

Федеральное государственное бюджетное
образовательное учреждение
высшего образования
Пензенский Государственный университет
Кафедра вычислительной техники

ДИССЕРТАЦИЯ

на соискание степени магистра техники и технологии на тему:

Многопользовательская клиент-серверная Web-система для управления распределенными вычислениями

По направлению 09.04.01 "Информатика и вычислительная техника"

Автор работы _____ А.А. Рзянкина

(подпись, дата)

Группа 15ВВм1

Руководитель работы _____ к.т.н., доцент А. Н. Токарев

(подпись, дата)

Нормоконтроль _____ к.т.н., доцент А.В. Кучин

(подпись, дата)

Работа допущена к защите

Заведующий кафедрой _____ д.т.н., профессор Д.В. Пащенко

(подпись, дата)

Оценка _____ Протокол ГАК № _____ от «__» _____ 2017 г.

Председатель ГАК _____

(подпись, дата)

Реферат

Магистерская диссертация 111 листов, 34 рисунка, 1 таблица, 9 источников, 1 приложение.

КЛАСТЕР, ПРОТОКОЛ, САЙТ, ХЭШ, АЛГОРИТМ, ДЕРЕВО, ПЕРЕДАЧА ДАННЫХ, ЗАДАЧА, ПОДЗАДАЧА, РАСПРЕДЕЛЕННАЯ ОБРАБОТКА

Цель работы — создание многопользовательской клиент-серверной Web-системы для управления распределенными вычислениями. В качестве задачи был выбран подбор пароля по ХЭШ значению.

Объект разработки — многопользовательская клиент-серверная WEB-система, выполняющая распределение подзадач между клиентами в зависимости от их вычислительных мощностей.

Предмет исследований — оптимизация разделения подзадач поиска пароля по ХЭШ значению и организация удаленного доступа к системе управления заданиями.

В результате проведенных исследований достигнуто следующее:

- создан web-портал для удаленного управления системой распределенных вычислений
- разработан алгоритм разбиения задач на подзадачи
- разработан алгоритм распределения подзадач между клиентами

Проведен анализ предметной области, с использованием алгоритмов разбиения задач на подзадачи, распределения подзадач для обработки, а также методов хеширования.

Практическая ценность — разработанная программа может использоваться для восстановления пароля пользователя, если нерационально или невозможно изменить его административными методами. Для восстановления данных, зашифрованных несимметричными криптографическими алгоритмами. При изменении расчетного алгоритма программа может быть исполь-

зована для решения широкого спектра поисково-переборных задач в гетерогенных распределенных вычислительных системах.

Эффективность разработки заключается в возможности повысить скорость поиска пароля по сравнению с перебором на одном компьютере или на нескольких.

Реализация и внедрение: сайт был размещен в сети Интернет, было осуществлено бета-тестирование системы в рабочем режиме с несколькими удаленными клиентами.

Содержание

Введение.....	6
1 Анализ поставленной задачи и методов ее решения.....	8
1.1 Анализ предметной области	8
1.1.1 Постановка задачи	8
1.1.2 Обзор существующих решений систем распределенных вычислений	8
1.1.3 Обзор существующих решений подбора пароля.....	13
1.1.4 Необходимость разработки.....	14
1.1.5 Требования к разработке	16
1.2 Технология разработки	17
1.2.1 Выбор средств разработки	17
1.2.2 Выбор языка разработки	25
1.2.3 Выбор инструментов разработки	29
1.3 Выбор стратегии разбиения задачи	32
1.3.1 Перебор вариантов. Построение дерева вариантов.....	32
1.3.2 Организация обхода деревьев.....	34
1.4 Выбор методологии разработки	37
1.4.1 Обзор основных методологий разработки ПО	37
1.4.2 Выбор пути разработки проекта.....	39
2 Реализация многопользовательской клиент-серверной Web-системы для управления распределенными вычислениями	40
2.1 Способ распределения подзадач	40
2.1.1 Определение размера статистической подзадачи	40
2.1.2 Коэффициент производительности клиента	43
2.2 Описание разработанного программного обеспечения	44
2.2.1 Сервер.....	45
2.2.2 Клиент	59
2.2.3 Взаимодействие между сервером и клиентами	66

2.2.4	Состояния работы программы.....	67
3	Результаты работы	69
3.1	Тестирование	69
3.2	Руководство пользователя	70
	Заключение	71
	Список использованных источников	72
	Приложение А Исходный код программы	73

Введение

Имеется большое количество важнейших задач, решение которых требует использования огромных вычислительных мощностей, зачастую недоступных для современных вычислительных систем. К таким задачам, прежде всего, относятся задачи точных долгосрочных прогнозов климатических изменений и геологических катаклизмов (землетрясений, извержений вулканов, столкновений тектонических плит), прогнозов цунами и ураганов, а также экологических прогнозов и т.п. Сюда следует отнести также прогнозирование результатов экспериментов во многих разделах физики, в особенности, экспериментов по выявлению основ мироздания (экспериментов на коллайдерах со встречными пучками частиц, экспериментов, направленных на получение антиматерии и, так называемой, темной материи и т.д.). Важными задачами являются расшифровка генома человека, определение роли каждого гена в организме, влияние генов на здоровье человека и на продолжительность жизни. В приоритете задача безопасного хранения вооружений, в особенности, ядерного оружия (из-за запрета на ядерные испытания состояние накопленных ядерных зарядов можно определить лишь путем моделирования на компьютере большой мощности).

Постоянно появляются новые задачи подобного рода и, возрастают требования к точности и к скорости решения прежних задач; поэтому вопросы разработки и использования сверхмощных компьютеров (называемых суперкомпьютерами) актуальны сейчас и в будущем. К сожалению, технологические возможности увеличения быстродействия процессоров ограничены: увеличение быстродействия связано с уменьшением размеров процессоров, а при малых размерах появляются трудности из-за квантово-механических эффектов, вносящих элементы недетерминированности; эти трудности пока что не удается преодолеть. Из-за этого приходится идти по пути создания параллельных вычислительных систем, т.е. систем, в которых предусмотрена одновременная реализация ряда вычислительных процессов, связанных с реше-

нием одной задачи. На современном этапе развития вычислительной техники такой способ, по-видимому, является одним из основных способов ускорения вычислений.

Первоначально идея распараллеливания вычислительного процесса возникла в связи с необходимостью ускорить вычисления для решения сложных задач при использовании имеющейся элементной базы. Предполагалось, что вычислительные модули (процессоры или компьютеры) можно соединить между собой так, чтобы решение задач на полученной вычислительной системе ускорялось во столько раз, сколько использовано в ней вычислительных модулей. Однако, достаточно быстро стало ясно, что для интересных сложных задач достичь упомянутого ускорения, как правило, невозможно по двум причинам:

1) любая задача распараллеливается лишь частично (при полном распараллеливании параллельные части не могут взаимодействовать в процессе счета и представляют собой отдельные задачи меньшего размера, так что использование параллельной системы теряет смысл) — закон Амдала,

2) коммуникационная среда, связывающая отдельные части параллельной системы, работает значительно медленнее процессоров, так что передача информации существенно задерживает вычисления.

