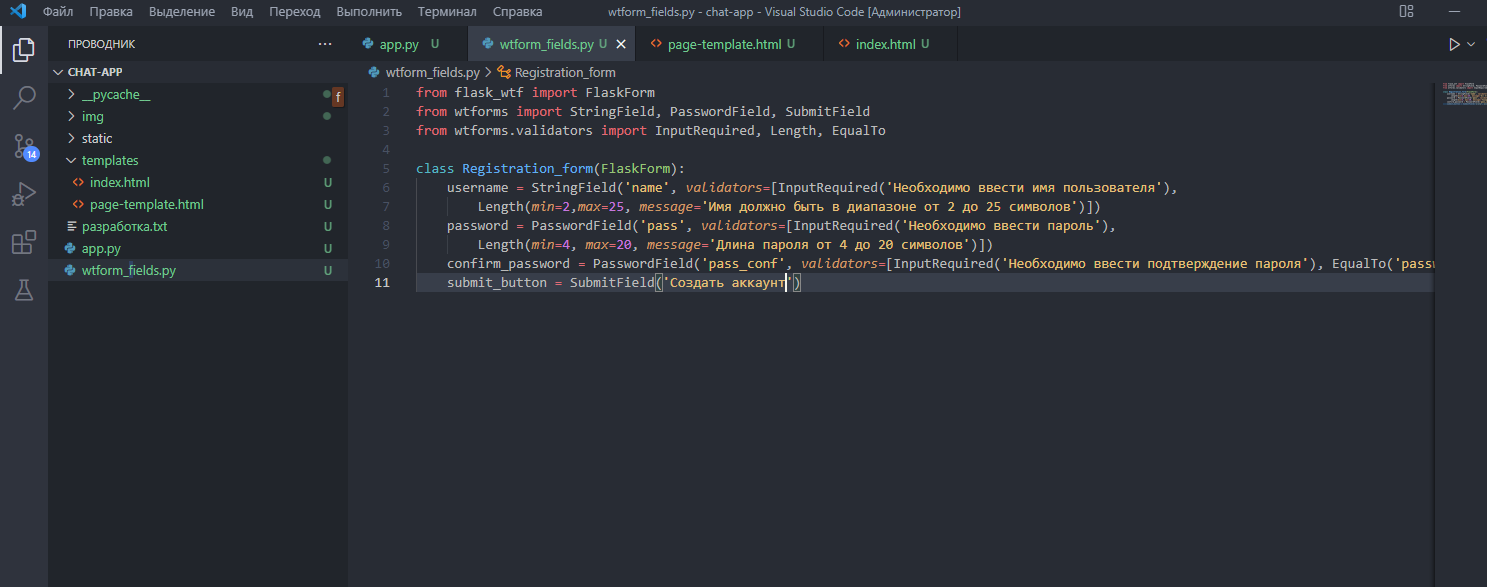


Рисунок 4 – создание форм с помощью WTForms

В файл index.html была добавлена форма и поля, необходимые для регистрации.

