Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Казанский (Приволжский) федеральный университет»

*Институт вычислительной математики и информационных технологий*

**ОТЧЁТ**

**по производственной технологической (проектно-технологической) практике**

|  |  |
| --- | --- |
| Обучающийся Мишин С.С., 09-051  *(ФИО, группа)* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* |

Руководитель практики от КФУ

ст. преподаватель кафедры анализа данных

и технологий программирования Жажнева И.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(должность, ФИО) (подпись)*

Оценка за практику \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Дата сдачи отчета \_\_23.05.2024\_\_\_\_\_

Казань, 2024

Содержание

[АННОТАЦИЯ 3](#_Toc167642488)

[Введение 4](#_Toc167642489)

[1. Разработка системы контроля версий 5](#_Toc167642490)

[1.1. Реализация клиента VCS 5](#_Toc167642491)

[1.1.1. Хранение объектов файловой системы 5](#_Toc167642492)

[1.1.2. Структура хранения файлов в репозитории 6](#_Toc167642493)

[1.1.3. Сравнение файлов и создание патчей 7](#_Toc167642494)

[1.1.4. Сравнение деревьев и коммитов 8](#_Toc167642495)

[1.1.5. Отмена изменений и откат к предыдущему состоянию 9](#_Toc167642496)

[1.1.6. Игнорирование 10](#_Toc167642497)

[1.1.7. Русификация и выбор языка 11](#_Toc167642498)

[1.1.8. Шифрование и дешифрование 12](#_Toc167642499)

[1.1.9. Управление VCS через командную строку 13](#_Toc167642500)

[1.2. Реализация сервера VCS 13](#_Toc167642501)

[1.2.1. Архитектура сервера 13](#_Toc167642502)

[1.2.2. Хранение данных на сервере 14](#_Toc167642503)

[1.2.3. Реализация пользовательского интерфейса 15](#_Toc167642504)

[1.3. Интеграция клиента и сервера VCS 23](#_Toc167642505)

[1.3.1. Отправка репозитория на сервер 23](#_Toc167642506)

[1.3.2. Загрузка удаленного репозитория с сервера на клиент 25](#_Toc167642507)

[2. Тестирование приложения 27](#_Toc167642508)

[2.1. Тестирование клиента 27](#_Toc167642509)

[2.2. Тестирование сервера 27](#_Toc167642510)

[Заключение 30](#_Toc167642511)

[Список использованных источников 31](#_Toc167642512)

[ПРИЛОЖЕНИЕ 1 32](#_Toc167642513)

[ПРИЛОЖЕНИЕ 2 34](#_Toc167642514)

[ПРИЛОЖЕНИЕ 3 36](#_Toc167642515)

[ПРИЛОЖЕНИЕ 4 37](#_Toc167642516)

[ПРИЛОЖЕНИЕ 5 38](#_Toc167642517)

[ПРИЛОЖЕНИЕ 6 39](#_Toc167642518)

АННОТАЦИЯ

Целью данной дипломной работы является разработка современной системы контроля версий (VCS). Система включает в себя ряд уникальных функций, таких как упрощенный набор команд для снижения порога вхождения для новых пользователей, поддержка различных языков интерфейса, в том числе русского, и шифрование данных репозитория для обеспечения безопасности.

Система контроля версий состоит из двух основных частей:

– Клиентская часть - предоставляет пользователям возможность взаимодействовать с VCS через набор команд, локально, у себя на компьютере. Разработана на Python с использованием библиотек difflib (для сравнения последовательностей и генерации патчей) и hashlib (для получения hash значений)

– Серверная часть - центральное место для хранения всех репозиториев всех пользователей. Сервер управляет хранением, извлечением и обновлением данных репозиториев. Разработана на Python с помощью веб-фреймворка Django и библиотеки Bootstrap для создания современного дизайна. В качестве БД была использована SQLite.

Благодаря правильной архитектуре и низкой степени связанности между компонентами, части системы могут функционировать независимо друг от друга, в автономном режиме.

Ключевые слова: Python, Django, SQLite, Bootstrap

# Введение

Целью производственной технологической (проектно-технологической) практики направления подготовки 09.03.03 «Прикладная информатика» согласно программе практики является закрепление теоретических знаний, практических умений и навыков, полученных в процессе освоения дисциплин основной образовательной программы приобретение практического опыта решения профессиональных задач. При прохождении данной практики обучающийся опирается на материалы ранее освоенных дисциплин (модулей) и/или практик: Информатика, Информационные технологии, Архитектура информационных систем, Управление данными, Технологии программирования, Ознакомительная практика.

Освоение данной практики способствует эффективному выполнению и защите выпускной квалификационной работы.

Основными задачами производственной технологической (проектно-технологической) практики являются:

-овладеть навыками использования отечественных и международных стандартов, норм и правил; навыками разработки технической документации на всех стадиях жизненного цикла информационной системы;

- овладеть практическими навыками разработки программного обеспечения;

- овладеть навыками выполнения работ на всех стадиях жизненного цикла создания информационных систем; навыками использования инструментальных средств управления проектами в области разработки информационных систем.

Технологической (проектно-технологической) практика проходила на базе Института вычислительной математики и информационных технологий Казанского (Приволжского) федерального университета.

Сроки прохождения практики: с 08.04.2024 по 23.05.2024.

1. Разработка системы контроля версий

1.1. Реализация клиента VCS

1.1.1. Хранение объектов файловой системы

Разработку клиента я начал с поиска и создания инструментов, которые отслеживают информацию об изменениях файлов и папок и хранят эти данные. За основу я взял модель хранения данных в Git репозиториях.

Для понимания, что файл был изменен, можно использовать hash функции, которые вычисляют уникальный hash объекта, исходя из его содержимого. Для получения hash файла я складывал значение hash Blob’a (объект, хранящий содержимое файла) с именем файла. Таким образом, если файл будет переименован или изменено содержимое, у него будет новый hash.

Для хранения информации о каталоге был создан класс Tree, который хранит список файлов (объектов File) и подкаталогов (объектов Tree). Hash каталога вычисляется как сумма hash значений всех вложенных в него объектов + имя каталога.

Чтобы сохранить текущее состояние проекта, создается объект Commit – снимок проекта в определенный момент времени. Commit хранит ссылку на каталог проекта (Tree), ссылку на предыдущий коммит (если он был), а также имя автора, сообщение и время создания. При сравнении коммитов сравниваются только объекты Tree без учета времени и предыдущих коммитов.

Все перечисленные классы (Blob, File, Tree, Commit) имеют методы save() и load() для сохранения в виде файла и обратной загрузки. При сохранении каждый объект записывает бинарную информацию в файл репозитория. В качестве имени файла используется его hash значение. Таким образом, если есть несколько файлов с одинаковым содержимым, у них будут одинаковые Blob объекты и в файловой системе будет храниться только один файл Blob’a, на который будут указывать несколько объектов File.

Проблема выбранного способа хранения заключается в том, что может возникнуть ситуация, когда у двух разных файлов будет одинаковый hash и они будут сохранены как один файл. Это связано с ограниченным количеством hash значений, которые можно использовать (поскольку длина hash строки ограничена). В моей программе используется алгоритм хеширования SHA-1, который возвращает строку из 40 шестнадцатеричных символов (160 бит), что дает различных комбинаций. Вероятность описанной ситуации довольно мала, но есть. Можно увеличить размер hash до 256 или 512 бит, но это может повлечь к проблемам, связанным с ограничением на длину имен файлов и длину полного пути к файлу.

Листинг кода получения hash значений и классов, представляющих объекты файловой системы, в Приложении 1.

1.1.2. Структура хранения файлов в репозитории

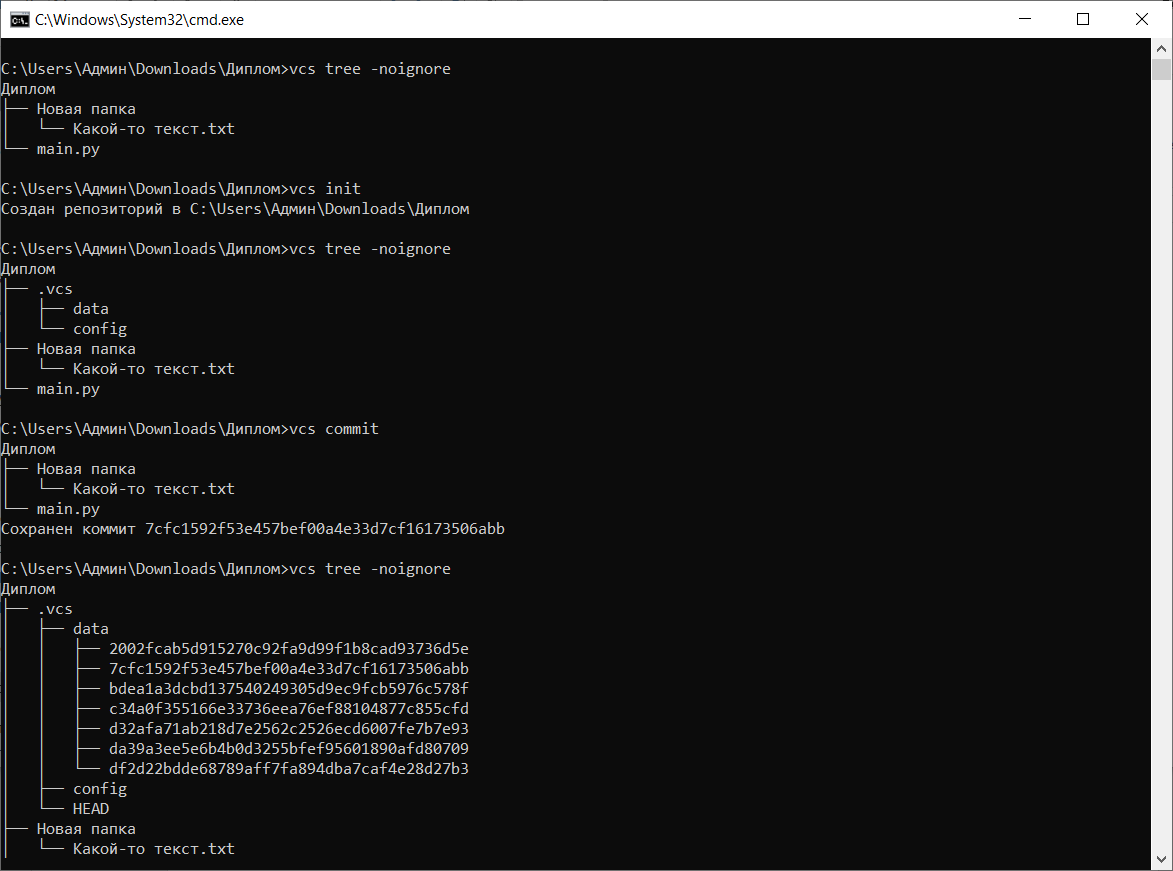


Рисунок 1. Пример хранения данных репозитория

На рисунке 1 можно увидеть, как через консоль создаются коммит в папке «Диплом», в которой изначально хранится одна папка «Новая папка» и два текстовых файла.

Сначала выводится файловая структура корневой папки с помощью команды vcs tree –noignore. Флаг –noignore нужен для того, чтобы не игнорировались файлы и папки. После, командой vcs init инициализируется репозиторий. Создается папка репозитория .vcs, в этой папке находится папка data, куда будут сохраняться все объекты файловой системы и изменения. Также создается файл config, в котором хранится информация об используемом языке и в дальнейшем будет храниться информации о пользователе (имя, почта, название удаленного репозитория, токен) после первой отправки файлов на сервер VCS. Коммит создается командой vcs commit, после чего все объекты (Blob, File, Tree, Commit) сохраняются в папке .vcs/data. На рисунке 1 видно, как было создано 7 файлов: 2 файла объектов Tree (папки «Диплом», «Новая папка»), 2 файла объекта File («Какой-то текст.txt», «main.py»), 2 файла объекта Blob (содержимое файлов) и 1 файл объекта Commit. Имена файлов совпадают с их хешами. Также был создан файл HEAD, в котором теперь будет храниться hash текущего коммита (для проверки изменений и проч.)

1.1.3. Сравнение файлов и создание патчей

После того, как по hash значениям мы поняли, что в файл был изменен, нужно получить список этих изменений (разница между старой и новой версиями файлов). Для получения этих изменений и работы с ними я использовал библиотеку difflib. Она хорошо подходит для сравнения последовательностей символов и генерации патчей. Это процесс определения и записи изменений, внесенных в файл, по сравнению с его предыдущей версией. Патч содержит информацию о различиях между двумя версиями файла и может быть применен к исходной версии файла для преобразования его в новую версию. Библиотека difflib основывается на алгоритме Мэйерса и используется для нахождения наименьшего количества операций (вставка, удаление, замена), необходимых для преобразования одной последовательности в другую. Алгоритм Мэйерса работает за время O(ND), где N — длина последовательностей, а D — размер минимального набора операций для преобразования одной последовательности в другую.

Используя данные инструменты, мы решаем сразу две проблемы – определение различий между файлами и уменьшение размера хранимых файлов (без использования данного алгоритма пришлось бы хранить обе версии файла, а с ним мы храним изначальную версию файла и список изменений).

1.1.4. Сравнение деревьев и коммитов

В начале создается текущий коммит (снимок) проекта. Алгоритм проходится по всем объектам файловой системы, создает объекты классов Blob, File, Tree, описанных ранее. После создания коммита происходит его сравнение с предыдущим коммитом. Это происходит, как уже говорилось выше, путем сравнения деревьев коммитов (корневой каталог проекта). Сравниваются их hash значения и имена. При наличии изменений алгоритм рекурсивно сравнивает подкаталоги и файлы, сохраняя список изменений и возвращая его.

На рисунке 2 можно посмотреть пример получения изменений в репозитории. Был добавлен файл «новый файл.bmp», переименован «main.py» в «main2.py» и изменен «Какой-то текст.txt». После выполнения команды мы видим список изменений между текущим состоянием проекта и предыдущим.

Листинг кода, осуществляющего сравнение коммитов, в Приложении 2.

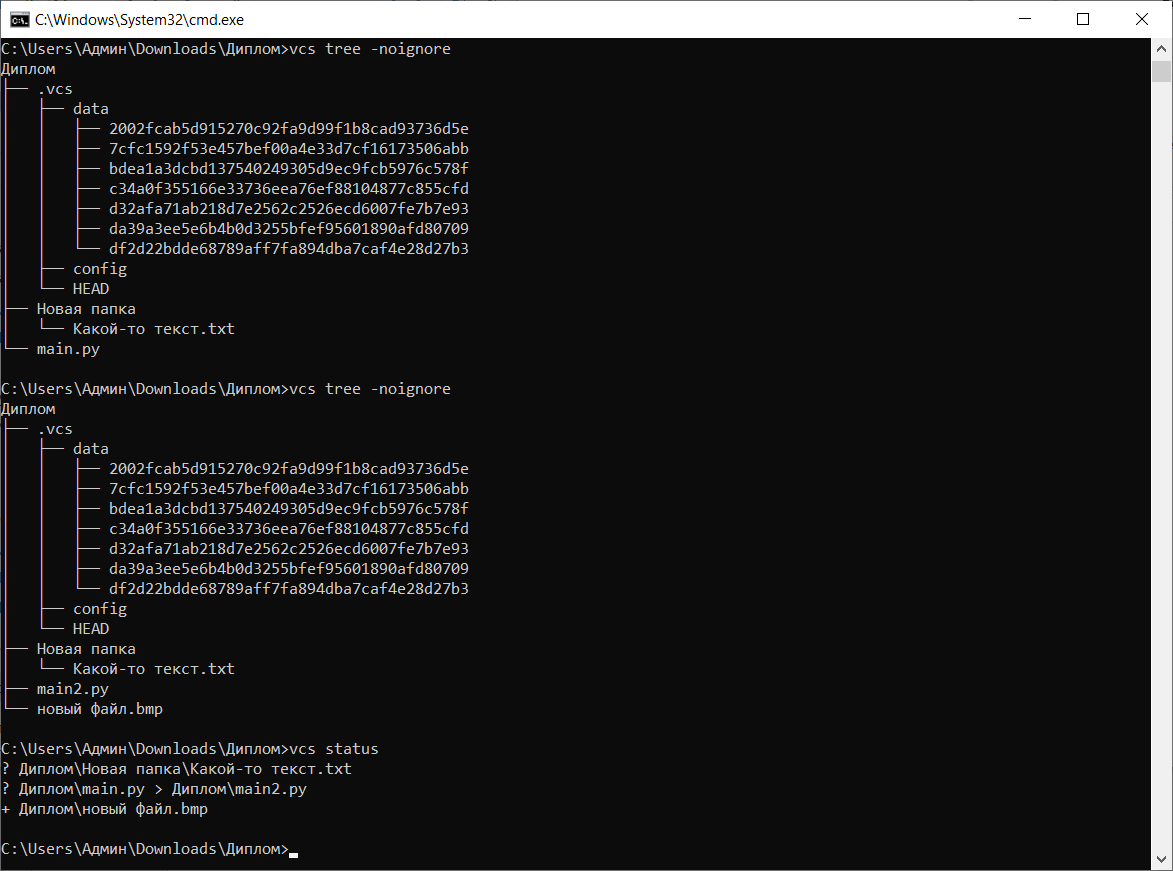


Рисунок 2. Пример изменения файлов репозитория

1.1.5. Отмена изменений и откат к предыдущему состоянию

Представим ситуацию, что мы допустили ошибку и хотим вернуться к предыдущей версии нашего проекта. Для начала просмотрим историю изменений с помощью команды vcs hist. Копируем hash последнего коммита и возвращаемся к нему командой vcs rollback <hash>. После этого проверяем, что все файлы вернулись в прежнее состояние.