Параллельное программирование невозможно без представления о методах решения возникающих задач, об архитектуре параллельных вычислительных систем и о математическом обеспечении, которое имеют эти системы. Важно знать класс методов, удобных для реализации на параллельной системе, и алгоритмическую структуру этих методов, а также изучить средства параллельного программирования.

1 Анализ поставленной задачи и методов ее решения

1.1 Анализ предметной области

1.1.1 Постановка задачи

Задача данной работы — создание клиент-серверной WEB-системы, обеспечивающей удобный доступ к вычислительным ресурсам, выполняющей разбиение основной задачи на подзадачи, анализ работоспособности клиентов и назначение клиенту подзадачи в соответствии с мощностью его компьютера.

Клиентом является любой зарегистрированный в системе пользователь, установивший на компьютер службу-клиент. Задачей в контексте данной работы является подбор пароля (в рамках заданной пользователем максимальной длины) по ХЭШ-значению. Подзадачей является часть вариантов перебора, ограниченная областью алфавита и возможной длиной пароля.

Служба-клиент принимает от сайта-сервера подзадачи по HTTP протоколу и отправляет результат вычислений.

1.1.2 Обзор существующих решений систем распределенных вычислений

Распределённые вычисления — способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров, чаще всего объединённых в параллельную вычислительную систему.

Особенностью распределенных многопроцессорных вычислительных систем, в отличие от локальных суперкомпьютеров, является возможность неограниченного наращивания производительности за счет масштабирования.

На данный момент существует несколько готовых решений для удобного распараллеливания, как низкоуровневых, так и высокоуровневых.

1.1.2.1 Низкоуровневые средства: MPI, OpenMp

Message Passing Interface (MPI, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет процессам, выполняющим одну задачу, обмениваться сообщениями. MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран. Дает описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор подчиненных (slave) потоков и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков).

1.1.2.2 GRID-вычисления

GRID — это совокупность нескольких компьютеров для решения единой вычислительно сложной задачи, разбитой на подзадачи. Каждый компьютер решает несколько подзадач, после чего результаты отдельных вычисле-

ний объединяются. Главное преимущество GRID в том, что она может состоять из компьютеров, удаленных друг от друга на сотни и тысячи километров, имеющих различные аппаратные и программные характеристики. Задачу объединения компьютеров выполняет промежуточное программное обеспечение, которое (виртуально) связывает все компьютеры через Интернет в единый суперкомпьютер. Идея GRID возникла в 1990-х годах, когда с развитием средств компьютерных коммуникаций объединение географически удаленных друг от друга компьютеров стало более дешевым, простым и потенциально более мощным средством повышения производительности, чем наращивание мощности одного-единственного суперкомпьютера.

Существуют различные виды GRID. Volunteer Computing — это форма реализации GRID-вычислений, особенность которой заключается в использовании времени простоя компьютеров обычных пользователей по всему миру. На текущий момент самым крупным проектом Volunteer Computing как по числу участников, так и по суммарной мощности является Folding@home — проект по проведению компьютерной симуляции свертывания молекул белка, запущенный в октябре 2000 года учеными из Стэнфордского университета. Однако речь в этой статье пойдет не о Folding@home, а о втором по величине проекте — системе BOINC. Причина проста — в отличие от специализированного Folding@home, проект BOINC предоставляет возможность участия в самых разнообразных научных проектах, начиная от взлома криптографических систем и заканчивая поиском внеземных цивилизаций.

Для организации распределенных вычислений (GRID-вычислений) необходима соответствующая программная платформа. Система должна уметь разбивать одну большую задачу на множество маленьких подзадач, распределять эти подзадачи по вычислительным узлам, принимать результаты вычислений и объединять их в единое целое. Для этого были созданы различные программные «прослойки» между управляющим сервером и вычислительными узлами. Одной из таких программных «прослоек» является BOINC.

BOINC (Berkeley Open Infrastructure for Network Computing – открытая платформа Университета Беркли для сетевых вычислений) – это свободная (распространяется под лицензией GNU LGPL) программная платформа для проведения распределенных вычислений (если точнее, то Volunteer Computing). Система BOINC была разработана в Калифорнийском университете в Беркли под руководством Дэвида Андерсона (David Anderson) командой, создавшей легендарный проект SETI@home. Основным мотивом разработки системы послужила нехватка свободных вычислительных мощностей для обработки данных, поступающих от радиотелескопов. Именно поэтому разработчики решили привлечь вычислительные ресурсы и объединить сообщества нескольких научных проектов. Для решения этой масштабной задачи и была создана программная платформа BOINC.

Система BOINC состоит из программы-клиента, общей для всех BOINC-проектов, составного сервера (термин «составной сервер» означает, что физически сервер может состоять из нескольких отдельных компьютеров) и программного обеспечения. Для выполнения распределенных вычислений используется архитектура клиент-сервер.

Пользователи, передающие вычислительные мощности BOINC-проектам типа SETI@home, «зарабатывают» кредиты, начисляемые за выполнение отдельных подзадач. Эти кредиты нужны только для создания духа соревновательности среди участников проекта – пользователи, имеющие наибольшее число кредитов, имеют шанс «засветиться» на главной странице проекта.

Особенность «volunteer computing» заключается в том, что для успешного решения отдельные небольшие подзадачи должны быть очень слабо связаны между собой и практически не зависеть от результатов параллельно выполняемых заданий. В противном случае очень большие издержки производительности будут приходиться на ожидание других результатов и на их синхронизацию. Идеальной задачей является, например, подбор пароля ме-

тодом «грубой силы» – для каждого варианта (десятка, сотни – в зависимости от сложности вычисления) пароля отдельный компьютер вычисляет хэш и сравнивает его с заданным. Первое совпадение – задача решена, но никакой другой результат не может приблизить к правильному ответу, а подзадачи абсолютно независимы (в силу свойств парольных хэш-функций). Под такие задачи, состоящие из независимых подзадач, и разработана архитектура системы BOINC.

На рисунке ниже представлена архитектура системы BOINC.