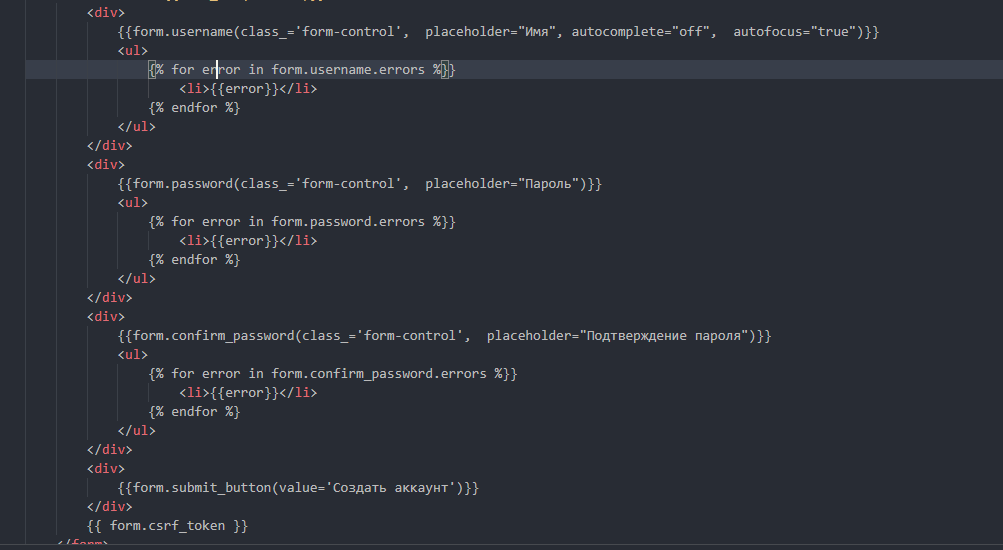


Рисунок 5 – вставка форм в HTML

Выглядит всё это на начальном этапе следующим образом:

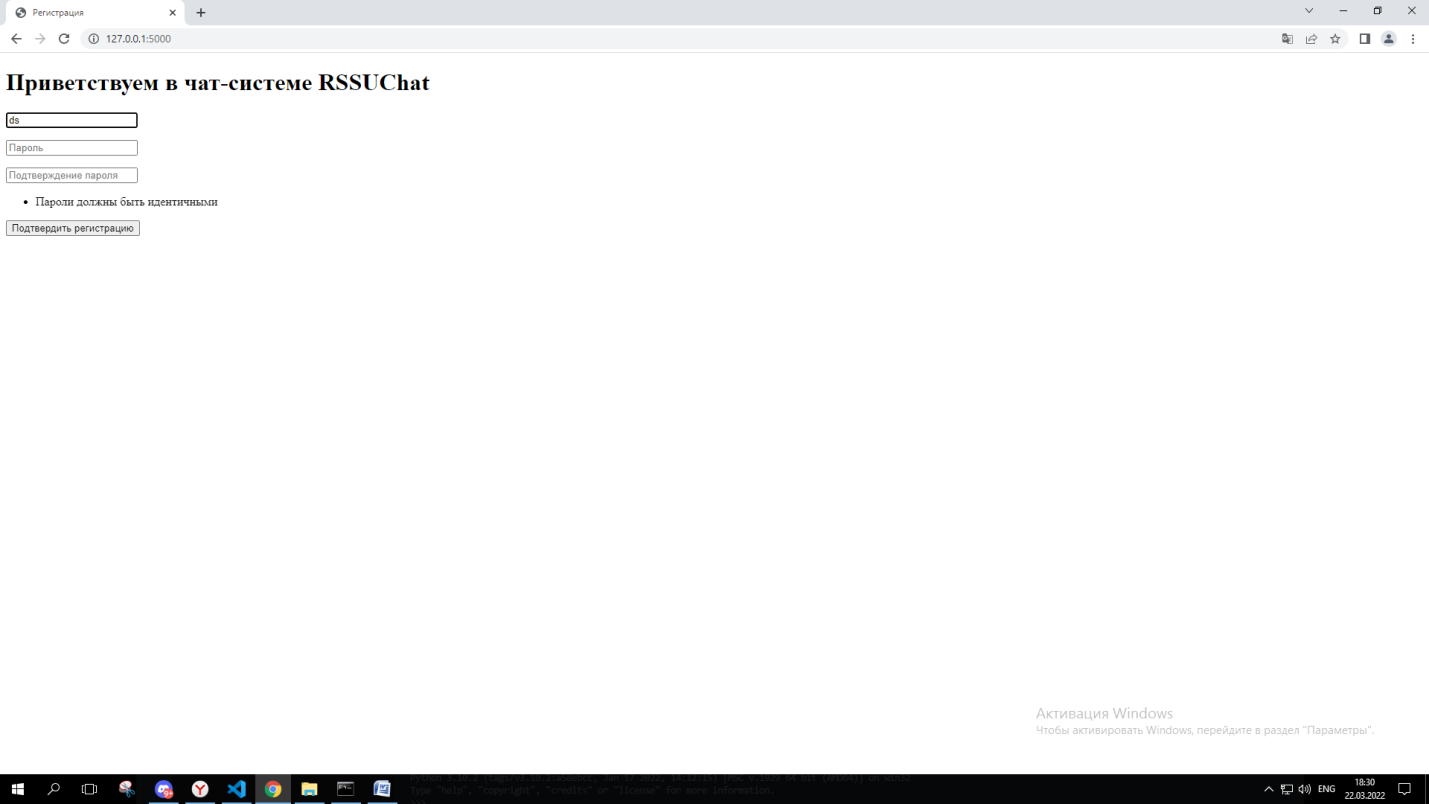


Рисунок 6 – страница index.html в начальном виде регистрации

Следующий этап-подключение БД к проекту, для этого был установлен SQLite и SQLAlchemy, для удобной работы с базами данным на Python.