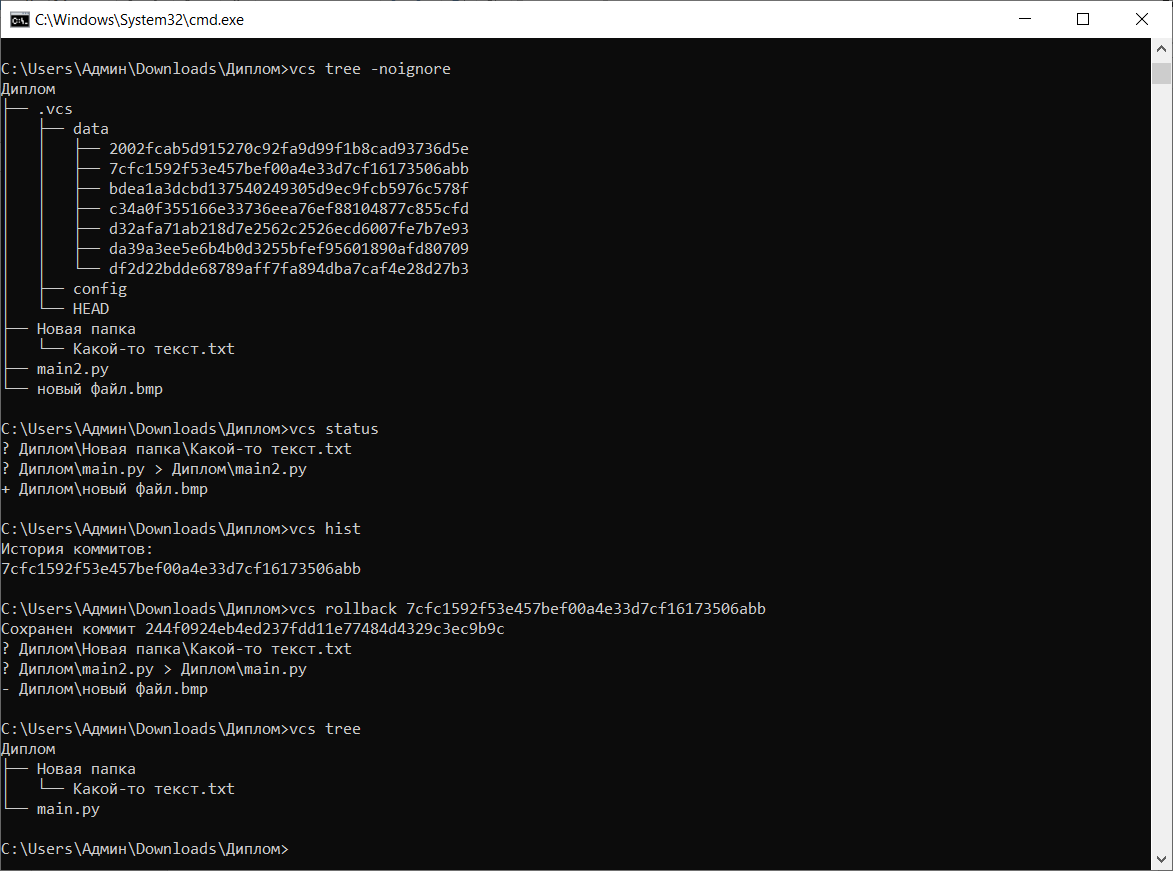
Листинг кода, который возвращает в состояние предыдущего коммита, в Приложении 3. Данная функция получает список изменений между коммитами, вызывая функцию из Приложения 2, удаляет новые и измененные файлы и папки после нового коммита, после чего проходится по дереву старого коммита и восстанавливает удаленные и измененные файлы и папки.

Рисунок 3. Пример отката изменений

1.1.6. Игнорирование

Одной из важных и полезных фич является игнорирование файлов и папок, чтобы не отслеживать их историю. Это могут быть файлы БД, файлы с ключами и токенами доступа, которые в принципе не должны появиться в открытом доступе. Также желательно игнорирование файлов библиотек, фреймворков, служебных файлов, виртуальных сред и так далее. Я не стал придумывать новый способ игнорирования, а просто перенял старый. То есть пользователь может добавить файл .gitignore в любую папку проекта, и файлы и папки, перечисленные в этом файле или совпадающие с указанными там шаблонами, будут проигнорированы. Я использовал библиотеку gitignore\_parser, которая проверяет, нужно ли игнорировать файл, и немного изменил одну из ее функций, чтобы данная библиотека поддерживала названия файлов и папок на русском языке.

Так как файлов .gitignore может быть несколько с разными уровнями вложенности, необходимо собирать все .gitignore файлы, которые находятся по пути к файлу или папке, чтобы проверить, игнорировать ее или нет.

1.1.7. Русификация и выбор языка

Для того чтобы пользователи без знания английского могли использовать VCS и не испытывать языковых сложностей, я решил добавить русификацию системы и возможность выбора языка.

На рисунке 4 можно увидеть пример переключения языка. Переведена не только справка, но и все возможные подсказки и информационные сообщения, отображающиеся у пользователя. При желании можно добавить в систему и другие языки, достаточно изменить файл с переводами.

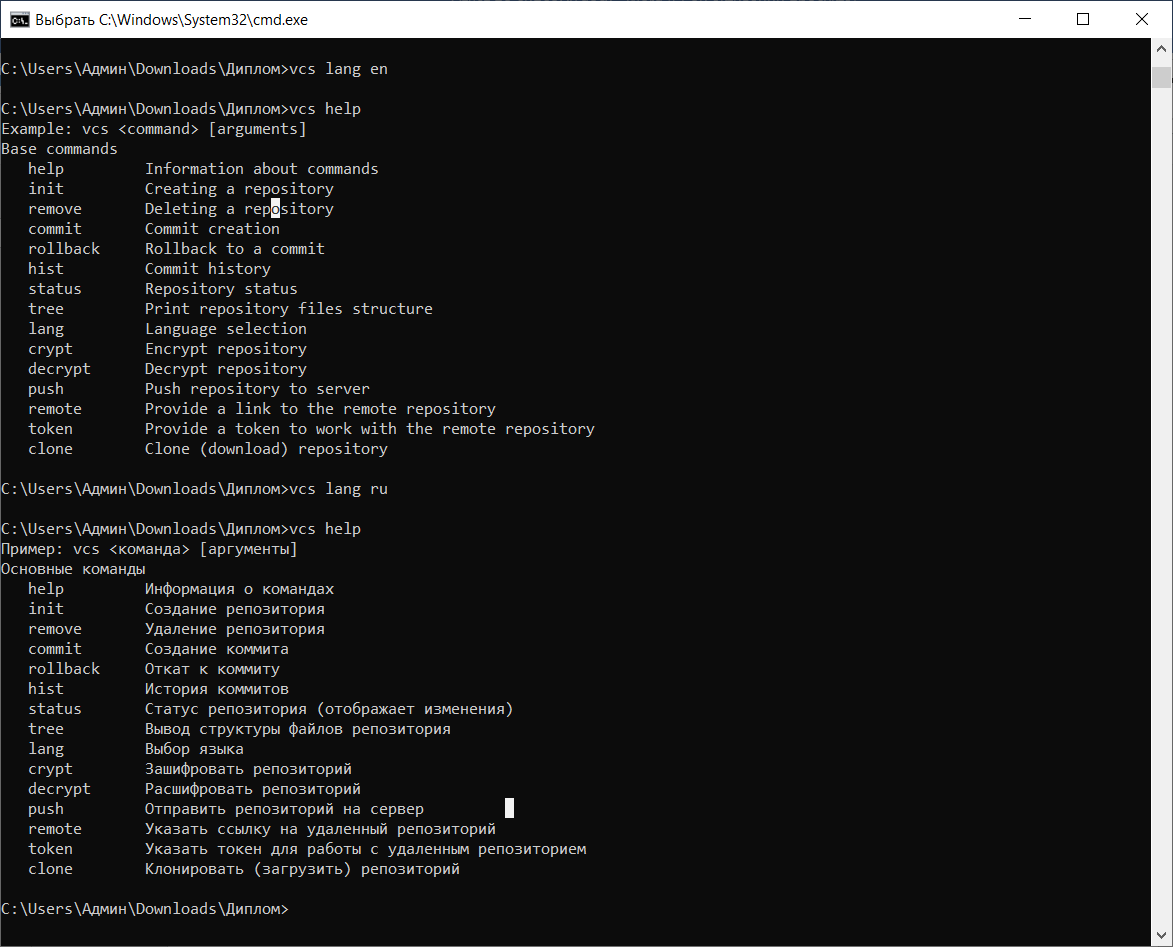


Рисунок 4. Выбор языка

1.1.8. Шифрование и дешифрование

Для обеспечения безопасности данных пользователей я решил создать возможность шифрования файлов всего репозитория и дешифрования их при наличии ключа. В качестве алгоритма шифрования я выбрал AES (Advanced Encryption Standard) – симметричный алгоритм шифрования, который стал стандартом для шифрования данных. У данного алгоритма достаточно высокий уровень безопасности и хорошее быстродействие. Как видно на рисунке 5, помимо шифрования содержимого файлов, шифруются их имена (более коротким ключом, чтобы не превысить ограничение на длину имен файлов), что обеспечивает неплохую защиту от несанкционированного доступа. Шифрование осуществляется командой vcs crypt, после чего выводится сгенерированный ключ, который нужно указать при дешифровании командной vcs decrypt.

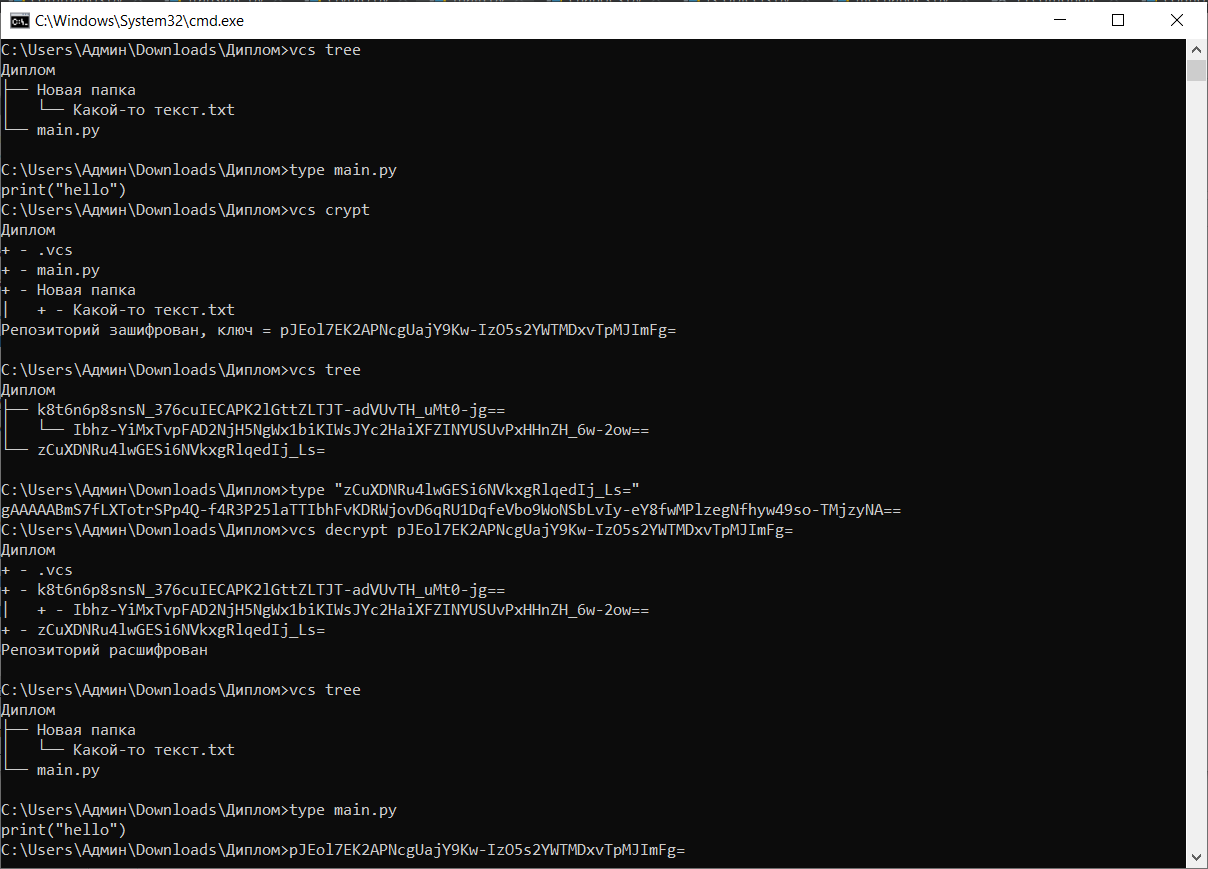


Рисунок 5. Шифрование данных репозитория

1.1.9. Управление VCS через командную строку

Чтобы была возможность управлять клиентом VCS через командную строку Windows без запуска каких-либо скриптов, проделаны следующие шаги:

- Создан файл vcs.bat, который запускает main.py файл моего проекта, который, в свою очередь, обрабатывает аргументы командной строки (команды VCS и различные параметры) и выполняет необходимые действия.

- Добавлен путь к папке, в которой находится vcs.bat, в переменную среды Path.

Таким образом, при вводе команды vcs <command> в командную строку запускается файл vcs.bat, который вызывает main.py и передает <command> в качестве аргумента.

1.2. Реализация сервера VCS

1.2.1. Архитектура сервера

В качестве веб-фреймворка, я выбрал Django. Этот фреймворк обладает множеством встроенных функций, которые позволяют быстро разрабатывать веб-приложения, например, в Django есть встроенная система аутентификации и авторизации, которую я успешно использую в своем сервере. Помимо этого, есть множество других полезных функций, для работы с учетными записями пользователей и обеспечения безопасности от XSS, CSRF атак и SQL-инъекций.

Построение архитектуры приложения на Django требует продуманного подхода, чтобы обеспечить масштабируемость, гибкость и простоту изменения существующих модулей. Основным аспектом является разделение проекта на приложения, что позволяет легко управлять кодом и повторно использовать компоненты в других проектах, а также уменьшить зависимость одного модуля от другого.

В своем проекте я создал два приложения – account (все, что связано с аккаунтами, аутентификацией, авторизацией, изменением профиля, выходом) и repository (все, что связано с репозиториями и данными в них).

Внутри приложений используется архитектура MTV (Model-Template-View), которая способствует организации кода и логики веб-приложений. Она обеспечивает четкое разделение обязанностей, что способствует упрощению разработки, тестирования и поддержки кода.

Модели (models.py) представляют данные и логику, связанную с данными, работой с БД, созданием таблиц и т.д.

Шаблоны (templates) отвечают за представление данных. Они определяют, как информация отображается пользователю. В моем случае это набор html файлов.

Представления (views.py) обрабатывают запросы и возвращают ответы. Они связывают модели и шаблоны, обеспечивая логику, необходимую для обработки запросов и отображения данных.

1.2.2. Хранение данных на сервере

Для хранения данных о пользователях я использовал встроенную модель User, которая хранит имя пользователя, пароль (в хешированном виде), почту, роль и прочие основные сведения. Но было необходимо также хранить токен, который будет использоваться для отправки запросов с клиента, для аутентификации, фото профиля и личную информацию. Для хранения этих данных я создал модель Profile, которая связана с User отношением один к одному, и создается вместе с созданием пользователя.

Для хранения имени, описания и владельца репозитория я создал модель Repository. Но данные самих репозиториев, файлы и папки проблематично хранить обычной БД, так как у этих файлов может быть большой объем и доступ к большому количеству таких данных может быть медленным. Также, моделирование структуры файловой системы внутри БД достаточно сложная задача. Исходя из этого, я принял решение хранить данные репозиториев непосредственно на сервере, в папке media проекта Django. Для удобства навигации все репозитории хранятся по следующему пути: media/files/<username>/<repository\_name>, где username – имя пользователя (уникально среди пользователей) и repository\_name – имя репозитория (уникально среди репозиториев данного пользователя). На самом сервере для доступа к репозиторию используется аналогичный URL адрес: <hostname>/<username>/<repository\_name>, hostname – имя сервера.