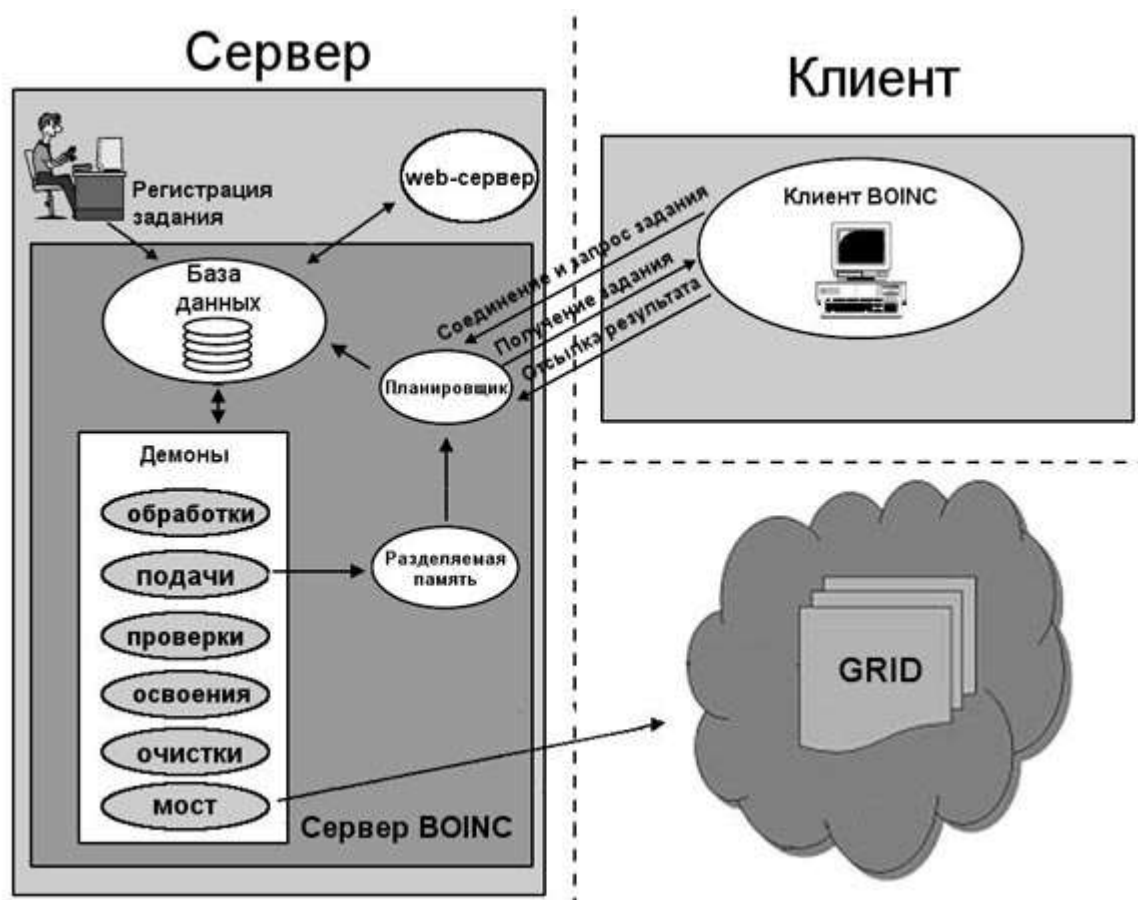


Рисунок 1 — Архитектура системы BOINC

В основу архитектуры BOINC положена идея конечного автомата – сервер состоит из набора отдельных подсистем, каждая из которых отвечает за свою вполне определенную задачу, например, выполнение вычислений, передачу файлов и т.д. Каждая из подсистем проверяет состояние подзадачи,

производит какие-то действия и изменяет состояние подзадачи – так они работают в бесконечном цикле.

В целом, система состоит из сервера BOINC (при необходимости, распределенного на несколько физических серверов в целях повышения производительности, отказоустойчивости и безопасности), множества клиентов, выполняющих задания сервера и, возможно, дополнительных компонент в виде присоединенных GRID-сетей (например, на основе широко распространенного инструментария Globus Toolkit).

Globus Toolkit – это открытый (Open Source) инструментарий для создания вычислительных Grid. Включает в себя набор программных сервисов и библиотек для проведения мониторинга ресурсов, обнаружения и управления вычислительными узлами, обеспечения безопасности и управления файлами. Разрабатывается и поддерживается организацией Globus Alliance.

1.1.3 Обзор существующих решений подбора пароля

Раньше большинство паролей хранилось в открытом виде, но очень быстро разработчики поняли, что так делать небезопасно. Лучше хранить не сам пароль, а его хэш, набор цифр определенного размера, который генерируется на основе пароля. В таком случае, даже если злоумышленник получит хэш, то он не сможет узнать пароль. Существуют различные способы хэширования, например, md5, sha1, sha2 и многие другие. Но существует способ восстановления пароля из хэша. Для этого можно использовать перебор, нам достаточно создавать хэш для каждого возможного пароля и сравнивать его с тем хэшем, который нужно расшифровать.

Для перебора хешей существуют различные программы, самая популярная из них - это Hashcat. С помощью этой утилиты можно перебирать значение хэша по словарю или полным перебором по всем значениям.

Программа hashcat поддерживает работу с такими алгоритмами хэширования: md5, md5crypt, sha1, sha2, sha256, md4, mysql, sha512, wpa, wpa2, grub2, android, sha256crypt, drupal7, scrypt, django и другими.

Минус Hashcat — сложность настройки и необходимость запоминать больше количество ключей командной строки.

1.1.4 Необходимость разработки

1.1.4.1 Необходимость использования сайта

Представление в виде сайта намного упрощает пользователю работу с распределенными вычислениями — он может приступить к работе в любом месте, где есть доступ к Интернету. Подобное представление избавляет пользователя от необходимости изучать сложную документацию на средства распараллеливания, запоминать ключи командной строки. Кроме регистрации и установки службы от пользователя ничего не требуется.

Регистрация клиентов позволяет вести статистику добавления задач, статистику участия пользователей в решении задач. Предоставление пользователю актуальной информации о количестве клиентов on-line, готовых к работе над задачами, предоставление списка решенных задач, которые создал конкретный пользователь, упрощает работу с порталом.

Возможность обратной связи позволяет разработчику своевременно реагировать на замечания клиентов.

1.1.4.2 Необходимость поиска пароля по ХЭШ значению

Разработанная программа полезна администраторам в больших и малых компаниях в ситуациях, когда пользователь забыл свой пароль. Обычно в БД хранится ХЭШ, а обратных алгоритмов преобразования из хэша в пароль нет. Если пользователь использует один пароль для нескольких ресурсов, легче восстановить его путем подбора, чем оставлять заявки на изменение.

Также программа может помочь, если компьютер был заражен вирусом-шифровальщиком. Вирус блокирует файлы или папки паролем и требует деньги за доступ к нему. Программа позволяет определить методом перебора закрытый ключ, который позволит расшифровать файлы.