Было создано два файла, первый – models.py, в котором содержится код создания таблицы и файл create.py, который осуществляет запуск кода создания. Создана таблица users в БД project.db, в которую будут помещаться аккаунты.

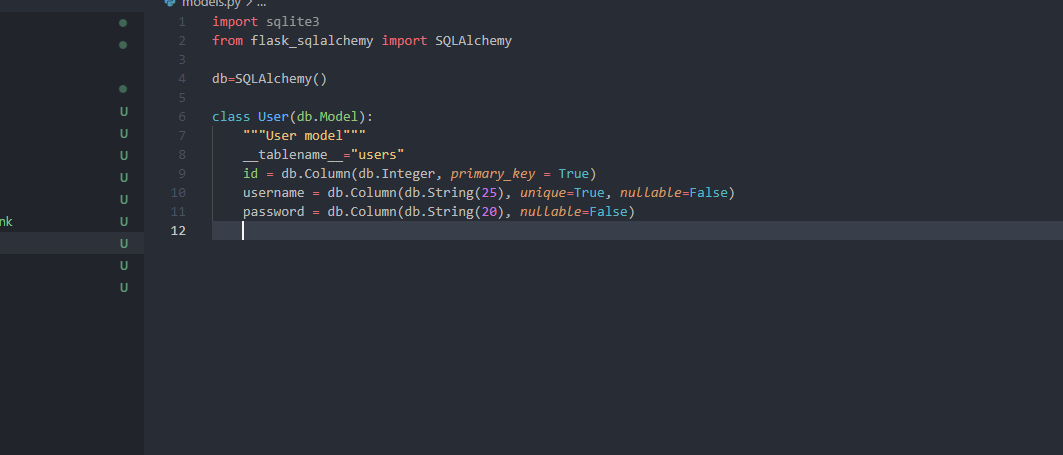


Рисунок 7 – работа БД SQLite с помощью SQLAlchemy