1.2.3. Реализация пользовательского интерфейса

Для создания структуры веб-страниц (расположение элементов, формы ввода, кнопки и прочее) использован HTML. Для придания современного и стильного вида, я использовал готовые классы и шаблоны Bootstrap, а также добавил немного собственных стилей с помощью CSS.

Для динамических элементов (выпадающие списки, раскрывающиеся папки, смена темы) был использован JavaScript. Для создания интерфейса загрузки файлов и самой отправки файлов на сервер была использована JS библиотека Dropzone.js. Также для корректной загрузки целых папок с файлами и подкаталогами пришлось написать скрипт на JavaScript, который сохраняет относительный путь каждого файла и отправляет отдельным параметром в POST запросе. Без этого не получить относительный путь файла (и не создать правильную файловую структуру проекта), так как браузер и сам Django очищают все, кроме имени файла, перед отправкой, для обеспечения безопасности. Листинг формы для загрузки файлов через браузер в Приложении 4.

Далее будет идти перечисление страниц, которые есть на сервере, их скриншоты и краткое описание.

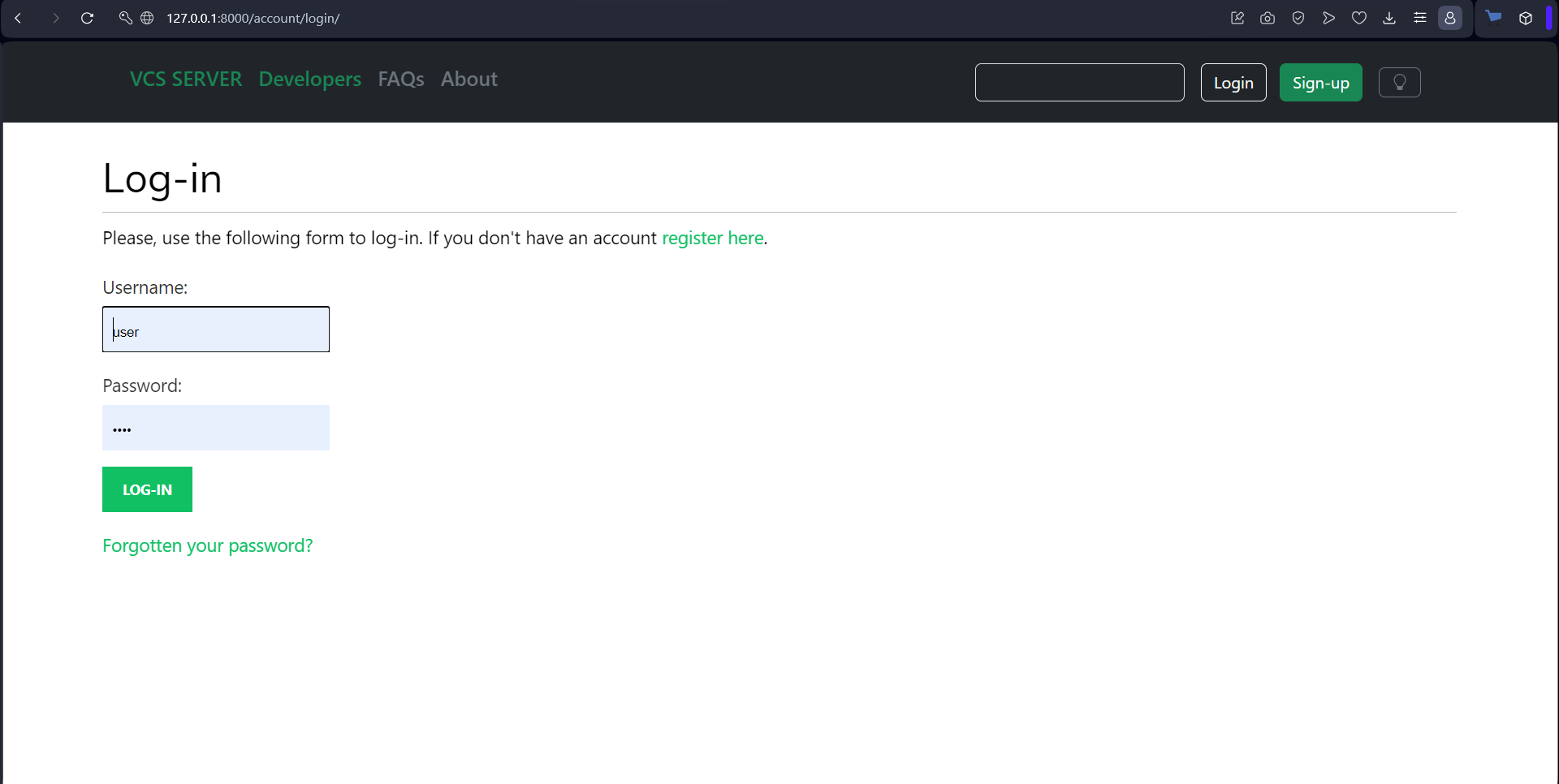


Рисунок 6. Страница входа

Как уже написано выше, для страницы входа и выхода использована встроенная механика аутентификации и авторизации. То есть мне не нужно было создавать для них представления (views.py) в Django, я лишь создал нужные формы (forms.py) и шаблоны (templates). На рисунке 6 показана страница входа. Пользователь может входить не только по имени, но и по почте, поскольку она уникальна.

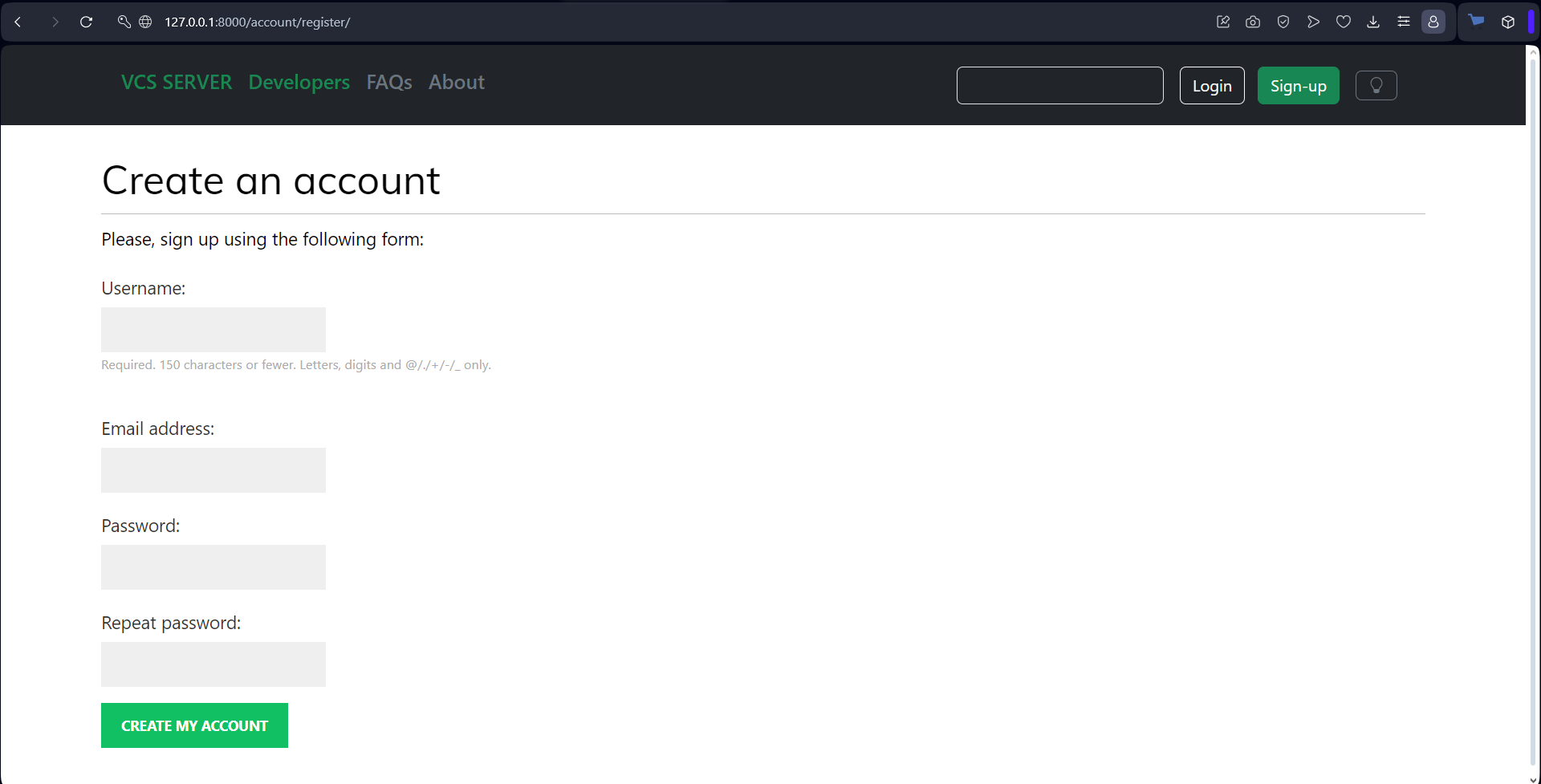


Рисунок 7. Страница регистрации

На рисунке 7 представлена страница регистрации на сайте. На рисунке 8 – страница, отображающаяся после выхода.

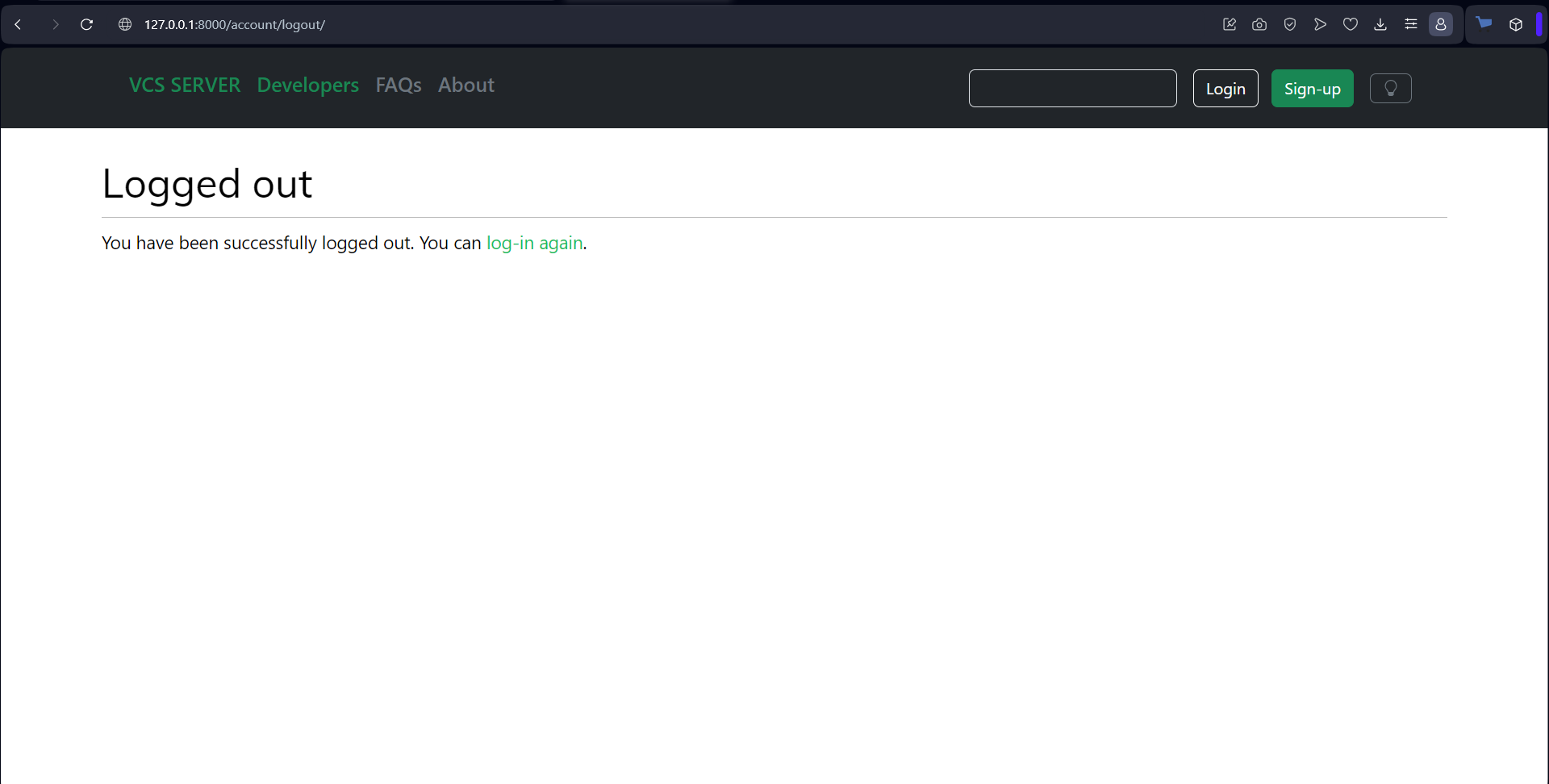


Рисунок 8. Страница выхода

На рисунке 9 показана страница редактирования профиля. Здесь пользователь может изменить свои сведения, посмотреть токен для загрузки локальных репозиториев на сервер и войти через Яндекс OAuth.

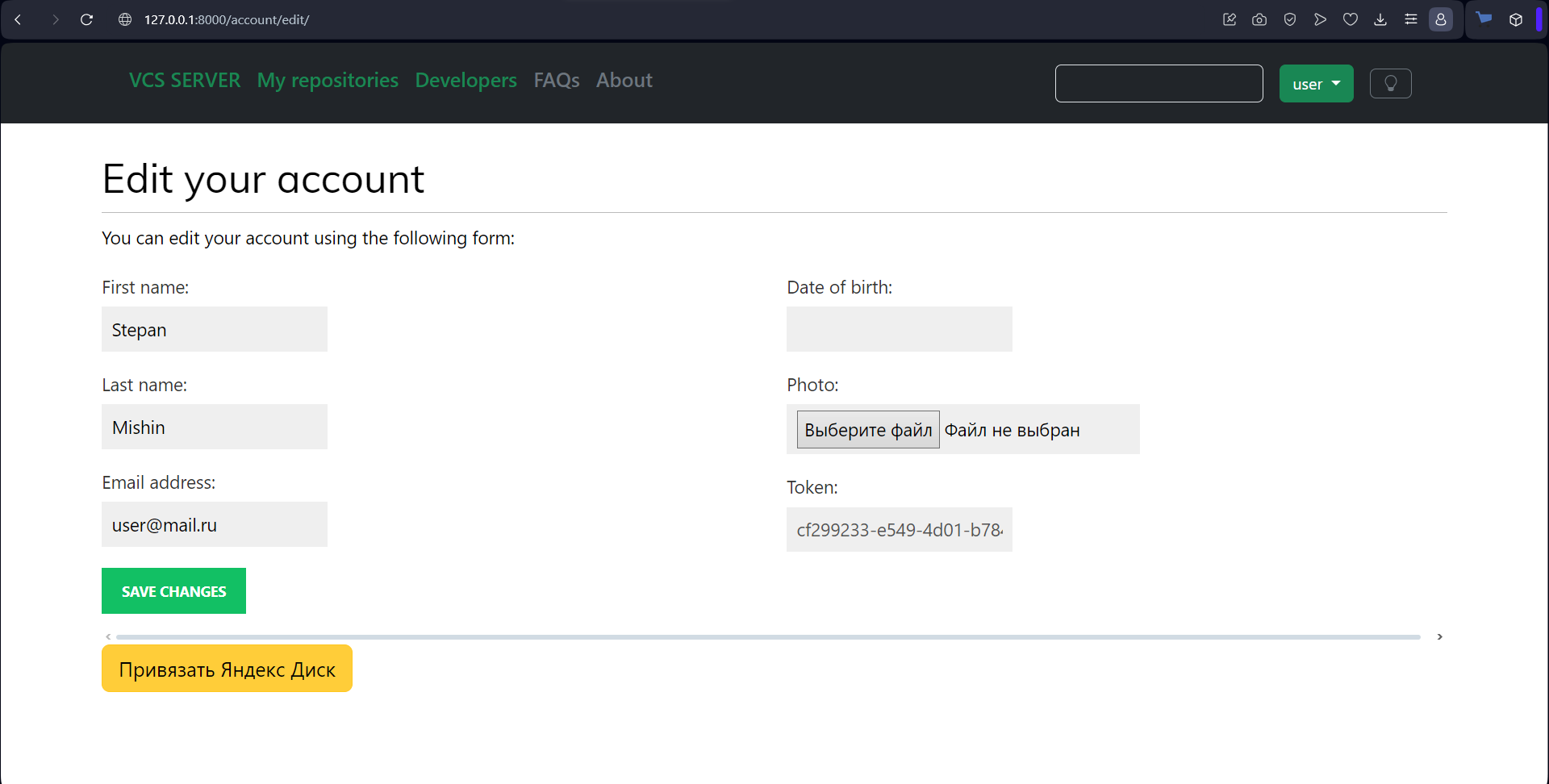


Рисунок 9. Страница редактирования профиля

На рисунке 10 показана страница изменения пароля.