Достоинством программы является применение современного и часто используемого алгоритма шифрования. Но если он не подходит, в программе его можно легко заменить.

1.1.4.3 Необходимость использования базы данных

Для работы с большим количеством данных люди используют либо файловое хранение, либо базы данных.

При работе с файлами необходимо постоянно держать под контролем много вспомогательных параметров и файлов. Усложняется поиск по сайту и создание архива.

В противоположность файловой организации хранения информации, использование баз данных дает неоспоримые преимущества. Например, легко организовывать поиск, сортировать записи по дате и времени, проводить различные отборы записей, так как в БД предусмотрена эффективная организация хранения информации, минимизирующая время доступа и поиска. Можно быстро найти среди многих тысяч записей одну определенную (по заданному идентификатору).

Базы данных лишены большого недостатка файлов: с ними нет проблем с совместным доступом к данным. Сценарий, который изменяет файл в процессе своей работы, могут запустить одновременно два человека, и, если не принять мер по блокировке файла, то могут возникнуть проблемы. С базами данных таких проблем не существует, потому что предусмотрено решение проблем совместного доступа на низком уровне с максимальной эффективностью.

1.1.5 Требования к разработке

Требуется разработать клиент-серверное приложение для реализации распределенного алгоритма поиска пароля по хэш-значению.

Операционная система – Windows 7/8/10.

Протокол взаимодействия клиента/сервера – HTTP.

Языки разработки – Python, C#.

Среда разработки – PyCharm, Microsoft Visual Studio.

Исходные данные: зарегистрированные пользователи с указанием IP-адреса компьютера, технических возможностей процессора и видеокарты, хэш-значение пароля, алфавит символов, содержащихся в пароле.

Результат работы программы: найденное значение пароля либо сообщение, что пароль не найден, статистические данные о производительности клиентов.

1.2 Технология разработки

1.2.1 Выбор средств разработки

1.2.1.1 Платформа Django

Django — свободно-распространяемый высокоуровневый инструмент для разработки web-приложений. Django написан на языке Python на основе архитектуры MVC. Он содержит все необходимые компоненты для разработки web-сайтов: интерфейсы аутентификации пользователей (вход, выход, регистрация), панель управления сайтом, формы, инструменты для загрузки файлов и т. д.

Приложение на основе Django состоит из четырех основных компонентов.

1) Модель данных. Модель — важнейшая часть приложения, которое постоянно обращается к данным при любом запросе из любой сессии. Любая модель является стандартным классом. Таким классам обеспечивается непосредственный доступ к базам данных, и программисту нет необходимости писать запросы на SQL. Модель не зависит от конкретного приложения. Данными можно манипулировать даже из командной строки, не используя при этом Web-сервер.

2) Представления (view) в Django выполняют разнообразные функции, в том числе контролируют запросы пользователя, выдают контекст в зависимости от его роли. View — это обычная функция, которая вызывается в ответ на запрос какого-то адреса (url) и возвращает контекст.

3) Шаблоны являются формой представления данных. Шаблоны имеют свой собственный простой мета-язык и являются одним из основных средств вывода на экран.

4) URL — это механизм внешнего доступа к представлениям (view). Встроенные в url регулярные выражения делают механизм достаточно гиб-

ким. При этом одно представление может быть сконфигурировано к нескольким url, предоставляя доступ различным приложениям.

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google и др.

1.2.1.2 Запросы HTTP-протокола

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста». В соответствии со спецификацией OSI, HTTP является протоколом прикладного (7-ого) уровня.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом. Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

Также HTTP часто используется как протокол передачи информации для других протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. В таком случае говорят, что протокол HTTP используется как «транспорт».

Сейчас наиболее часто используются всего два HTTP метода: GET и POST. Но даже из двух вариантов веб-разработчики не всегда выбирают подходящий, поскольку оба метода можно использовать для получения одинакового результата. Необдуманное применение какого-либо из методов может повлечь за собой большие нагрузки на канал или привести к проблемам безопасности.

В программировании POST — один из многих методов запроса, поддерживаемых HTTP протоколом, используемым во Всемирной паутине. Метод запроса POST предназначен для запроса, при котором веб-сервер принимает данные, заключённые в тело сообщения. В рамках POST запроса произвольное количество данных любого типа может быть отправлено на сервер в теле сообщения запроса.

Характеристики метода POST:

- предназначен для отправления данных на сервер;
- передача данных происходит в теле запроса;
- обработка на стороне сервера медленнее и «тяжелее», чем GET, потому что помимо заголовков нужно анализировать тело запроса;
- способен передать большие объёмы данных;
- способен передать файлы;
- страницу, сгенерированную методом POST, нельзя сохранить в закладки;
- разрывает HTTP-соединение;
- для передачи даже очень малого объема информации большинством браузеров отправляет минимум два TCP-пакета: заголовок, а потом тело запроса.
- если по логике происходит небезопасный запрос — используется POST;
- если требуется передать большой объем данных или файл — используется POST;

- в остальных случаях используется GET.

GET не должен использоваться для конфиденциальной информации, такой как имена пользователей и пароли, которые должны быть представлены вместе с другими данными для завершения запроса. Даже при использовании HTTPS, предотвращающим данные от перехвата при передаче, истории браузера и журналы веб-сервера, вероятно, содержат полную информацию об открываемых ранее ссылках в виде открытого текста, которые могут быть найдены, если система будет взломана. В этих случаях используется HTTP POST.

Характеристики метода GET:

- предназначен для получения данных с сервера;
- тело запроса пустое;
- обрабатываются на стороне сервера быстрее и с меньшим потреблением ресурсов сервера за счет пустого тела запроса;
- передача переменных происходит в адресной строке (так видит пользователь, технически данные предаются в строке запроса) и поэтому видна информация о переменных и их значениях (данные не защищены);
- способен передать небольшое количество данных на сервер: есть ограничения на длину URL, которое зависит от браузера, например, IE6 = 2Kb.
- может передать только ASCII символы;
- такой запрос можно скопировать, сохранить (например, в закладках);
- запрос может кэшироваться (этим можно управлять);
- для дополнительного снижения нагрузки на канал и сервер доступны условные и частичные запросы;

- не разрывает HTTP соединение (при включенном на сервере режиме keepAlive).

Поскольку данные обладают определенной степенью секретности, был отдан приоритет безопасному запросу POST.

1.2.1.3 База данных SQLite

Для управления данными существует множество решений, самые распространенные из них — это СУБД SQLite, MySQL, PostgreSQL.