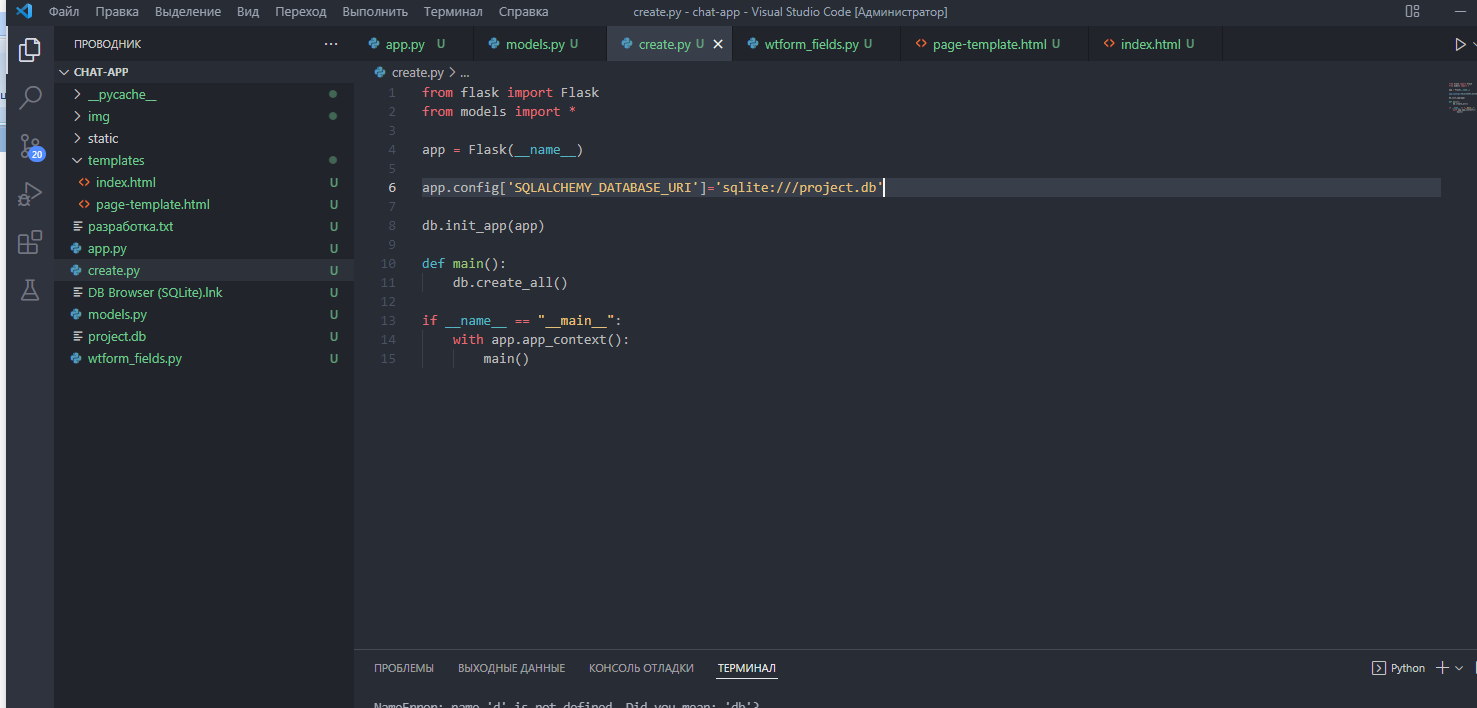


Рисунок 8 - работа БД SQLite с помощью SQLAlchemy

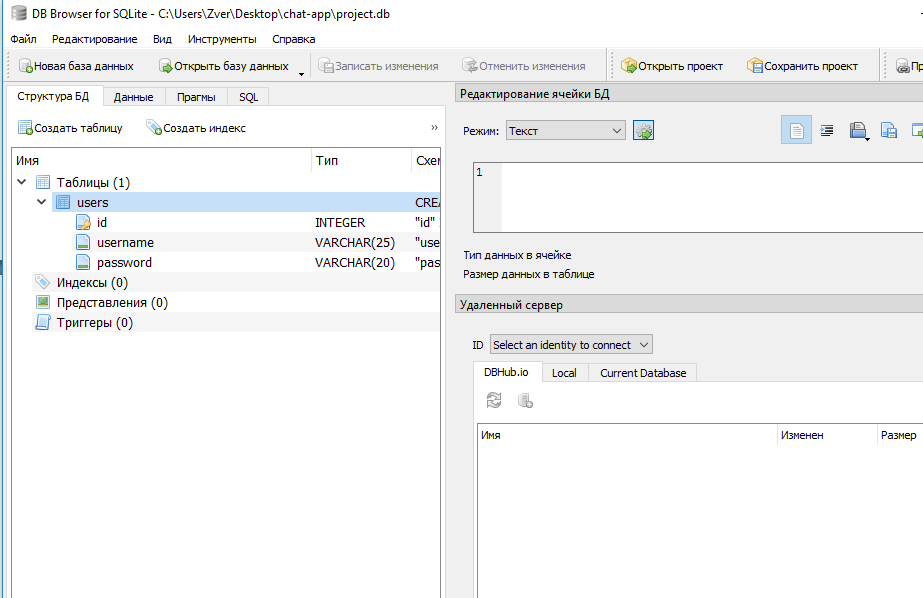


Рисунок 9 – созданная таблица users в SQLite

Добавлена обработка полей формы регистрации и занесение данных оттуда в БД, в случае, если имя пользователя уже имеется в БД, выдаётся ошибка.