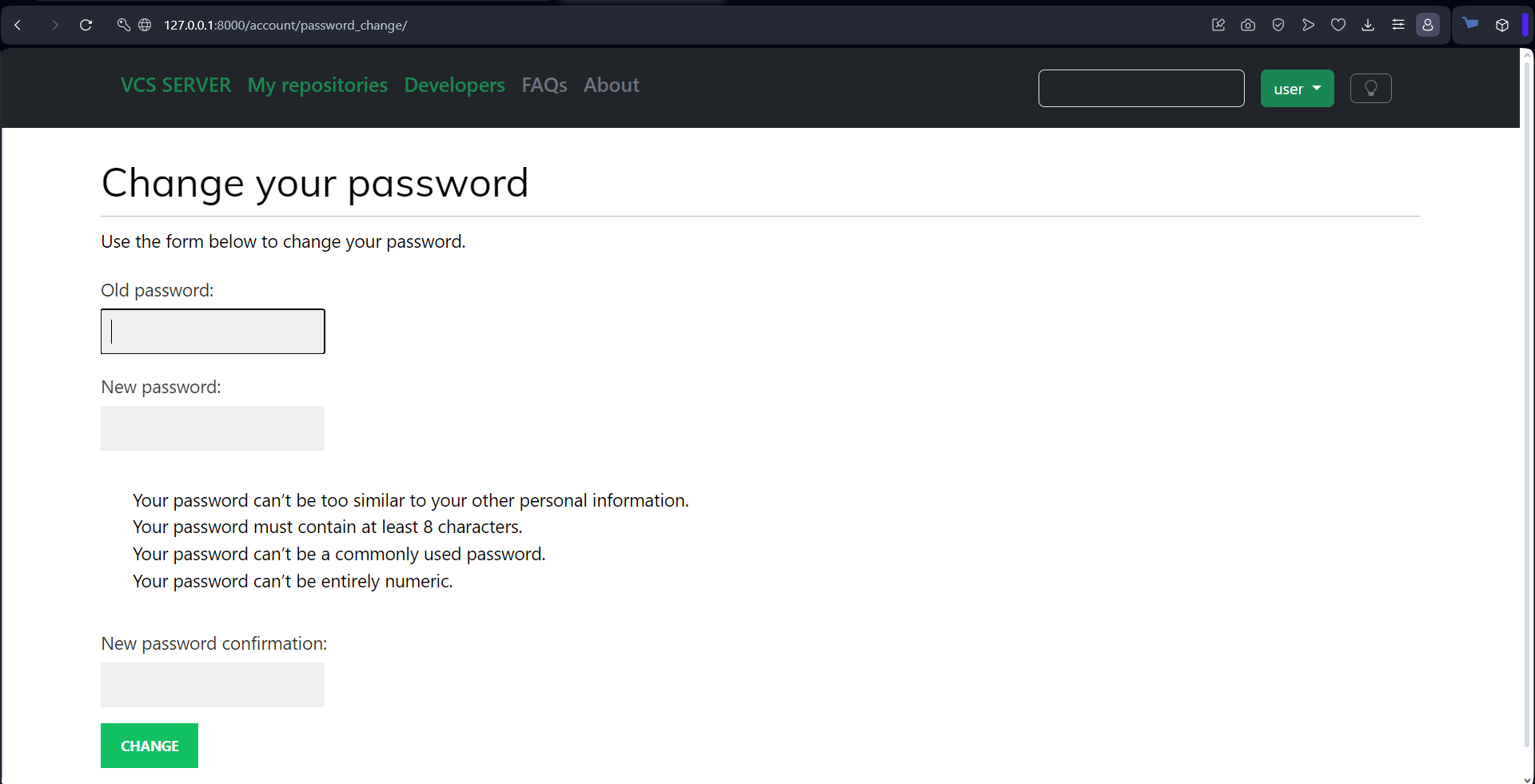


Рисунок 10. Страница изменения пароля

На рисунке 10 показана страница сброса пароля для авторизованного пользователя.

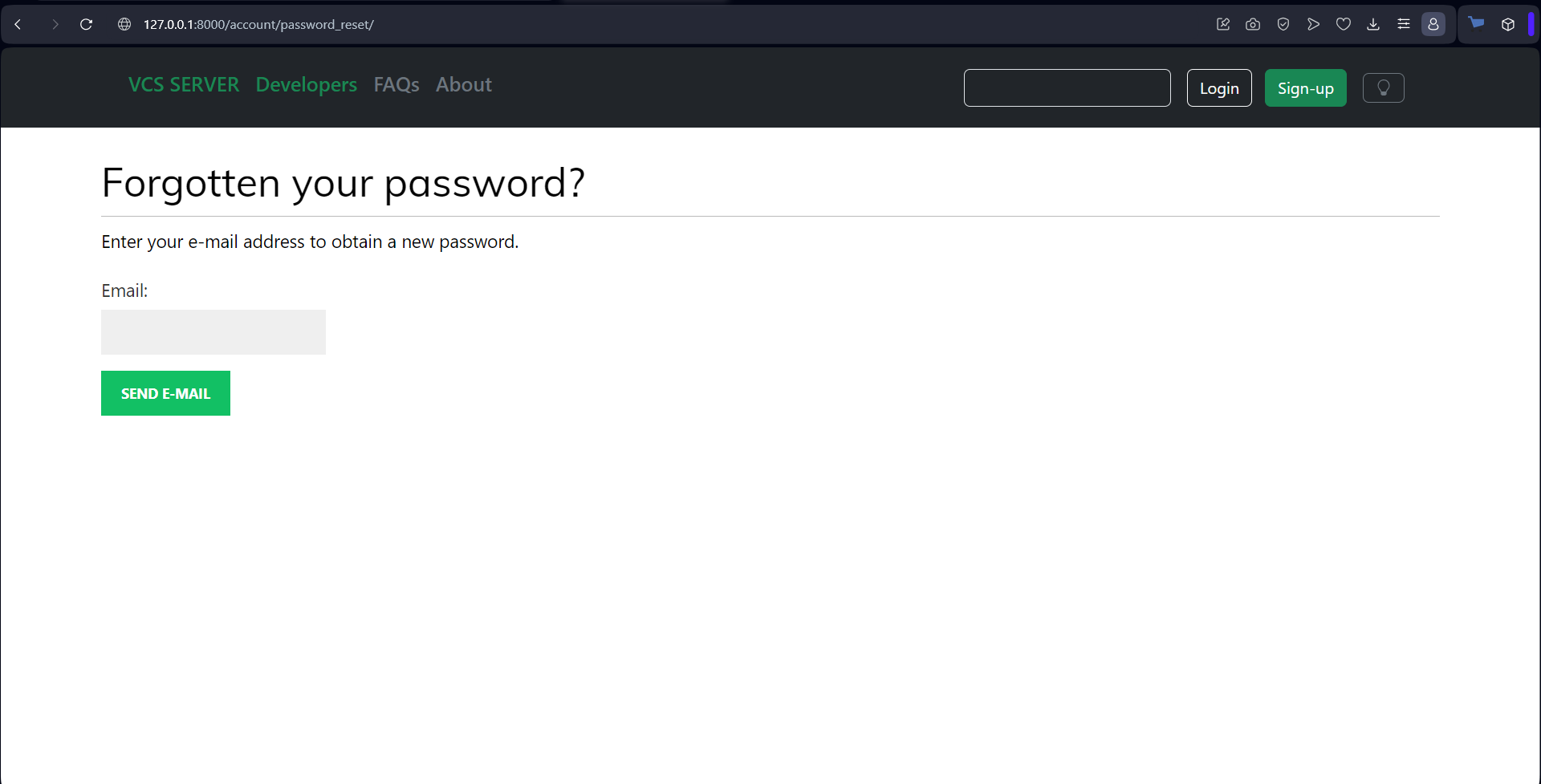


Рисунок 11. Страница сброса пароля

На рисунке 11 показана страница сброса пароля для неавторизованного пользователя. При вводе почты ссылка с уникальным токеном для сброса пароля отправляется на почту пользователя. Пример показан на рисунке 12.

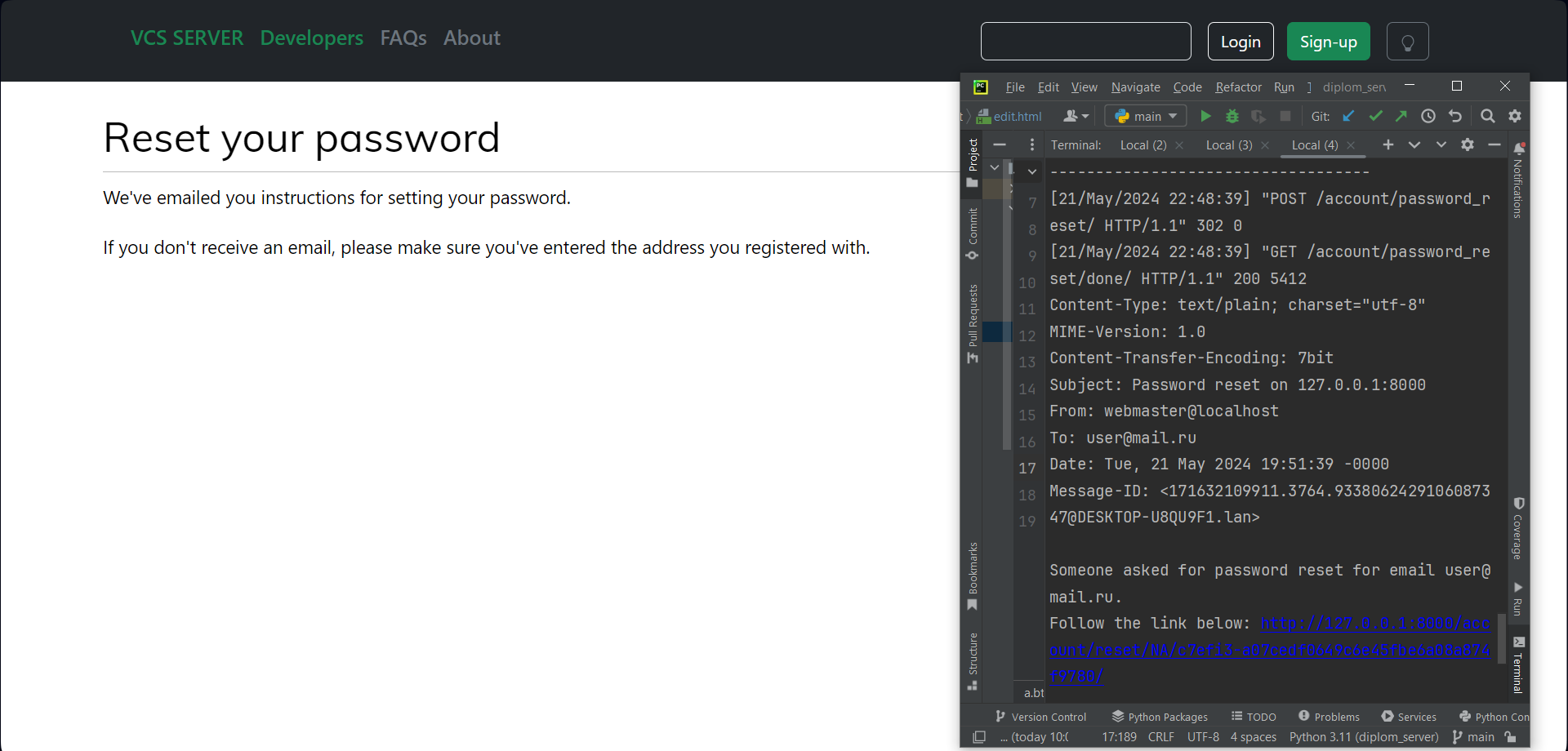


Рисунок 12. Сброса пароля с помощью почты

На рисунке 13 показана главная страница – список всех репозиториев всех пользователей.

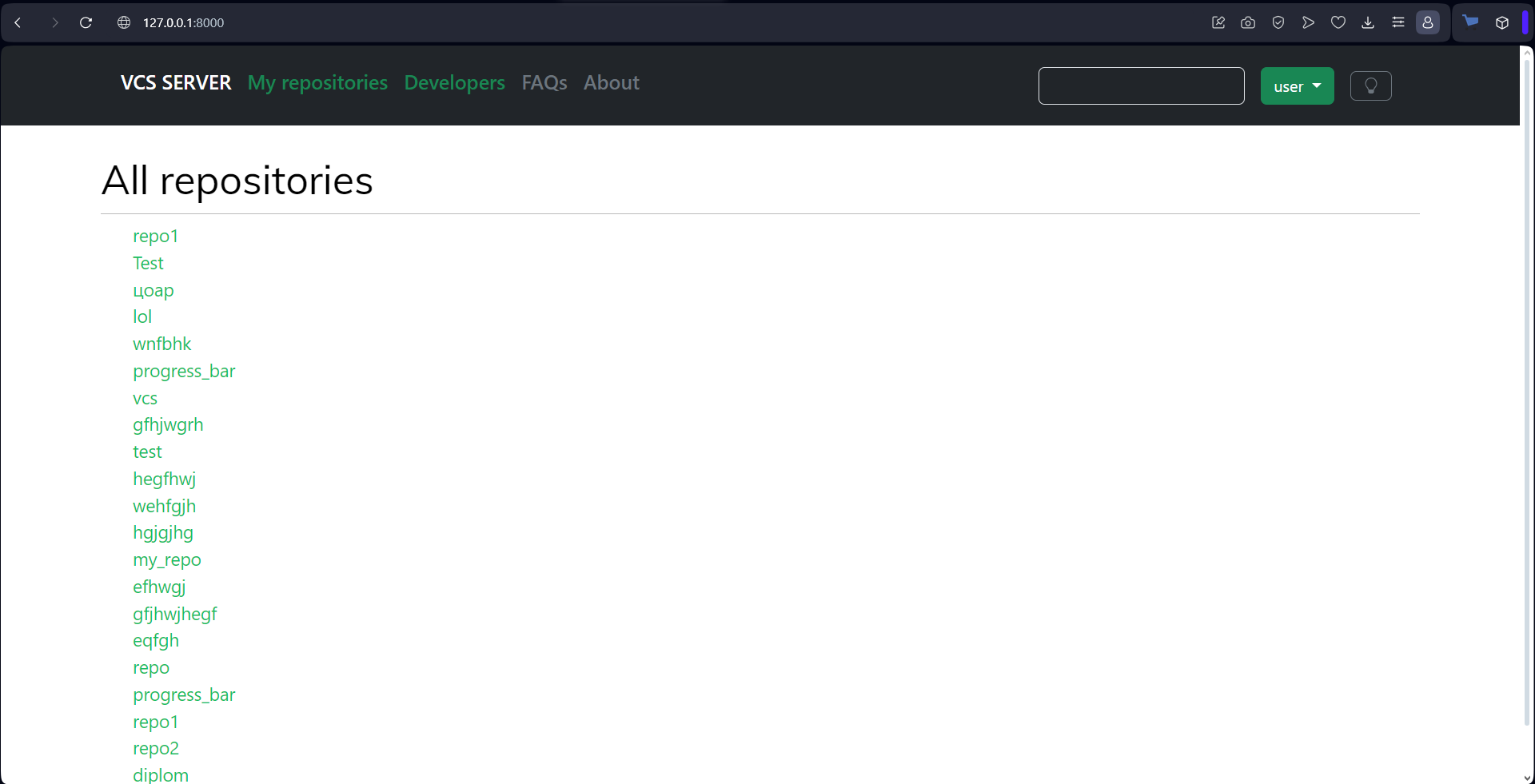


Рисунок 13. Страница со списком всех репозиториев

На рисунке 14 показана страница со списком всех пользователей.