SQLite — компактная, встраиваемая реляционная система управления базой данных. Вся база данных состоит из одного файла, поэтому её удобно перемещать между машинами. SQLite удобна при разработке и тестировании. Если в процессе создания приложений появляется необходимость масштабирования, база данных предлагает всё что необходимо для этих целей, так как состоит всего из одного файла и библиотеки написанной на языке C. К недостаткам можно отнести отсутствие системы управления правами доступа пользователей, которую предоставляют более крупные СУБД. Обычно применения этой функции не так критично, так как эта СУБД используется в небольших приложениях.

MySQL — СУБД с широкими функциональными возможностями и высокой производительностью. MySQL проста в использовании, безопасна, позволяет работать с большими объемами данных. Недостаток заключается в достаточно медленной разработке и некотором ограничении функционала, который иногда необходим в особо требовательных приложениях.

PostgreSQL — мощная СУБД с открытым исходным кодом. Существует большое количество тематических форумов, на которых можно найти ответы на вопросы, возникающие при использовании СУБД. PostgreSQL дает возможность расширения функционала за счет сохранения своих процедур. Значительным недостатком является низкая производительность. При про-

стных операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL.

После анализа преимуществ и недостатков баз данных, была выбрана SQLite. SQLite — это встраиваемая кроссплатформенная СУБД, которая поддерживает достаточно полный набор команд SQL, легко встраивается в приложения. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с SQLite обращения происходят напрямую к файлам с данными без использования портов и сокетов, как в сетевых СУБД. Именно поэтому SQLite очень быстрая, а также мощная благодаря технологиям обслуживающих библиотек.

1.2.1.4 Технология WIX

Технология WIX представляет собой набор инструментов и спецификаций упрощающих процесс создания дистрибутивов на базе MSI (Microsoft Installer). Главное преимущество WIX перед конкурентами (InstallShield, InstallAware, InnoSetup, NSIS) — возможность бесплатного использования.

Структура базы данных Windows Installer достаточно сложна, поэтому непосредственная работа с ней неудобна и непродуктивна. В то же время Windows Installer XML позволяет создавать установочные пакеты без непосредственного заполнения таблиц, а его компилятор и компоновщик при работе проверяют выполнение ряда условий, в том числе наличие и целостность ссылок между элементами. Использование WiX основывается на редактировании XML файла (или нескольких файлов) определенной структуры. Windows Installer XML позволяет создавать программы установки в любом текстовом редакторе. Однако извлечь максимум преимуществ от использова-

ния пакета позволяет его интеграция в Visual Studio, упрощающая и ускоряющая процесс разработки.

1.2.1.5 Интерфейс CryptoAPI

В качестве решаемой задачи был выбран подбор пароля по ХЭШ значению. Поскольку пароль должен выступать надежным хранителем данных, было решено использовать современные средства шифрования.

В дипломном проекте используются функции вычисления хэш-значений на основе 128-битного алгоритма хеширования MD5 (класс MD5Cng именованного пространства System.Security.Cryptography). Класс MD5Cng на более низком уровне использует примитивы BCryptCNG.

Cryptographic Application Programming Interface (CryptoAPI) содержит как функции, осуществляющие базовые криптографические операции, так и функции, реализующие преобразования более высокого уровня работу с сертификатами X.509, работу с криптографическими сообщениями PKCS#7 и другие функции, поддерживающие так называемую инфраструктуру открытых ключей (Public Key Infrastructure).

Реализация всех алгоритмов полностью выведена из состава CryptoAPI и реализуется в независимых динамических библиотеках — «криптопровайдерах». CryptoAPI предоставляет конечному пользователю унифицированный интерфейс работы с функциями криптопровайдера.

Общая архитектура CryptoAPI состоит из пяти основных функциональных групп:

1. Базовые криптографические функции:

- функции шифрования/расшифровывания данных;
- функции хеширования и получения цифровой подписи данных;
- функции инициализации криптопровайдера и работы с полученным контекстом;

- функции генерации ключей;
 - функции обмена ключами.
2. Функции кодирования/декодирования. Под кодированием в данном случае подразумевается получение на выходе информации, кодированной в формате ASN.1 (Abstract Syntax Notation One).
 3. Функции работы с сертификатами.
 4. Высокоуровневые функции обработки криптографических сообщений.
 5. Низкоуровневые функции обработки криптографических сообщений.

Служба CNG (Cryptography Next Generation API) поддерживает все алгоритмы, предлагаемые CryptoAPI, а также новые алгоритмы, включая алгоритмы шифрования, цифровых подписей, обмена ключами и хеширования.

На рисунках ниже приведены общие схемы CNG. BCrypt — это подмножество CNG, которое предоставляет такие примитивы криптографии, как средства генерации случайных чисел, функции хеширования, подписи и ключи шифрования. NCrypt обеспечивает механизмы сохранения асимметричных ключей и поддержки аппаратных средств, таких как смарт-карты. BCrypt и NCrypt — имена заголовочных файлов и DLL, предоставляющих базовые сервисы для CNG и высокоуровневую функциональность для хранения ключей соответственно.



Рисунок 2 — Архитектура CNG

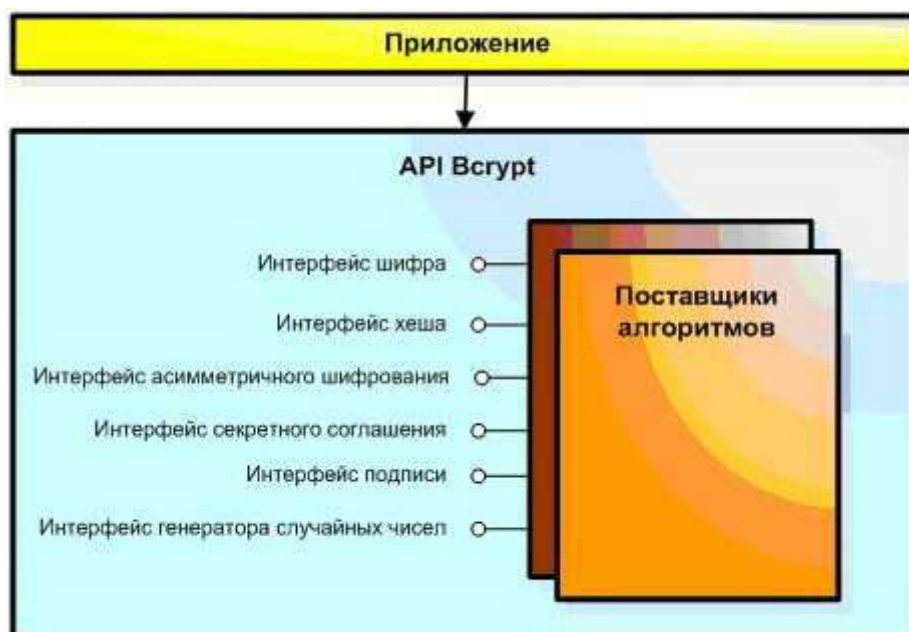


Рисунок 3 — Интерфейсы CNG

1.2.2 Выбор языка разработки

Учитывая наиболее подходящие решению задачи технологии, а также опыт разработки оконных и сетевых приложений, было принято решение ис-

пользовать для реализации клиента язык C#, а для реализации сервера язык Python 3.