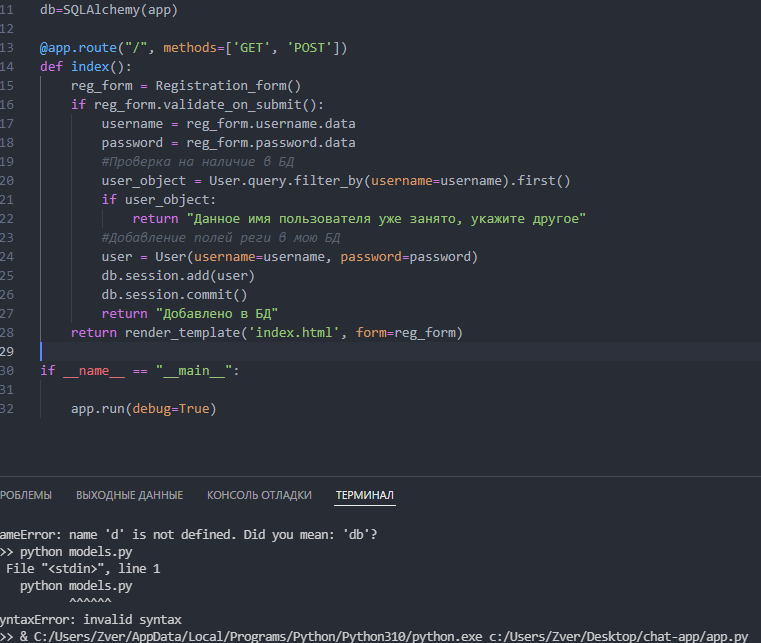


Рисунок 10 – обработка полей регистрации с проверкой на ошибки и занесением в БД

При проверке в рабочей состоянии всё завершилось успешно, запись была добавлена а при попытке использования того же самого никнейма выдавалась ошибка.

Следующим этапом, после изучения материала стала дорабатываться форма, целью было сделать вывод предупреждения, если имя пользователя уже занято, непосредственно на странице с формой. Сделано это было также с помощью библиотеки для Flask WTForms, которое позволяет создавать поля формы и добавлять к ним валидаторы (проверки на требования). В данном случае была добавлена функция, в которую добавляется кастомный валидатор, то есть который настраивается конкретно, в данном случае с помощью SQLAlchemy берётся запрос на соответствие данного с вводимого поля на данные из БД. Если введённое имя пользователя уже существует и переменная user\_object получает значение, то выполняется условие по выводу ошибки.

Перед продвижением к следующим этапам было принято решение стилистически оформить шаблон будущих страниц и саму страницу регистрации. Промежуточный результат является следующим:

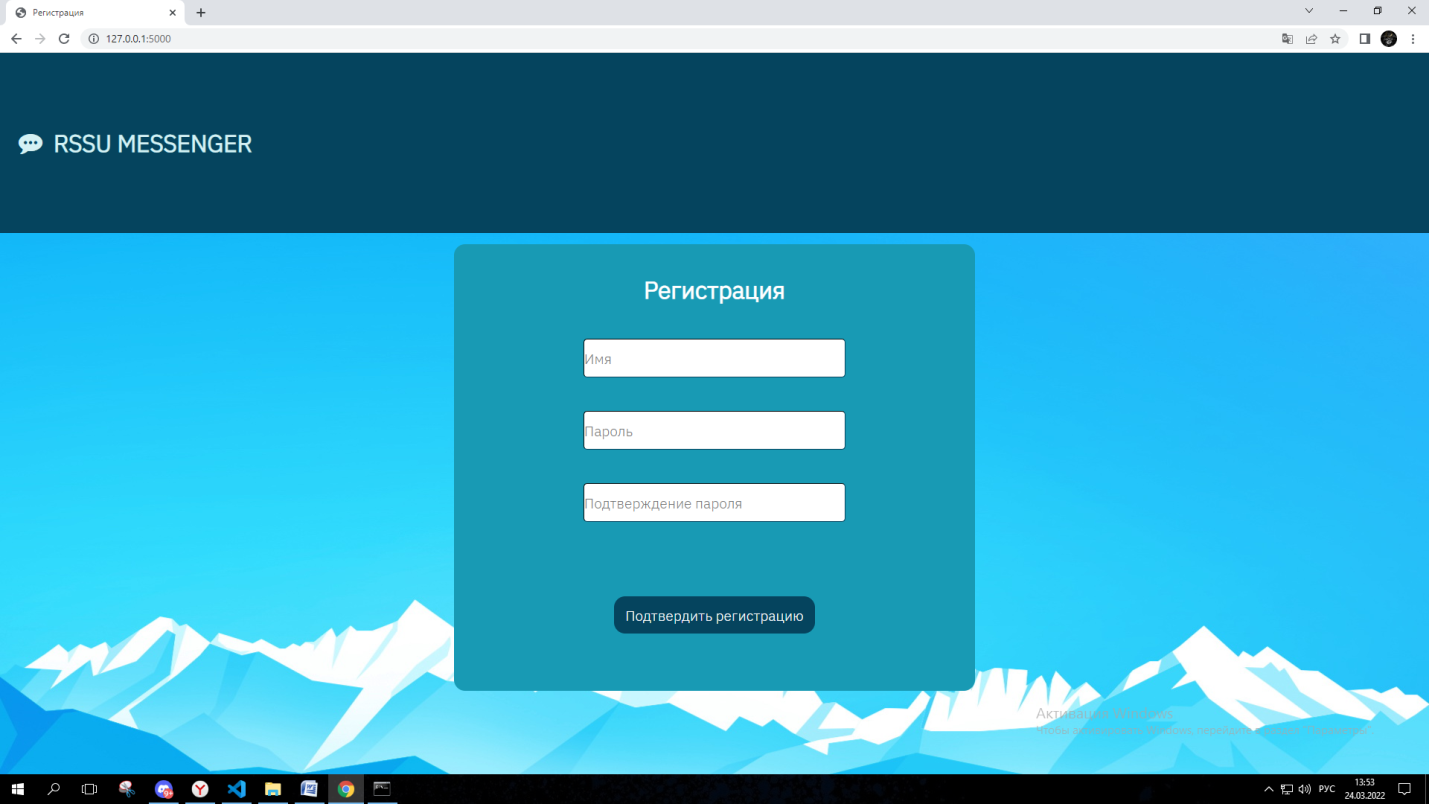


Рисунок 11 – приближенный к финалу внешний вид страницы регистрации

Следующий этап – создание страницы для непосредственного входа в систему(логинизация). Для этого схожим образом была создана страница login.html, в питон файле создана ещё одна форма с помощью WTForms, а также добавлена функция login, возвращающая страницу login.html. По начальному результату в файла wtforms создана функция, которая сверяет введённые данные в полях входа с данными пользователей в БД, в случае несоответствия выводит ошибку, а в случае успешного входа возвращает сообщение о удачном входе( в дальнейшем будет направлять на страницу чата). Изображения кода, а также внешнего вида страницы входа изображены дальше.



Рисунок 12 – файл wtform\_fields.py в который добавлен класс с формой для логинизации и функция для сверки данных полей с полями в БД



Рисунок 13 – файл app.py в который добавлена функция login, в которую входит класс Login\_form и возвращающая страницу login.html