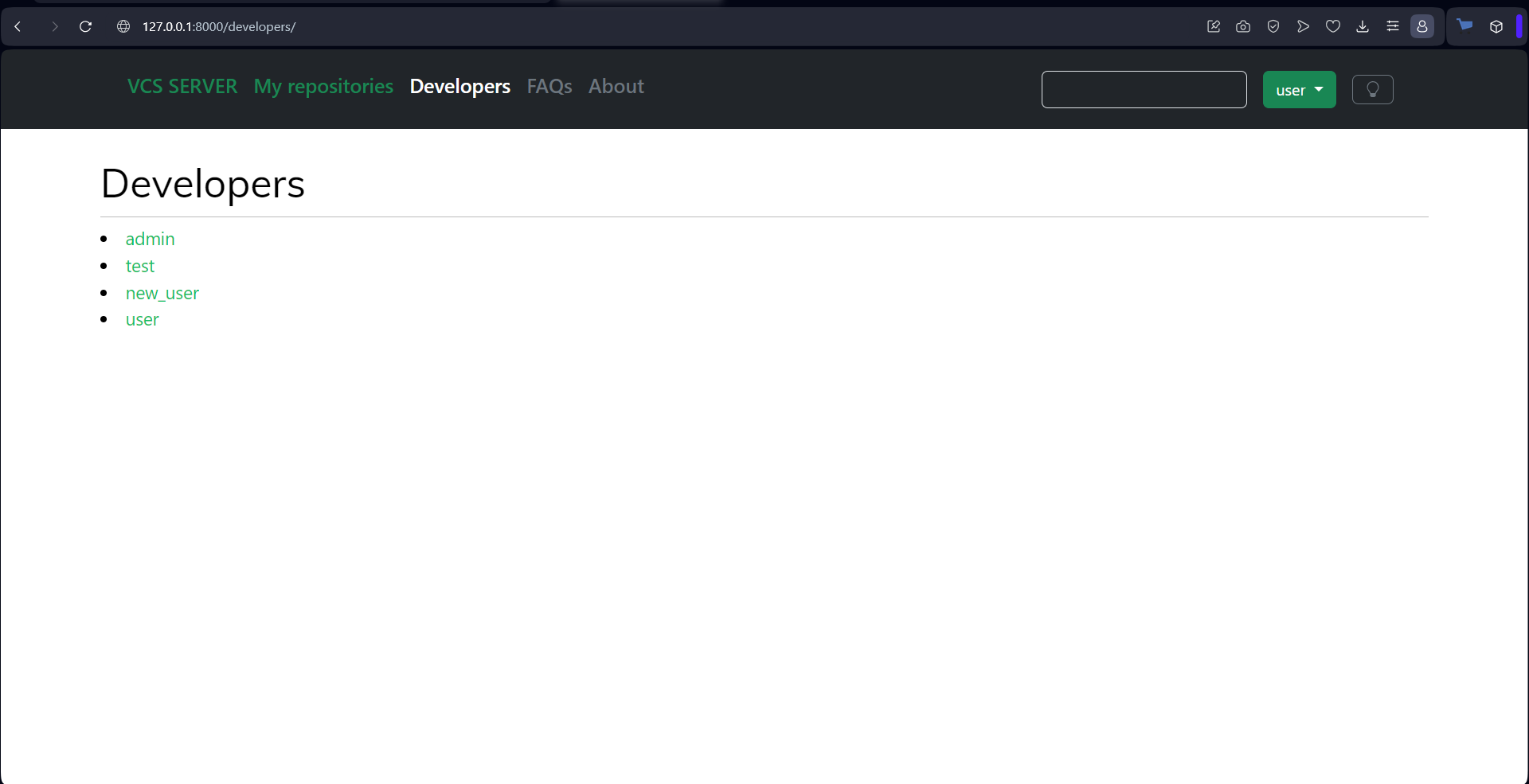


Рисунок 14. Страница со списком всех пользователей

На рисунке 15 показана страница пользователя со списком его репозиториев и кнопкой создания нового. Как видно, URL адрес совпадает с именем пользователя.

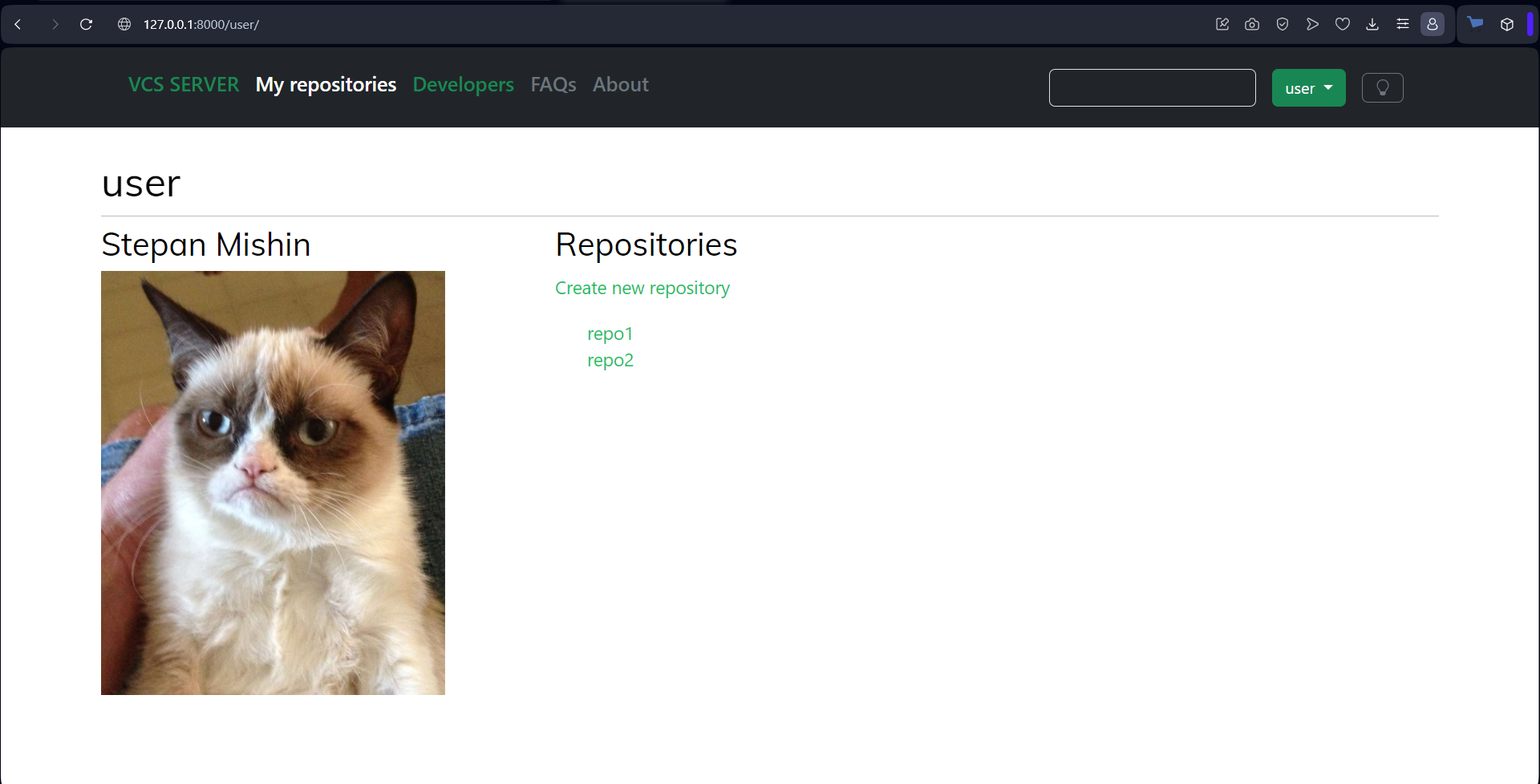


Рисунок 15. Страница пользователя

На рисунке 16 показана страница создания нового репозитория. Файлы можно загрузить двумя способами: кнопка «Выбрать файлы» открывает стандартный файловый менеджер и позволяет загрузить каталог с файлами и подкаталогами (рисунок 17). Также можно просто перетащить нужные файлы в поле drag and drop, созданное с помощью Dropzone.js, и они будут загружены на сервер.

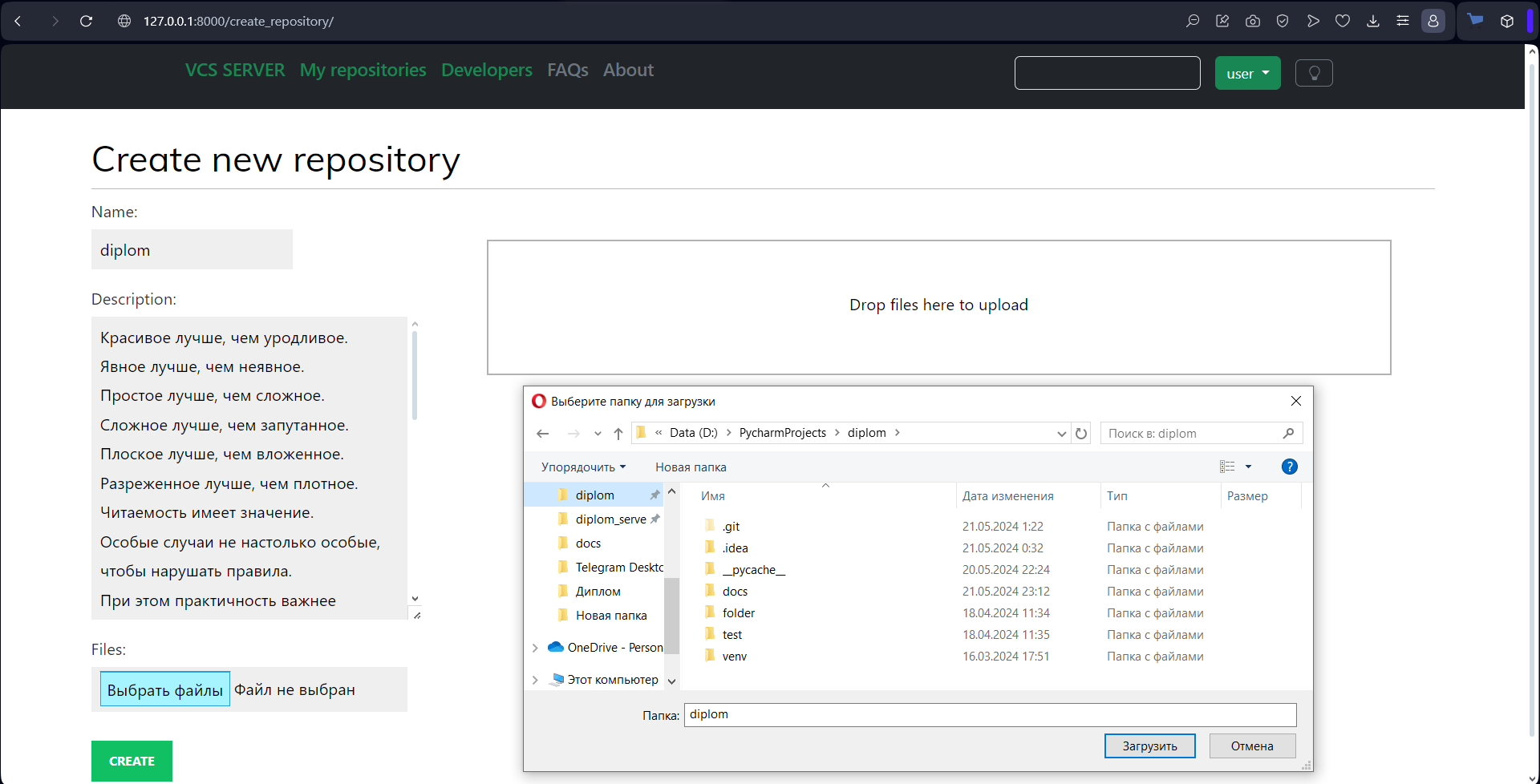


Рисунок 16. Страница создания репозитория

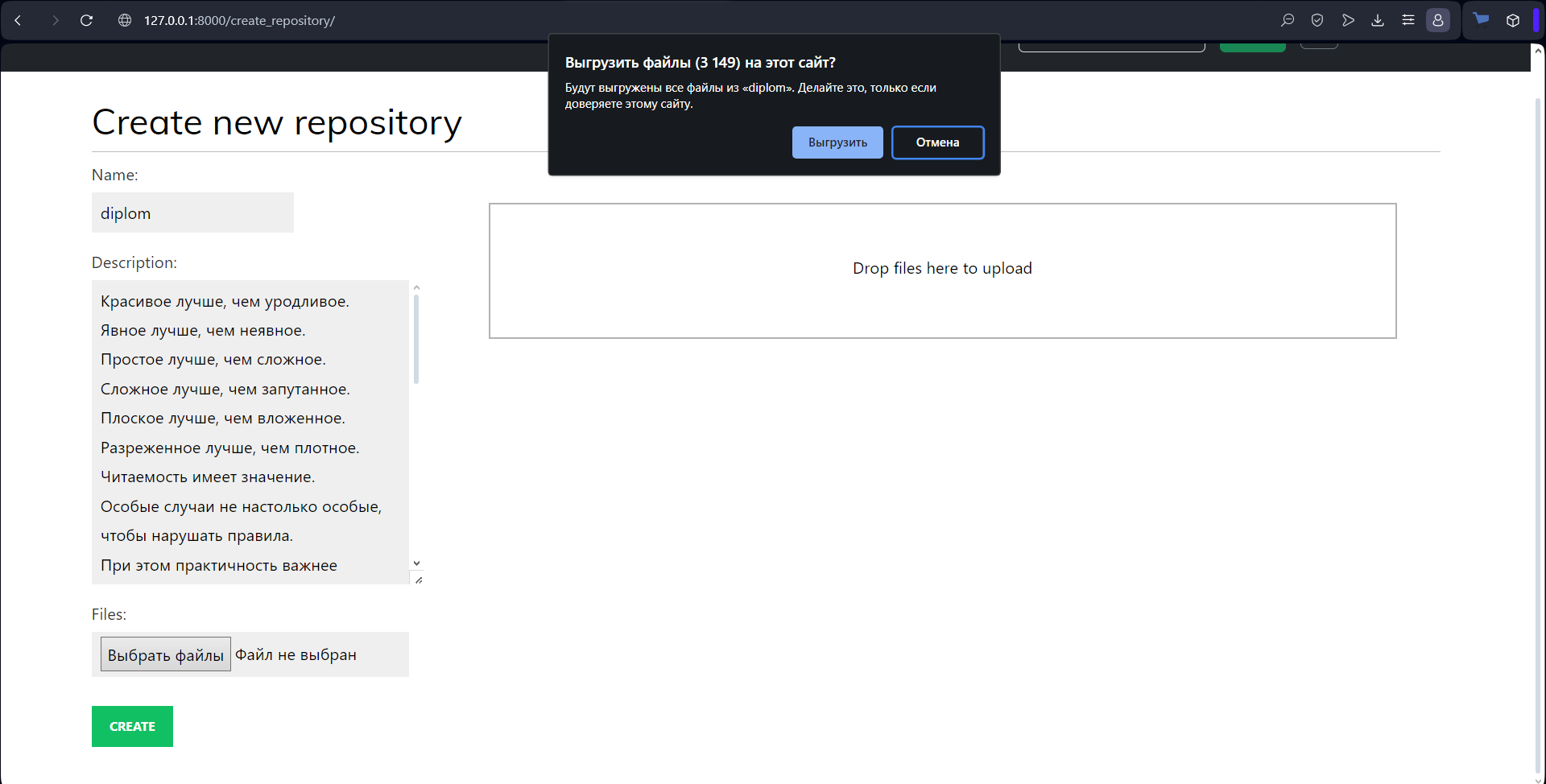


Рисунок 17. Загрузка файлов через файловый менеджер

На рисунке 18 показана страница нового репозитория. Слева отображается файловая структура проекта, справа – описание. URL адрес имеет вид <hostname>/<username>/<repository\_name>.