C# — элегантный, типобезопасный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде .NET Framework. Он относится к семье языков с СИ-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

С помощью языка C# можно создавать обычные приложения Windows, XML-веб-службы, распределенные компоненты, приложения «клиент-сервер», приложения баз данных и т. д.

C# поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы. Скорость разработки на C# значительно превосходит C++.

Visual C# предоставляет развитый редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и множество других средств, упрощающих разработку приложений на базе языка C# и .NET Framework.

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python органи-

зовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Основные особенности Python:

- Python - интерпретируемый язык программирования. С одной стороны, это позволяет значительно упростить отладку программ, с другой - обуславливает сравнительно низкую скорость выполнения.
- Динамическая типизация. В python не надо заранее объявлять тип переменной, что очень удобно при разработке.
- Хорошая поддержка модульности. Вы можете легко написать свой модуль и использовать его в других программах.
- Встроенная поддержка Unicode в строках. В Python необязательно писать всё на английском языке, в программах вполне может использоваться ваш родной язык.
- Поддержка объектно-ориентированного программирования. При этом его реализация в python является одной из самых понятных.
- Автоматическая сборка мусора, отсутствие утечек памяти.
- Интеграция с C/C++, если возможностей python недостаточно.
- Понятный и лаконичный синтаксис, способствующий ясному отображению кода. Удобная система функций позволяет при грамотном подходе создавать код, в котором будет легко разобраться другому человеку в случае необходимости. Также вы сможете научиться читать программы и модули, написанные другими людьми.
- Огромное количество модулей, как входящих в стандартную поставку Python 3, так и сторонних. В некоторых случаях для написания программы достаточно лишь найти подходящие модули и правильно их скомбинировать. Таким образом, вы можете думать о составлении программы на более высоком уровне, работая с уже готовыми элементами, выполняющими различные действия.

– Кроссплатформенность. Программа, написанная на Python, будет функционировать совершенно одинаково вне зависимости от того, в какой операционной системе она запущена. Отличия возникают лишь в редких случаях, и их легко заранее предусмотреть благодаря наличию подробной документации.

1.2.3 Выбор инструментов разработки

Поскольку программа ориентирована на операционную систему Microsoft Windows, для разработки будут использованы средства, предоставляемые разработчикам под эту ОС:

- интегрированная среда разработки Microsoft Visual Studio 2010;
- программная платформа .NET Framework;
- интегрированная среда разработки PyCharm 2016.2.3.

1.2.3.1 Visual Studio 2010

Visual Studio является на сегодняшний день одной из самых мощных интегрированных сред разработки. Среда поддерживает разработку проектов на языках C, C++, C#, J#, Visual Basic, а также web-приложений и сочетает в себе редактор кода, компилятор, визуальный редактор и средства автоматизированного тестирования.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор.

Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне (например, Visual Assist X, разработанный компанией WholeTomato), включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и VisualSourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

1.2.3.2 Платформа .NET Framework

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общезыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). Затем код либо исполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR встроенный в неё JIT-компилятор «на лету» (justintime) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

1.2.3.3 PyCharm

PyCharm — интегрированная среда разработки для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией JetBrains на основе IntelliJ IDEA.

PyCharm работает под операционными системами Windows, Mac OS X и Linux.

Возможности:

- Статический анализ кода, подсветка синтаксиса и ошибок
- Навигация по проекту и исходному коду: отображение файловой структуры проекта, быстрый переход между файлами, классами, методами и использованиями методов.
- Рефакторинг: переименование, извлечение метода, введение переменной, введение константы, подъём и спуск метода и т. д.
- Инструменты для веб-разработки с использованием фреймворка Django
- Встроенный отладчик для Python
- Встроенные инструменты для юнит-тестирования
- Разработка с использованием Google App Engine
- Поддержка систем контроля версий: общий пользовательский интерфейс для Mercurial, Git, Subversion, Perforce и CVS с поддержкой списков изменений и слияния

1.3 Выбор стратегии разбиения задачи

1.3.1 Перебор вариантов. Построение дерева вариантов

В том случае, когда для решения задачи не удастся разработать эффективный алгоритм решения, используют алгоритм полного перебора.

Основной принцип, на котором базируются методы полного перебора вариантов, состоит в разбиении начальной задачи на подзадачи (в целом представляющих всю задачу) с последующей попыткой разрешить каждую из этих подзадач. Это разбиение описывается деревом, представленным ниже (Рисунок 4), причем вершины изображают подзадачи.

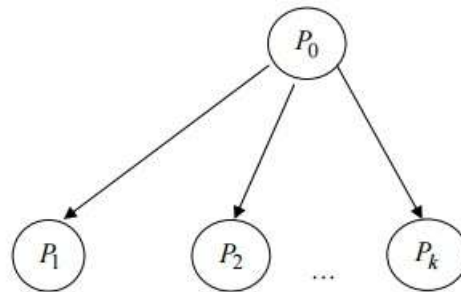


Рисунок 4 — Дерево решений

Выражение «решить подзадачу» означает одну из следующих возможностей:

1. Находится оптимальное решение.
2. Показывается, что значение оптимального решения подзадачи хуже, чем лучшее из найденных решений.
3. Показывается, что подзадача является недопустимой.

Смысл разбиения задачи на подзадачи состоит в том, что эти подзадачи проще решить, так как они имеют меньший размер, или обладают структурой, не присущей первоначальной задаче. Может оказаться, что подзадачу нельзя решить, и эта подзадача также разбивается на новые подзадачи, как это показано на рисунке ниже.

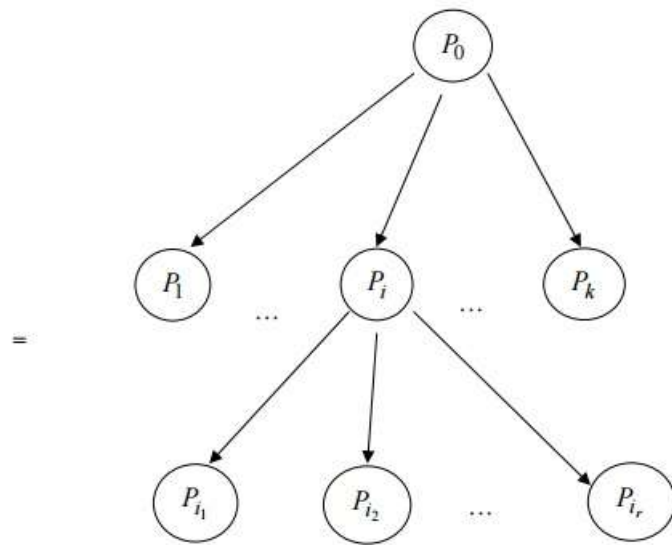


Рисунок 5 — Сложное дерево вариантов

Это разбиение, называемое также ветвлением, повторяется для каждой подзадачи, которая не может быть решена.