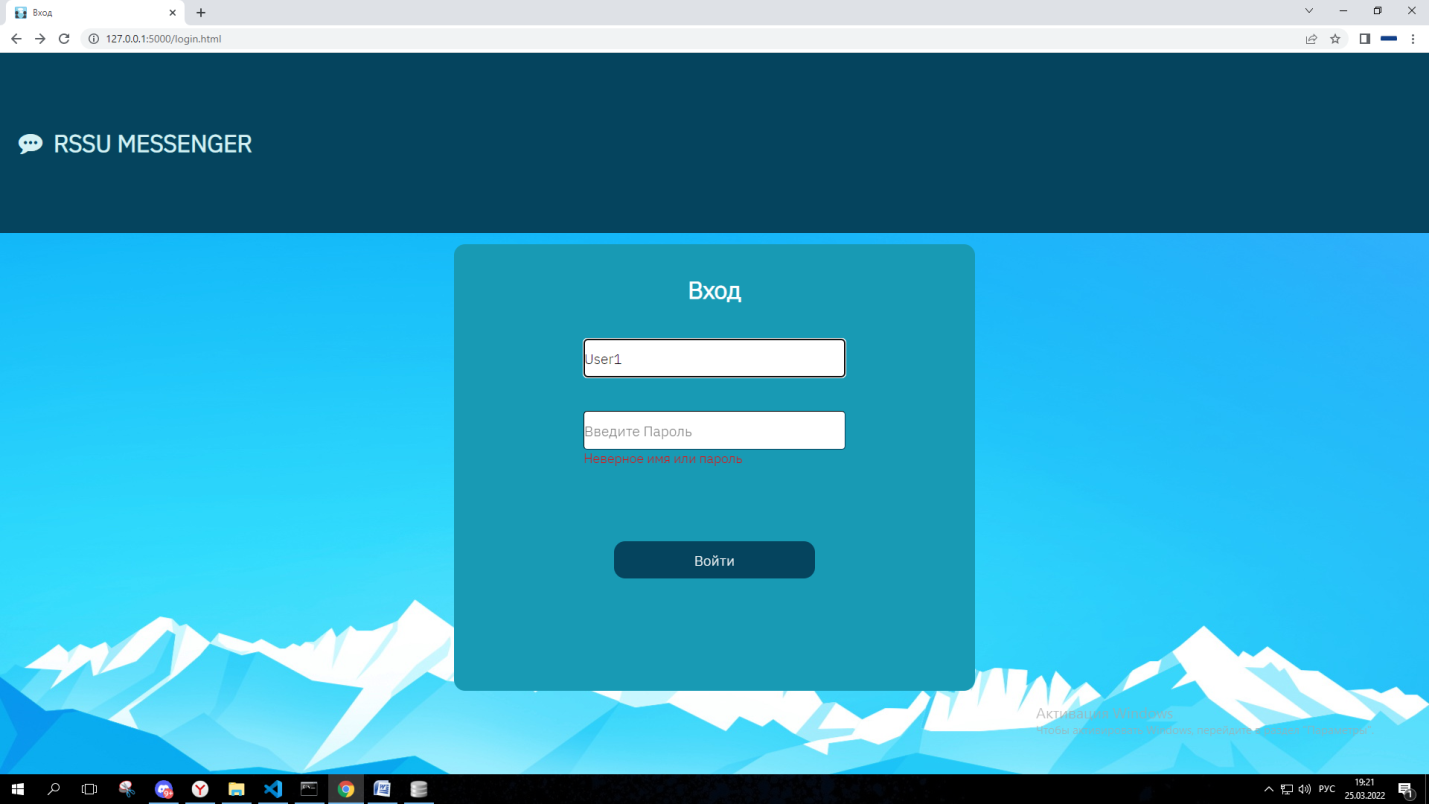


Рисунок 14 – страница входа

Следующим этапом было создание более безопасного способа работы с аккаунтами, которое заключается в хэширование паролей, чтобы не хранить их в БД в чистом виде и уменьшить риск хищения паролей пользователей. С этой целью была установлена библиотека passlib, которое позволяет производить хэширование. Из неё была импортирована кодировка pbkdf2\_sha512, которая применилась к записи пароля в БД.

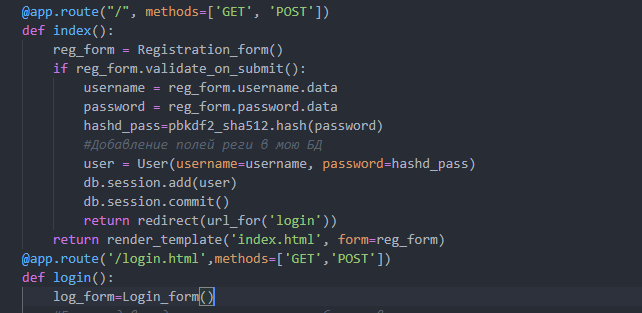


Рисунок 15 – добавление хэширования пароля в app.py

Итого запись пароля в базе данных выглядит следующим образом:

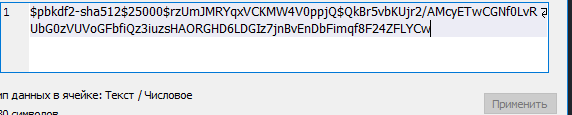


Рисунок 16 – пароль в БД в хэшированом виде

Для сверения введённого пароля в разделе «Вход» с тем, который был зарегистрирован в БД в хэшированом виде был написан следующий код:

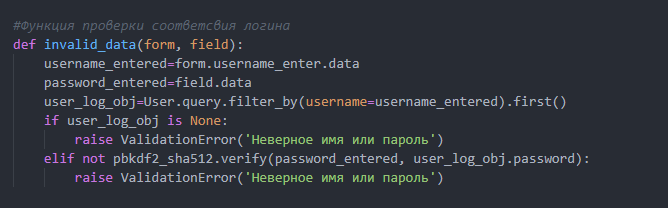


Рисунок 17 – сверка паролей