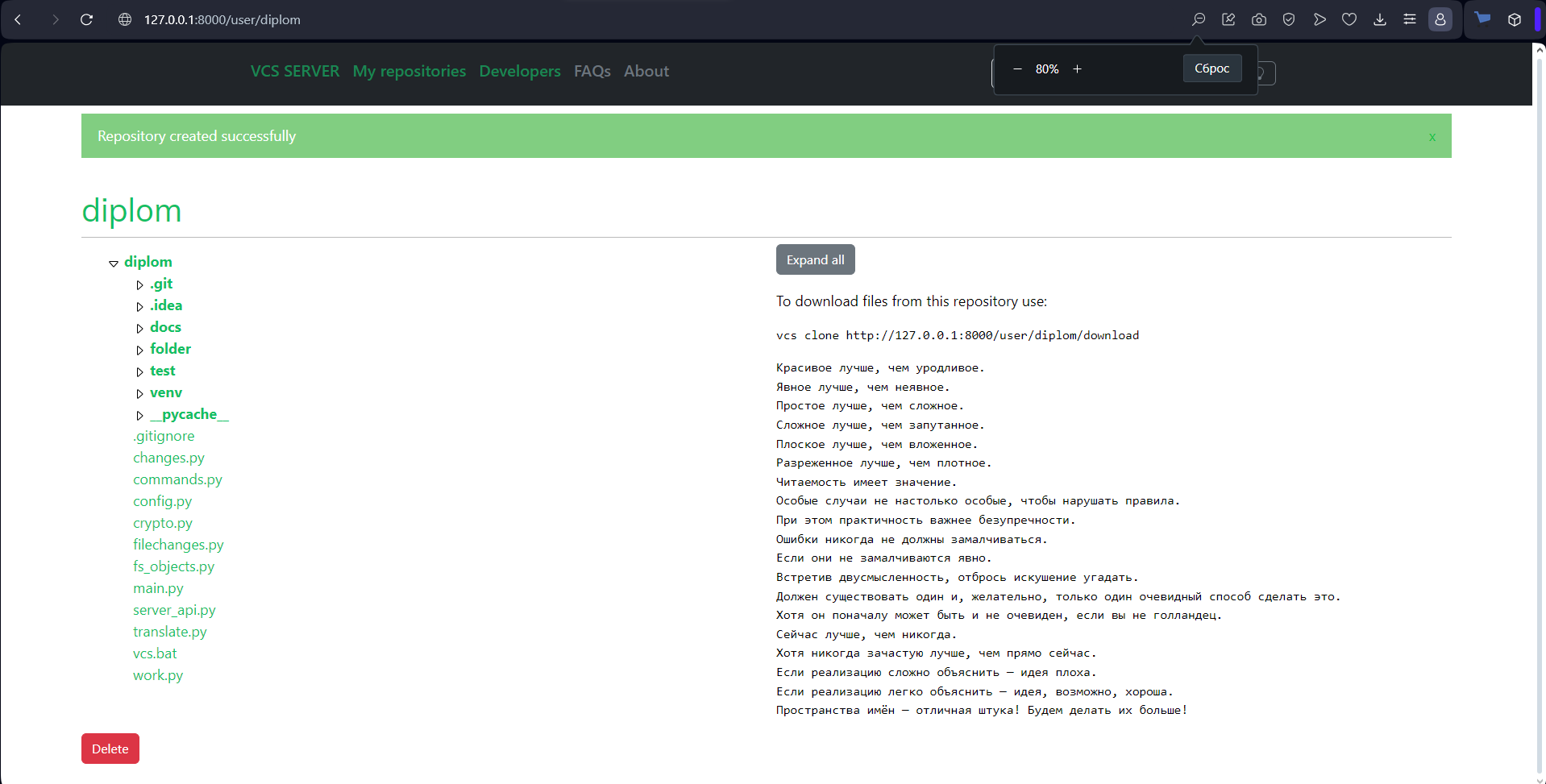


Рисунок 18. Страница репозитория

На рисунке 19 показана одна из папок нового репозитория. Сверху отображается относительны путь к данной папке из корневой. При нажатии на кнопку «Expand all» раскрывается содержимое всех папок, которые изначально свернуты. URL адрес имеет вид <hostname>/<username>/<repository\_name>/<folder\_path>, где folder\_path – относительный путь к репозиторию.

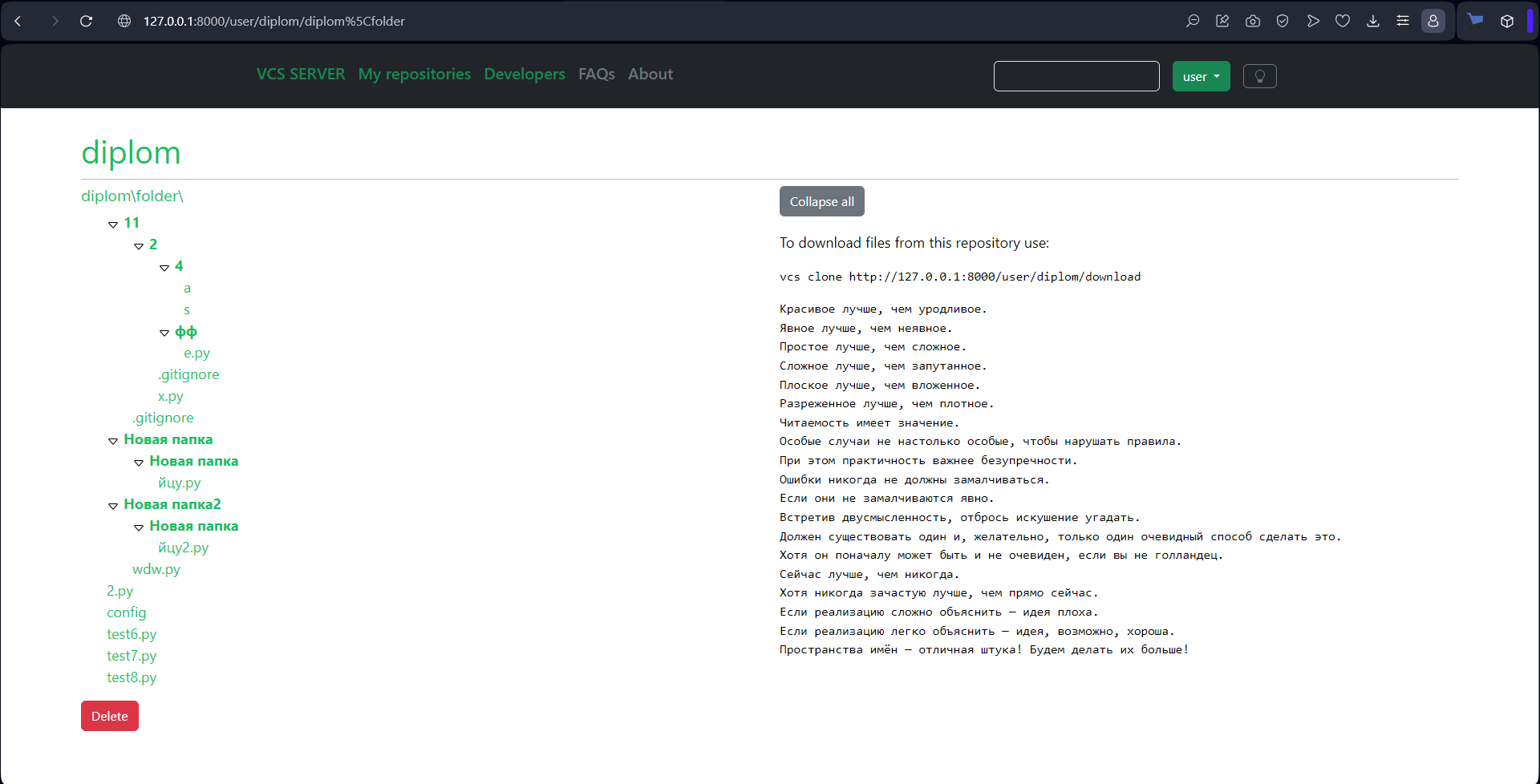


Рисунок 19. Страница папки внутри репозитория

На рисунке 20 показан один из файлов репозитория. Сверху также отображается относительный путь к файлу, размер и время изменения. Если файл является текстовым или картинкой – то отображается его содержимое. Для подсветки синтаксиса используется библиотека Prism.js

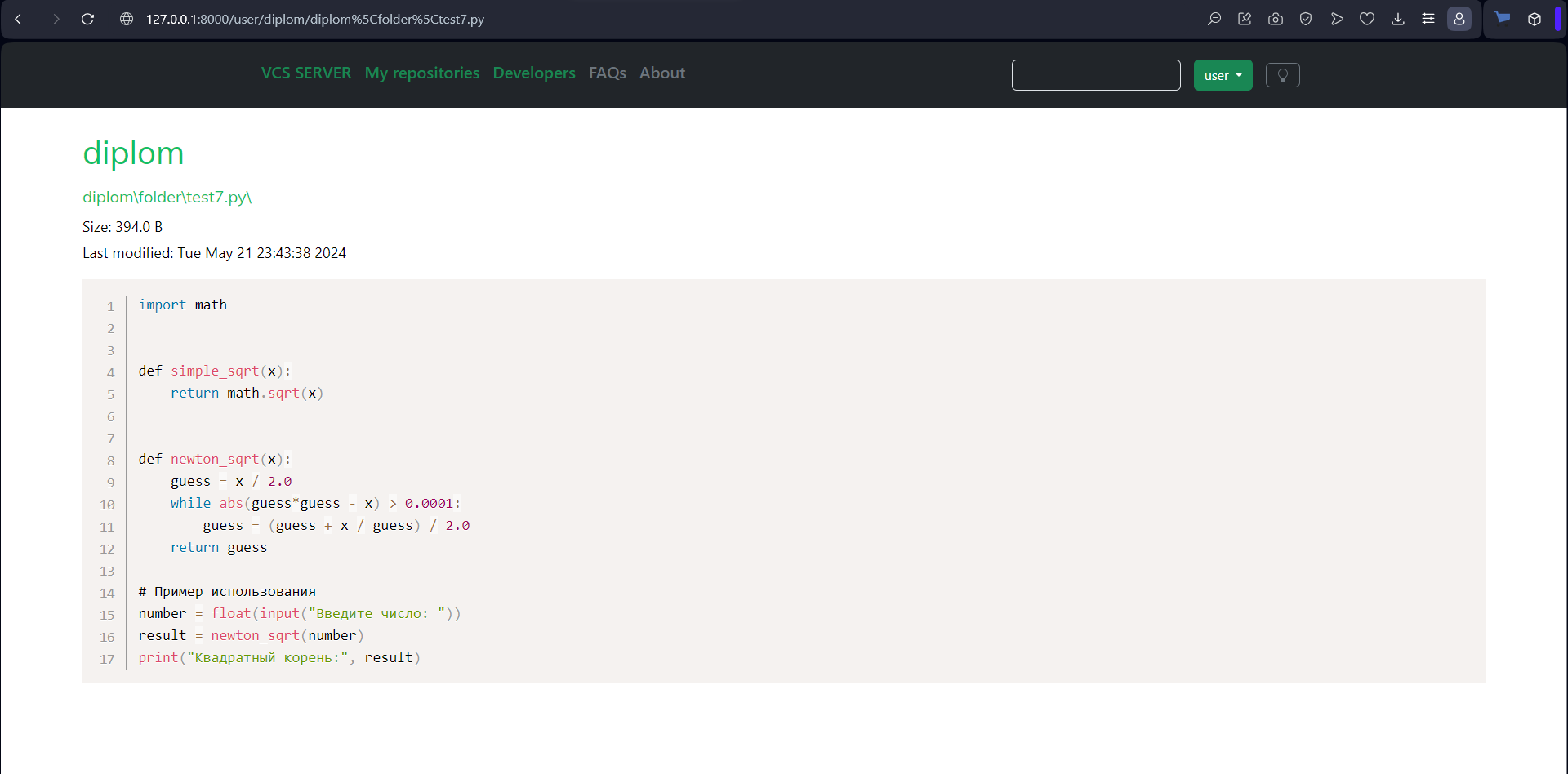


Рисунок 20. Страница файла внутри репозитория

1.3. Интеграция клиента и сервера VCS

1.3.1. Отправка репозитория на сервер

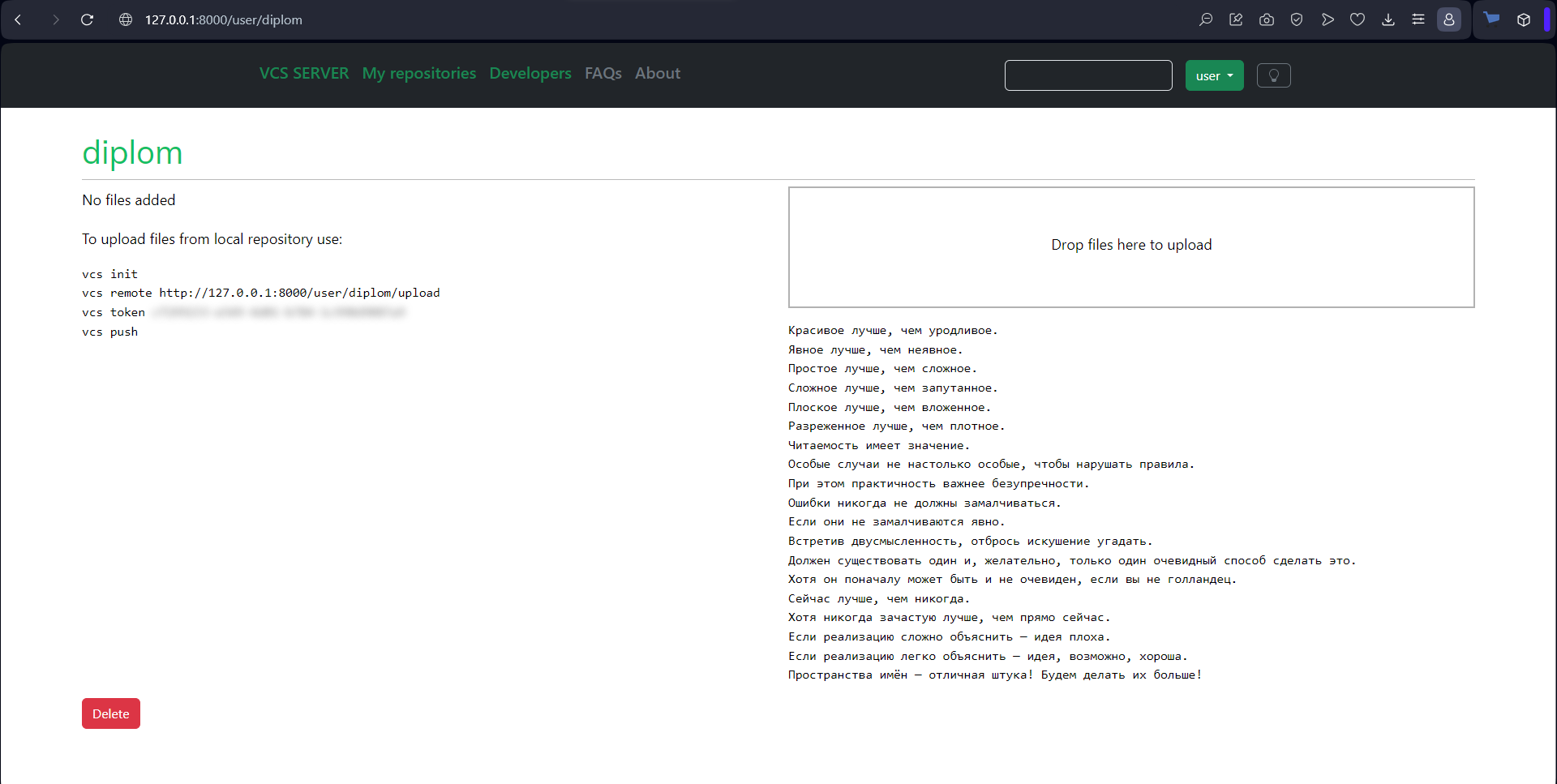


Рисунок 21. Страница репозитория без файлов

При создании репозитория, если не загрузить файлы на сервер, будет отображена страница на рисунке 21. На ней будут указаны команды, скопировав которые, можно отправить локальный репозиторий на удаленный сервер. Давайте разберем их:

vcs init – инициализирует репозиторий (если он еще не был создан)

vcs remote <hostname>/<username>/<repository\_name>/upload – указывает, куда загружать репозиторий (какому пользователю и в какой репозиторий). Эта информация сохраняется в config файл, после чего можно ей не пользоваться при отправке текущего локального репозитория в выбранный удаленный репозиторий.

vcs token <token> – сохраняет токен пользователя в config файл. Токен имя пользователя и имя репозитория будут использованы при отправке репозитория на удаленный сервер. Токен также можно увидеть на странице редактирования профиля (рисунок 9).

vcs push – отправляет файлы на сервер.

Скопируем данные команды в консоль, открыв предварительно папку своего проекта. Отправка файлов репозитория показана на рисунке 22.