На любом этапе полное множество подзадач, требующих решения, представляется множеством конечных вершин всех путей, исходящих из корня дерева решений.

1.3.2 Организация обхода деревьев

1.3.2.1 Фронтальное ветвление

При такой стратегии ветвление выполняется для наиболее перспективной вершины, так начальная задача разбивается на подзадачи, образуя фронт ветвления.

Из этих подзадач выбирается наиболее перспективная подзадача, и разбивается на подзадачи, увеличивая количество вершин. Вид дерева решений при этом типе поиска показан на рисунке ниже.

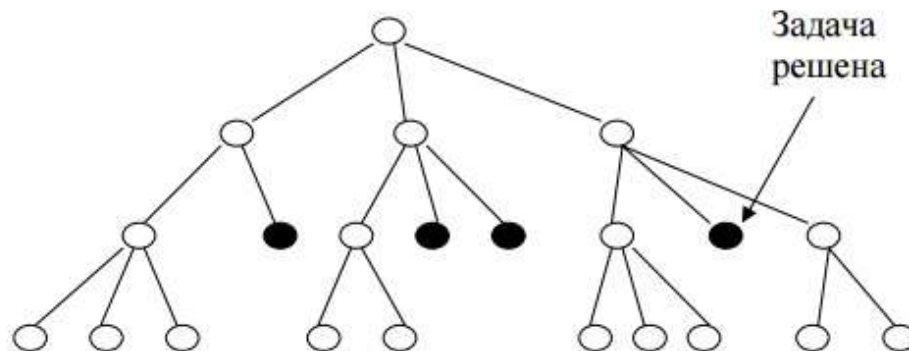


Рисунок 6 — Фронтальное ветвление

1.3.2.2 Одностороннее ветвление

При этом типе поиска, ветвление осуществляется в последней выбранной подзадаче. Такой процесс ветвления продолжается до тех пор, пока не будет порождена подзадача, которую можно решить.

В этом месте делается возврат, т. е. берется предпоследняя порожденная подзадача и ветвление продолжается в соответствующей вершине. При этом типе поиска задачи, получаемые на каждом этапе, хранятся в стеке вместе с исходной задачей.

Вновь получаемые задачи помещаются в стек, а когда подзадача разрешена, она удаляется из стека. Вид дерева решений при этом типе поиска, когда разрешается первая подзадача, показан ниже (Рисунок 7), где порядок приоритета исследования получаемых подзадач отражен соответствующей нумерацией.

Такая стратегия часто называется «поиск с возвратом». Основными ее достоинствами является простота реализации и минимизация используемой памяти, так как в отличие от фронтального ветвления нет необходимости хранить формат вершин кандидатов для ветвления.

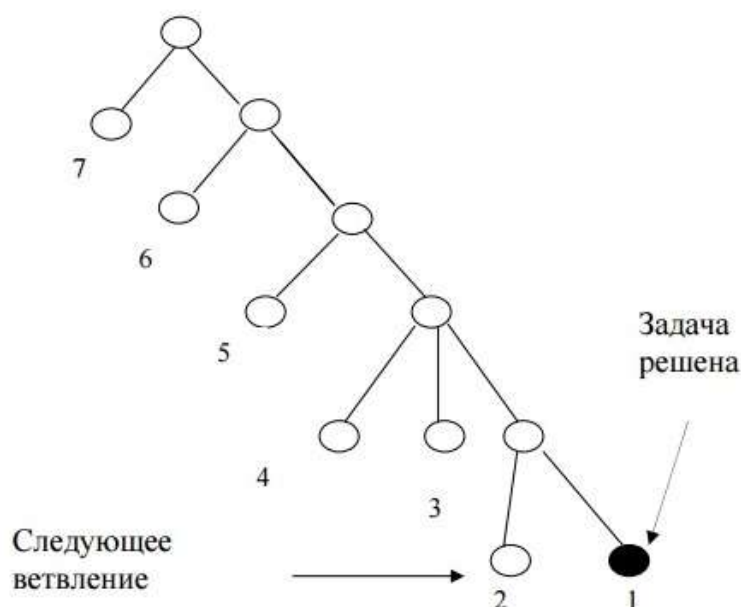


Рисунок 7 — Одностороннее ветвление

1.3.2.3 Выбор способа обхода дерева

Для задачи перебора алфавита подходит фронтальное ветвление. Изначально весь массив вариантов переборов делится на части (для каждого клиента в соответствии с его возможностями), а в последующем еще раз, уже с ориентиром на рассматриваемую длину пароля.

1.3.2.4 Сокращение числа подзадач

Основной целью при организации перебора вариантов решения является сокращение количества решаемых подзадач. Рассмотрим три стратегии, которые предназначены для отсева неперспективных вариантов ветвления.

1.Отсев по повторению (исключение повторяющихся подзадач).

2.Отсев по допустимости (если когда-то пришли в некоторое множество и выяснили, что оно нам ничего допустимого не даст, то незачем в дальнейшем туда ходить).

3.Отсев по рекорду (если заранее можно определить, что некоторая подзадача не даст решения лучше, чем построенное алгоритмом ранее (рекорд), то незачем решать такую подзадачу).

1.4 Выбор методологии разработки

1.4.1 Обзор основных методологий разработки ПО

При проектировании, реализации и тестировании программного обеспечения применяют два подхода:

- восходящий;
- нисходящий.

В литературе встречается ещё один подход, получивший название «расширение ядра». Он предполагает, что в первую очередь проектируют и разрабатывают некоторую основу — ядро программного обеспечения, например, структуры данных и процедуры, связанные с ними. В дальнейшем ядро наращивают, комбинируя восходящий и нисходящий методы. На практике данный подход в зависимости от уровня ядра практически сводится либо к нисходящему, либо к восходящему подходам, рассмотренным ниже.

1.4.1.1 Восходящий подход

При использовании восходящего подхода сначала проектируют и реализуют компоненты нижнего уровня, затем предыдущего и т. д. По мере завершения тестирования и отладки компонентов осуществляют их сборку, причём компоненты нижнего уровня при таком подходе часто помещают в библиотеки компонентов.

Для тестирования и отладки компонентов проектируют и реализуют специальные тестирующие программы. Подход имеет следующие недостатки:

- увеличение вероятности несогласованности компонентов вследствие неполноты спецификаций;
- наличие издержек на проектирование и реализацию тестирующих программ, которые нельзя преобразовать в компоненты;

– позднее проектирование интерфейса, а соответственно невозможность продемонстрировать его заказчику для уточнения спецификаций и т. д.