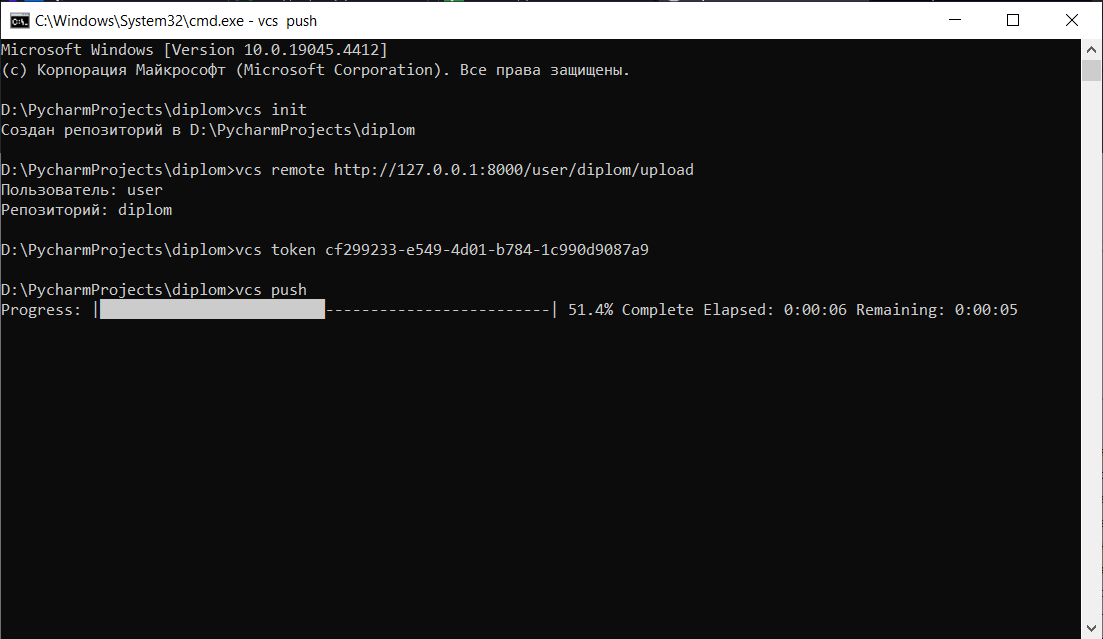


Рисунок 22. Отправка локального репозитория на сервер

После отправки репозитория возвращаемся на сервер и обновляем страницу репозитория. Как видим на рисунке 23, отображаются загруженные файлы и папки. Файлы отправляются по одному на сервер. Листинг кода отправки файлов на сервер в Приложении 5.

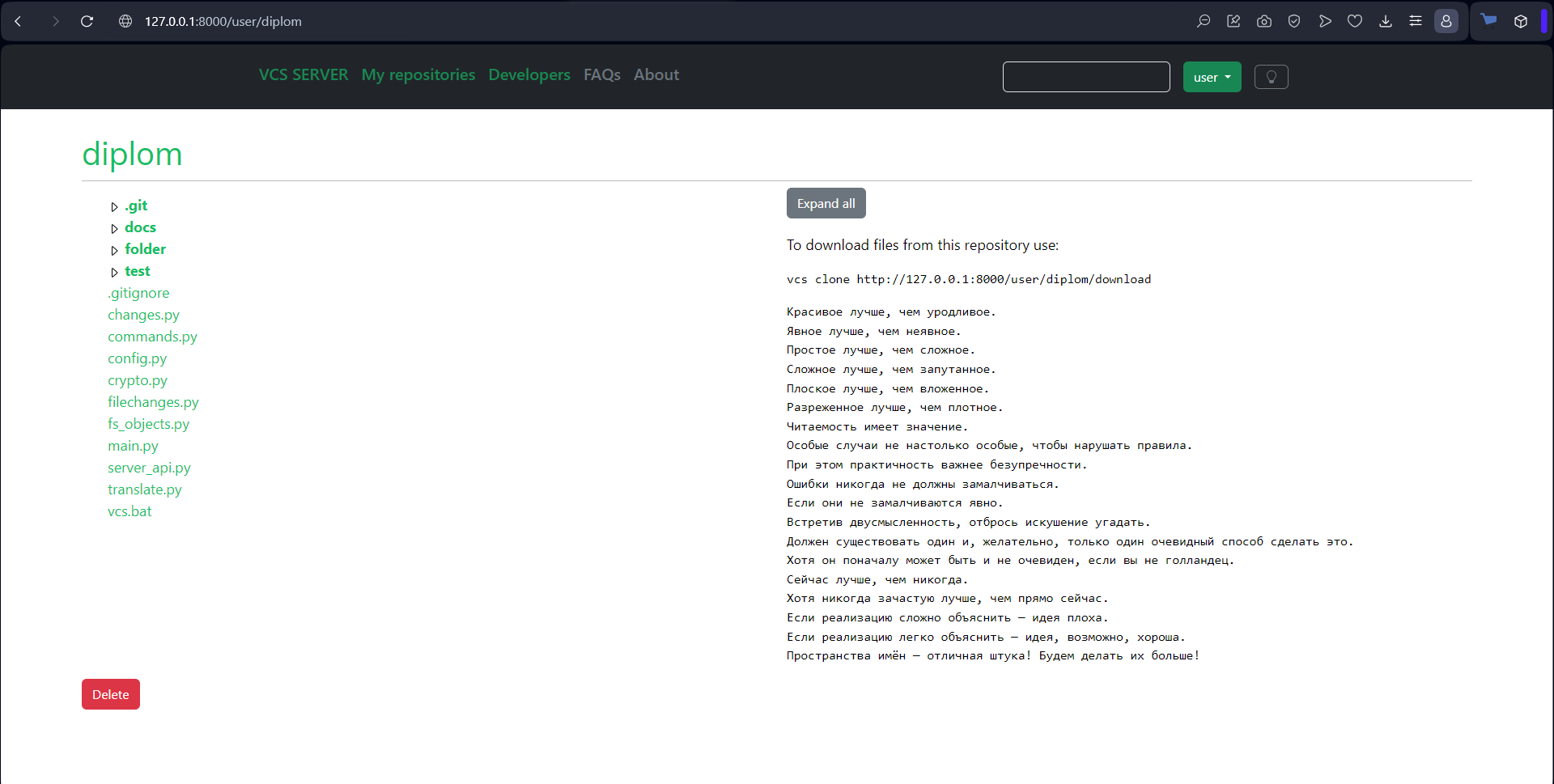


Рисунок 23. Страница репозитория

1.3.2. Загрузка удаленного репозитория с сервера на клиент

После отправки репозитория на сервер у нас или у другого пользователя может возникнуть необходимость загрузки удаленного репозитория к себе локально (клонирование). Как видим на рисунке 23, справа над описанием есть подсказка, как клонировать репозиторий с помощью команды vcs clone <hostname>/<username>/<repository\_name>/download. Открываем командную строку в том месте, куда нужно клонировать репозиторий, копируем и выполняем команду. Результат показан на рисунке 24. Файлы репозитория архивируются на сервере, и на клиент отправляется архив, который распаковывается. Листинг кода архивирования и отправки файлов сервером в Приложении 6.

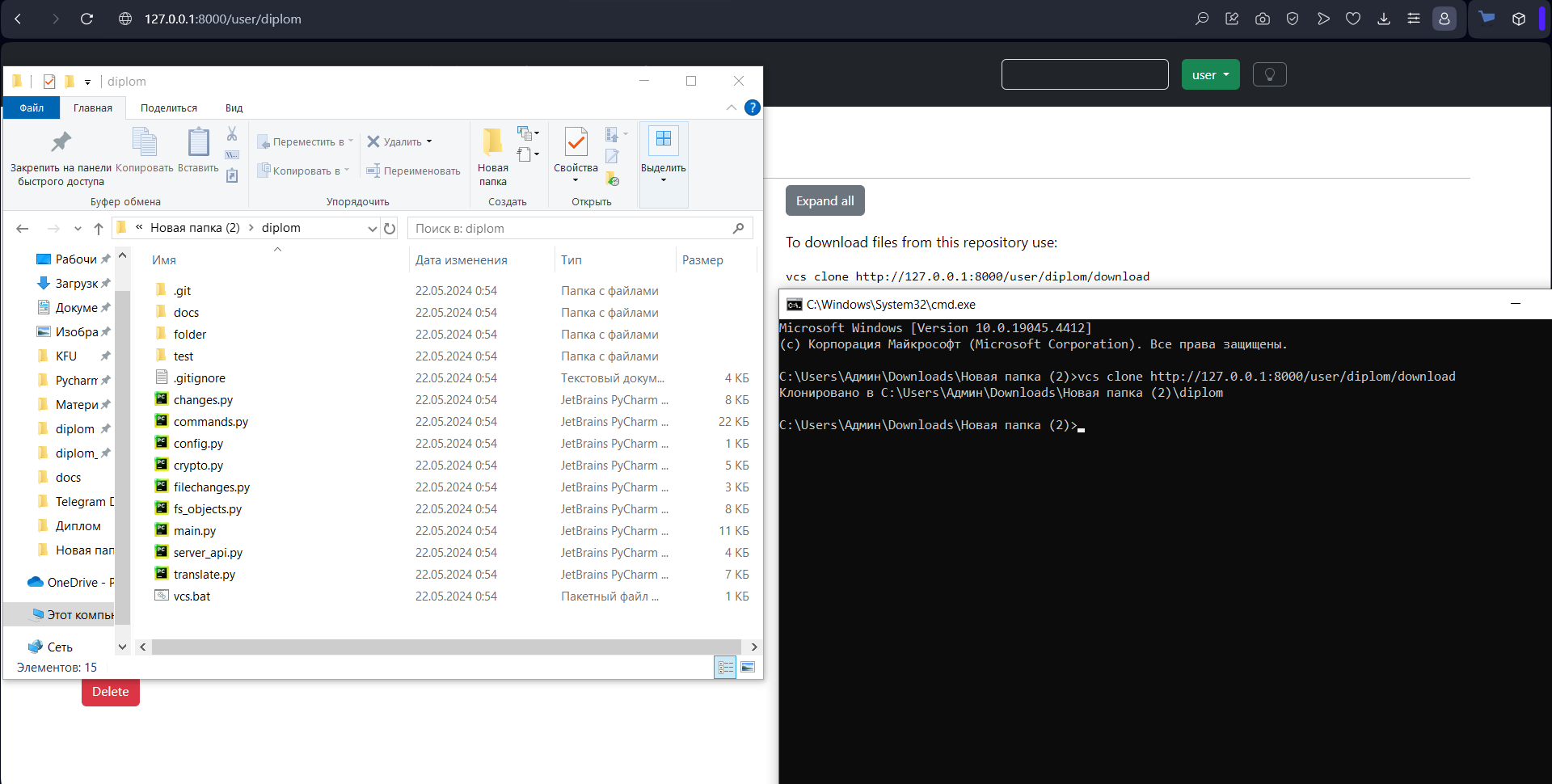


Рисунок 24. Загрузка удаленного репозитория на клиент

2. Тестирование приложения

2.1. Тестирование клиента

Было проведено модульное тестирование клиентской части с помощью фреймворка Pytest [9]. Проверены целостность файлов и папок после отката к предыдущим коммитам. Также я проверил правильность создания патчей после нескольких коммитов с изменениями. Были проверены функции для шифрования и дешифрования файлов и целостность данных после выполнения данных действий.

Часть тестов была произведена в ручном режиме, чтобы полностью охватить те кейсы, которые не были проверены при автоматизированном тестировании. Такой метод тестирования позволяет проверить функциональность и удобство использования программного продукта с точки зрения конечного пользователя.

При ручном тестировании я проверил следующие аспекты:

1. Установка и настройка клиента VCS
2. Создание нового репозитория
3. Создание нескольких репозиториев в разных каталогах
4. Удаление репозитория
5. Создание коммита
6. Игнорирование файлов
7. Вывод файловой структуры проекта
8. Просмотр истории коммитов
9. Откат к предыдущему коммиту
10. Шифрование/дешифрование репозитория
11. Отправка файлов репозитория на сервер
12. Загрузка файлов репозитория с сервера на клиент

2.2. Тестирование сервера

Тестирование серверной части было выполнено в ручном режиме с использованием Postman. Были проведены тесты каждой страницы в следующих популярных браузерах: Google Chrome, Opera, Mozilla Firefox, Microsoft Edge. Тесты показали, что веб-сервис достаточно адаптивен и хорошо выглядит во всех перечисленных браузерах.

Проведены следующие тесты:

1. Регистрация пользователя
2. Выход из аккаунта
3. Аутентификация, вход в аккаунт
4. Изменение пароля
5. Сброс пароля
6. Создание репозитория
7. Удаление репозитория
8. Загрузка файлов в репозиторий через браузер
9. Переключение тем на сайте
10. Отображение содержимого текстовых файлов и подсветка их синтаксиса
11. Отображение изображений у графических файлов
12. Вход через Яндекс OAuth

С помощью Postman я протестировал API, которое сервер предоставляет клиенту для отправки и загрузки файлов, чтобы убедиться, что сервер возвращает корректные ответы и статусы. Пример теста на рисунке 25.

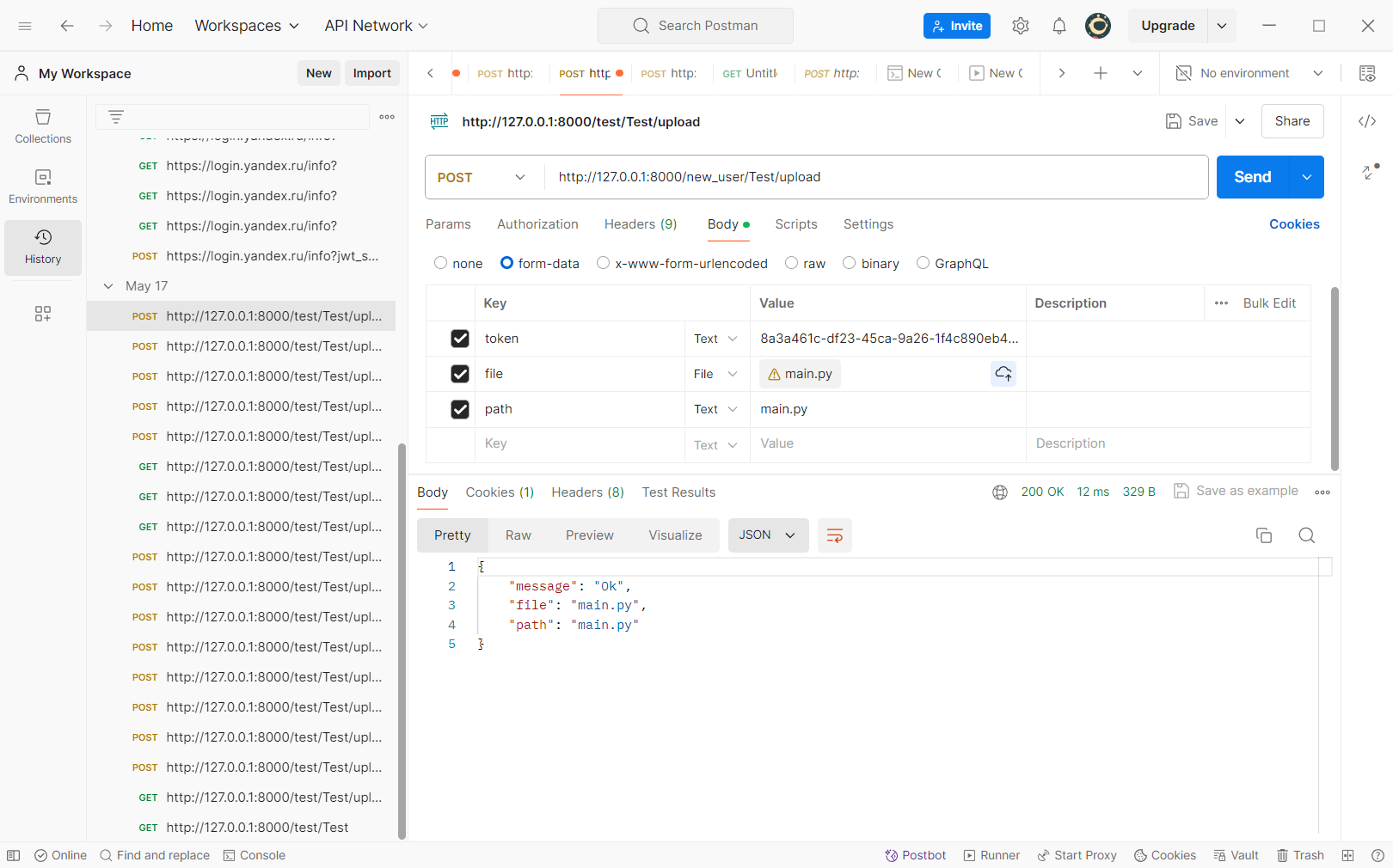


Рисунок 31. Тестирование API с помощью Postman

# Заключение

За время прохождения производственной технологической (проектно-технологической) практики мной приобретены следующие компетенции:

| **Шифр компетенции** | **Расшифровка приобретаемой компетенции** |
| --- | --- |
| ОПК-4 | Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью |
| ОПК-7 | Способен разрабатывать алгоритмы и программы, пригодные для практического применения |
| ОПК-8 | Способен принимать участие в управлении проектами создания информационных систем на стадиях жизненного цикла |

# Список использованных источников

1. Официальная документация Python. [Электронный ресурс] – URL: <https://docs.python.org/> (Дата обращения: 05.01.2024)
2. Официальная документация Django. [Электронный ресурс] – URL: <https://docs.djangoproject.com/> (Дата обращения: 10.03.2024)
3. Git - О системе контроля версий. [Электронный ресурс] – URL: <https://git-scm.com/book/ru/v2/> (Дата обращения: 12.01.2024)
4. Страуб Б., Чакон С. Pro Git. 2nd edition – Apress, 2014. – 440 с.
5. Руководство по SQLite. [Электронный ресурс] – URL: <https://metanit.com/sql/sqlite/> (Дата обращения: 05.04.2024)
6. Коржов В. Многоуровневые системы клиент-сервер – Открытые системы, 1997.
7. Web API с помощью Django REST framework. [Электронный ресурс] – URL: <https://habr.com/ru/articles/160117/> (Дата обращения: 19.02.2024)
8. Меле А. Django 4 в примерах. ДМК Пресс, 2023. – 800 с.
9. Full pytest documentation. [Электронный ресурс] – URL: <https://docs.pytest.org/en/latest/contents.html> (Дата обращения: 09.04.2024)

ПРИЛОЖЕНИЕ 1

Листинг кода классов, представляющих объекты файловой системы

**import** hashlib  
**import** os  
**import** pickle  
**from** pathlib **import** Path  
**from** typing **import** Union  
  
**from** config **import** DATA\_FOLDER  
  
**'''  
Классы описывающие объекты файловой системы  
'''  
  
def** get\_sha1\_hash(data: bytes) -> str:  
 *# Создаем объект хэша SHA-1* sha1\_hash = hashlib.sha1()  
 *# Обновляем объект хэша с данными* sha1\_hash.update(data)  
 *# Получаем вычисленный хэш в шестнадцатеричном формате* hex\_digest = sha1\_hash.hexdigest()  
 **return** hex\_digest  
  
  
**class** Blob:  
 *'''  
 Класс для хранения двоичных данных файлов  
 Блобов может быть меньше чем файлов, если в проекте есть файлы с одинаковым содержимым  
 В этом случае их хэш будет одинаков, и файл блоба будет общий  
 '''* **def** \_\_init\_\_(self, content):  
 self.content = content  
 self.hash = get\_sha1\_hash(content)  
  
 **def** save(self, path):  
 *'''  
 Сохраняем объект в бинарном файле  
 '''* **if** self.hash **in** os.listdir(DATA\_FOLDER):  
 *# такой объект уже есть в предыдущих коммитах, не сохраняем* **return** *# сохраняем объект* **with** open(os.path.join(path, self.hash), **'wb'**) **as** file:  
 pickle.dump(self, file)  
  
  
**class** File:  
 *'''  
 Класс для хранения данных о файле  
 '''* **def** \_\_init\_\_(self, name: Path):  
 self.name = name  
 **if** os.path.exists(name) **and** os.path.isfile(name):  
 **with** open(self.name, mode=**'rb'**) **as** file:  
 file\_content = file.read()  
 self.blob = Blob(file\_content)  
 self.name = self.name.name  
 self.hash = get\_sha1\_hash((str(self.name) + self.blob.hash).encode(**'utf-8'**))  
 **else**:  
 **raise** FileNotFoundError(**f"File '{**name**}' does not exist."**)  
  
 **def** save(self, path):  
 *'''  
 Сохраняем файл в бинарном файле, заменяя блоб его хэшем  
 Блоб сохраняем как отдельный файл  
 '''* **if** self.hash **in** os.listdir(DATA\_FOLDER):  
 *# такой объект уже есть в предыдущих коммитах, не сохраняем* **return** *# сохраняем объект* self.blob.save(path)  
 self.blob = self.blob.hash  
 **with** open(os.path.join(path, self.hash), **'wb'**) **as** file:  
 pickle.dump(self, file)  
  
 **def** load(self):  
 *'''  
 Загружаем блоб из файловой системы* **:return***:  
 '''* blob\_path = os.path.join(DATA\_FOLDER, self.blob)  
 **if not** os.path.exists(blob\_path):  
 **raise** Exception(**f'Нет такого элемента {**self.blob**}'**)  
 self.blob = load(blob\_path)  
  
  
**class** Tree:  
 **def** \_\_init\_\_(self, name: Path):  
 self.name = name  
 **if** os.path.exists(name) **and** os.path.isdir(name):  
 self.children = []  
 self.\_hash = **None** self.name = self.name.name  
 **else**:  
 **raise** FileNotFoundError(**f"Directory '{**name**}' does not exist."**)  
  
 **def** add\_child(self, child: Union[File, **'Tree'**]):  
 self.children.append(child)  
  
 @property  
 **def** hash(self):  
 data = str(self.name)  
 **for** child **in** self.children:  
 data += child.hash  
 **return** get\_sha1\_hash(data.encode(**'utf-8'**))

ПРИЛОЖЕНИЕ 2

Листинг кода получения списка изменений с предыдущим коммитом

**def** status(path=BASE\_PATH, old\_commit\_hash=**None**, new\_commit\_hash=**None**, gitignore=**True**):  
 *'''  
 Возвращает список кортежей изменений между old\_commit\_hash и new\_commit\_hash.  
 ('+', <filename>) добавление файла  
 ('-', <filename>) удаление файла  
 ('?', <filename>) изменение файла  
 ('?', <filename>, '>', > <new\_filename>) переименование файла* **:param** *path: путь по которому проверяется статус* **:param** *old\_commit\_hash: из какого коммита нужно сделать new\_commit\_hash* **:param** *new\_commit\_hash: коммит в который должны перейти  
 '''* changes\_list = []  
  
 **if** new\_commit\_hash **is None**: *# создаем новый коммит* new\_commit = make\_commit(path, print\_content=**False**, gitignore=gitignore)  
 **else**:  
 new\_commit = load(os.path.join(DATA\_FOLDER, new\_commit\_hash))  
 new\_commit.load()  
  
 **if** old\_commit\_hash **is None**: *# берем последний коммит* **if not** VCS\_FOLDER.exists():  
 print(**f"{**phrase[**'Репозиторий еще не создан'**][lang]**}"**)  
 **return** changes\_list  
 **elif not** HEAD\_PATH.exists():  
 print(**f"{**phrase[**'Коммитов еще не было'**][lang]**}"**)  
 **return** changes\_list  
 **else**:  
 **with** open(HEAD\_PATH, **'r'**) **as** file:  
 last\_commit\_hash = file.read()  
 prev\_commit = load(os.path.join(DATA\_FOLDER, last\_commit\_hash))  
 prev\_commit.load()  
 **else**:  
 *# prev\_commit = make\_commit(path, print\_content=False)* prev\_commit = load(os.path.join(DATA\_FOLDER, old\_commit\_hash))  
 prev\_commit.load()  
  
 *# сравниваем prev\_commit и new\_commit  
 # делаю у коммитов одинаковых родителей, чтобы если нет изменений, совпадали хэши* new\_commit = Commit(new\_commit.tree, prev\_commit.parent\_hash)  
 **if** new\_commit.hash == prev\_commit.hash:  
 print(**f"{**phrase[**'Изменений нет'**][lang]**}"**)  
 **return** changes\_list  
 **else**:  
 *# дерево проекта отличается* prev\_tree = prev\_commit.tree  
 new\_tree = new\_commit.tree  
 prev\_child\_hash = **''**.join([child.hash **for** child **in** prev\_tree.children])  
 new\_child\_hash = **''**.join([child.hash **for** child **in** new\_tree.children])  
 **if** prev\_child\_hash == new\_child\_hash:  
 changes\_list.append((**'?'**, prev\_tree.name, **'>'**, new\_tree.name))  
 **else**:  
 **if** prev\_tree.name != new\_tree.name:  
 changes\_list.append((**'?'**, prev\_tree.name, **'>'**, new\_tree.name))  
  
 **def** walk\_tree(new\_tree, prev\_tree, find\_deleted=**False**):  
 **for** new\_child **in** new\_tree.children:  
 **if** type(new\_child) == Tree:  
 new\_child\_hash = **''**.join([child.hash **for** child **in** new\_child.children])  
 **else**:  
 new\_child\_hash = new\_child.blob.hash  
  
 found = content\_changed = **False  
 for** prev\_child **in** prev\_tree.children:  
 **if** new\_child.name == prev\_child.name:  
 **if** new\_child.hash == prev\_child.hash:  
 found = **True** content\_changed = **False** *# удаляем найденный объект, чтобы он не использовался при поиске* prev\_tree.children.remove(prev\_child)  
 **else**:  
 found = **True** content\_changed = **True  
 break  
 else**:  
 **if** type(prev\_child) == Tree:  
 prev\_child\_hash = **''**.join([child.hash **for** child **in** prev\_child.children])  
 **else**:  
 prev\_child\_hash = prev\_child.blob.hash  
  
 **if** new\_child\_hash == prev\_child\_hash:  
 **if not** find\_deleted:  
 changes\_list.append((**'?'**, os.path.join(\*tree\_stack, prev\_child.name), **'>'**,  
 os.path.join(\*tree\_stack, new\_child.name)))  
 found = **True** content\_changed = **False** prev\_tree.children.remove(prev\_child)  
 **break** *# предыдущая версия была найдена* **if** found **and** content\_changed:  
 *# папка изменилась* **if** type(new\_child) == Tree:  
 tree\_stack.append(new\_child.name)  
 walk\_tree(new\_child, prev\_child, find\_deleted)  
 tree\_stack.pop()  
 prev\_tree.children.remove(prev\_child)  
 *# файл изменился* **else**:  
 **if not** find\_deleted:  
 changes\_list.append((**'?'**, os.path.join(\*tree\_stack, new\_child.name)))  
 prev\_tree.children.remove(prev\_child)  
 **elif not** found:  
 **if not** find\_deleted:  
 **if** type(new\_child) == Tree:  
 changes\_list.append((**'+'**, os.path.join(\*tree\_stack, new\_child.name)))  
 **else**:  
 changes\_list.append((**'+'**, os.path.join(\*tree\_stack, new\_child.name)))  
 **else**:  
 **if** type(new\_child) == Tree:  
 changes\_list.append((**'-'**, os.path.join(\*tree\_stack, new\_child.name)))  
 **else**:  
 changes\_list.append((**'-'**, os.path.join(\*tree\_stack, new\_child.name)))  
  
 tree\_stack = [new\_tree.name]  
 prev\_tree\_copy = copy.deepcopy(prev\_tree)  
 walk\_tree(new\_tree, prev\_tree\_copy)  
 walk\_tree(prev\_tree, new\_tree, find\_deleted=**True**)  
**return** changes\_list

ПРИЛОЖЕНИЕ 3

Листинг кода отката к предыдущему коммиту

**def** restore\_commit(commit\_hash, path=BASE\_PATH):  
 *'''  
 Восстанавливает файлы из коммита* **:param** *path: куда восстановить коммит  
 '''  
 # переходим на уровень вверх если корневая папка (иначе имя проекта будет дублироваться)* **if** path == BASE\_PATH:  
 path = path.parent  
 *# удаляем все файлы которых в коммите не было, и те которые были изменены* changes\_list = status(path, new\_commit\_hash=commit\_hash)  
 **for** change **in** changes\_list:  
 **if** change[0] **in '-?'**:  
 p = os.path.join(path, change[1])  
 **try**:  
 **if** os.path.isfile(p):  
 os.remove(p)  
 **elif** os.path.isdir(p):  
 shutil.rmtree(p)  
 **else**:  
 print(**f"{**phrase[**'Не найдено для удаления'**][lang]**}:"**, p)  
 **except** FileNotFoundError **as** e:  
 print(e)  
 **for** change **in** changes\_list:  
 print(\*change)  
  
 commit = load\_commit(commit\_hash)  
 **try**:  
 os.mkdir(os.path.join(path, commit.tree.name))  
 **except** FileExistsError **as** e:  
 **pass** *# добавляем папку проекта к пути* tree\_stack = [commit.tree.name]  
  
 *# проходимся по дереву коммита и восстанавливаем файлы и папки* **def** walk\_tree(tree, tree\_stack):  
 **for** child **in** tree.children:  
 **if** isinstance(child, File):  
 **with** open(os.path.join(path, \*tree\_stack, child.name), **'wb'**) **as** f:  
 f.write(child.blob.content)  
 **elif** isinstance(child, Tree):  
 **try**:  
 os.mkdir(os.path.join(path, \*tree\_stack, child.name))  
 **except** FileExistsError **as** e:  
 **pass** tree\_stack.append(child.name)  
 walk\_tree(child, tree\_stack)  
 tree\_stack.pop()  
  
 walk\_tree(commit.tree, tree\_stack)  
  
 *# обновляем хэш коммита в HEAD* **with** open(HEAD\_PATH, **'w'**) **as** file:  
 file.write(commit\_hash)

ПРИЛОЖЕНИЕ 4

Листинг кода формы для загрузки файлов через браузер

*<!-- Форма для загрузки файлов -->*<**link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.3/dropzone.min.css"**>  
<**script src="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.3/dropzone.min.js"**></**script**>  
  
<**form action="{% url 'file\_upload\_view' temp\_repository\_name %}" class="dropzone" id="my-dropzone" enctype="multipart/form-data"**>  
 {% csrf\_token %}  
 <**input type="text" id="full\_paths" name="full\_paths" hidden** />  
</**form**>  
  
<**script**>  
 Dropzone.options.myDropzone = {  
 paramName: "file",  
 init: function () {  
 var myDropzone = this;  
  
 // Обработка события "sending" для добавления информации о пути файла в FormData  
 this.on("sending", function(file, xhr, formData) {  
 var full\_paths = {};  
 if (file.webkitRelativePath) {  
 full\_paths[file.name] = file.webkitRelativePath;  
 } else {  
 full\_paths[file.name] = file.fullPath || file.name; // Обработать случаи без webkitRelativePath  
 }  
 console.log(full\_paths);  
 document.querySelector("#full\_paths").value = JSON.stringify(full\_paths);  
 });  
 }  
 };  
</**script**>

ПРИЛОЖЕНИЕ 5

Листинг кода отправки файлов на сервер

**import** datetime  
**import** os.path  
**import** sys  
  
**import** requests  
**import** zipfile  
**import** io  
**import** os  
  
**import** commands  
**from** config **import** BASE\_PATH  
**from** translate **import** phrase

**def** send\_file(filepath, url, token):  
 *# Открытие файла в режиме чтения двоичных данных* **with** open(filepath, **'rb'**) **as** f:  
 files = {**'file'**: f}  
 *# data = {'token': token, 'path': os.path.relpath(filepath, BASE\_PATH.parent)}* data = {**'token'**: token, **'path'**: os.path.relpath(filepath, BASE\_PATH)}  
 *# Отправка POST-запроса с файлом и данными формы* response = requests.post(url, files=files, data=data)  
  
 **if** response.status\_code != 200:  
 **raise** Exception(**f'Failed: {**response.status\_code**} {**response.text**}'**)  
  
  
**def** send\_files(url, token, path=BASE\_PATH):  
 tree = commands.iter\_folder(path=BASE\_PATH, print\_content=**False**)  
 filenames = list(tree.get\_filenames())  
 total\_tasks = len(filenames)  
 start\_time = datetime.datetime.now()  
 **for** i **in** range(total\_tasks):  
 full\_path = os.path.join(BASE\_PATH.parent, filenames[i])  
 **try**:  
 send\_file(full\_path, url, token)  
 **except** Exception **as** e:  
 print(e)  
 **return** print\_progress\_bar(i + 1, total\_tasks, start\_time, prefix=**'Progress:'**, suffix=**'Complete'**, length=50)

ПРИЛОЖЕНИЕ 6

Листинг кода архивирования и отправки файлов сервером

@csrf\_exempt  
**def** api\_download(request, username, repository\_name):  
 **if** request.method == **'GET'**:  
 user = get\_object\_or\_404(User, username=username)  
 repository = get\_object\_or\_404(Repository, user=user, name=repository\_name)  
 *# token = request.GET.get('token')  
 # if user.profile.token != token:  
 # return JsonResponse({'message': 'Invalid token'}, status=401)* full\_path = os.path.join(settings.MEDIA\_ROOT, **'files'**, username, repository\_name)  
 **if not** os.path.exists(full\_path):  
 **return** JsonResponse({**'message'**: **'Folder not exists'**}, status=404)  
  
 *# Create a zip file in memory* zip\_buffer = io.BytesIO()  
 **with** zipfile.ZipFile(zip\_buffer, **'w'**, zipfile.ZIP\_DEFLATED) **as** zip\_file:  
 **for** root, dirs, files **in** os.walk(full\_path):  
 **for** file **in** files:  
 file\_path = os.path.join(root, file)  
 arcname = os.path.relpath(file\_path, full\_path)  
 zip\_file.write(file\_path, arcname)  
 zip\_buffer.seek(0)  
  
 response = HttpResponse(zip\_buffer, content\_type=**'application/zip'**)  
 response[**'Content-Disposition'**] = **f'attachment; filename={**os.path.basename(full\_path)**}.zip'  
 return** response  
 **return** JsonResponse({**'message'**: **'Method not allowed'**}, status=405)