Исторически восходящий подход появился раньше, что связано с особенностью мышления программистов, которые в процессе обучения привыкают при написании небольших программ сначала детализировать компоненты нижних уровней (подпрограммы, классы). Это позволяет им лучше осознавать процессы верхних уровней. При промышленном изготовлении программного обеспечения восходящий подход в настоящее время практически не используют.

1.4.1.2 Нисходящий подход

Нисходящий подход предполагает, что проектирование и последующая реализация компонентов выполняется «сверху-вниз», т. е. вначале проектируют компоненты верхних уровней иерархии, затем следующих и так далее до самых нижних уровней. В той же последовательности выполняют и реализацию компонентов. При этом в процессе программирования компоненты нижних, ещё не реализованных уровней заменяют специально разработанными отладочными модулями — «заглушками», что позволяет тестировать и отлаживать уже реализованную часть.

Кроме того, при прочих равных условиях сложные модули должны разрабатываться прежде простых, так как при их проектировании могут выявиться неточности в спецификациях, а чем раньше это произойдёт, тем лучше.

Нисходящий подход допускает нарушение нисходящей последовательности разработки компонентов в специально оговорённых случаях. Так, если некоторый компонент нижнего уровня используется многими компонентами более высоких уровней, то его рекомендуют проектировать и разрабатывать раньше, чем вызывающие его компоненты. И, наконец, в первую очередь проектируют и реализуют компоненты, обеспечивающие обработку правиль-

ных данных, оставляя компоненты обработки неправильных данных напоследок.

Нисходящий подход обычно используют и при объектно-ориентированном программировании. В соответствии с рекомендациями подхода вначале проектируют и реализуют пользовательский интерфейс программного обеспечения, затем разрабатывают классы некоторых базовых объектов предметной области, а уже потом, используя эти объекты, проектируют и реализуют остальные компоненты. Нисходящий подход обеспечивает:

- максимально полное определение спецификаций проектируемого компонента и согласованность компонентов между собой;
- раннее определение интерфейса пользователя, демонстрация которого заказчику позволяет уточнить требования к создаваемому программному обеспечению;
- возможность нисходящего тестирования и комплексной отладки.

1.4.2 Выбор пути разработки проекта

В данном проекте я посчитала целесообразным использовать нисходящий подход программирования. Сначала была создана форма, затем нарисованы рабочие окна (класс Controls) с основным интерфейсом при помощи инструмента Windows Forms. После создания внешнего вида приложения были постепенно написаны обработки всех задействованных элементов (button, check-box, progress-bar, textBox).

2 Реализация многопользовательской клиент-серверной Web-системы для управления распределенными вычислениями

2.1 Способ распределения подзадач

Последовательная область вычислений содержит аналитическую часть по разбиению основной задачи на подзадачи. На первом этапе выделяется первая группа подзадач — одинаковых по количеству перебираемых вариантов. Эти подзадачи используются для определения коэффициентов производительности клиентов и называются в рамках данной работы статистическими. Количество статистических подзадач определяется количеством клиентов со статусом готовности на момент создания задачи.

Подзадачи, выделяемые для обработки на следующих этапах, имеют различную ресурсоемкость. Самые ресурсоемкие подзадачи направляются наиболее высокопроизводительным клиентам, наименее ресурсоемкие подзадачи — клиентам с наименьшим коэффициентом производительности.

2.1.1 Определение размера статистической подзадачи

Чтобы можно было с уверенностью определить работоспособность компьютера, необходимо определить размер подзадачи, который бы выполнялся бы в среднем несколько секунд. Определение размера задачи для определения коэффициента производительности клиентов основано на результатах оценки производительности на разных процессорах — Core i3, Core i5, Celeron, AMD (Таблица 1).

На основании анализа полученных статистических данных было введено эталонное значение размера одной задачи, обрабатываемой клиентом, для получения данных о производительности — 5 миллионов переборов.

Таблица 1 – Статистические данные

<div>Размер задачи (переборов)</div> <div>Процессор</div>	500.000	1.000.000	4.000.000	11.000.000
Intel Celeron N2840	13 сек	34 сек	63 сек	255 сек
AMD A4 5000	9 сек	27 сек	60 сек	245 сек
Intel Core i3 2310M	8 сек	25 сек	47 сек	210 сек
Intel Core i5 3570K	2 сек	5 сек	16 сек	60 сек

Далее рассмотрим алгоритм вычисления размера тестовой статистической задачи для клиента:

Объем данных, используемых для определения производительности компьютера (объем статистических исследований), не должен превышать значение X/N , где X — общее количество переборов, а N — количество действовавших клиентов.

$$P = \frac{X}{N}, \text{ где } P \text{ — общее количество переборов под статистику}$$

Далее вычисляем размер тестовой подзадачи для одного клиента:

$$T = \frac{P}{N}, \text{ где } T \text{ — размер одной задачи для одного клиента для оценки статистики производительности.}$$

Размер задачи должен быть кратен размеру используемого алфавита:

$$T = T - T \% A, \text{ где } A \text{ — размер заявленного алфавита, а } \% \text{ — остаток от деления.}$$

В качестве эталонного размера подзадачи (E) по результатам статистики был выбран размер 5 миллионов комбинаций.

Если $T > E$, за размер подзадачи берем эталонный ($T = E$), иначе, оставляем неизменным.

2.1.2 Коэффициент производительности клиента

По результатам выполнения тестовых статистических подзадач (одинаковых по трудоемкости) и на основании данных о времени выполнения подзадачи каждым клиентом, выполняется подсчет коэффициента производительности клиента. Клиент с наивысшими показателями производительности получает коэффициент $k=1$;

Чтобы определить коэффициент производительности, необходимо взять время первого справившегося ($T_э$) и разделить на время текущего клиента ($T_т$);

$$k = \frac{T_э}{T_т}$$

По получившимся коэффициентам распределяем уже оставшиеся подзадачи. Клиенту с коэффициентом $k=1$ самую большую, а с самым маленьким коэффициентом ($k = \min$) самую маленькую.

2.2 Описание разработанного программного обеспечения

Разработанное программное обеспечение представляет собой клиент-серверную архитектуру.

Сервер представлен сайтом, написанным на Python при помощи Django. Сервер постоянно обращается к базе данных Sqlite для хранения информации. Клиент представлен службой, написанной на C#.

Клиент каждые 2 секунды отправляет запрос серверу, а тот, если есть подзадача, требующая решения, отправляет ее клиенту. После выполнения задачи, клиент в запросе серверу отправляет результат вычислений.

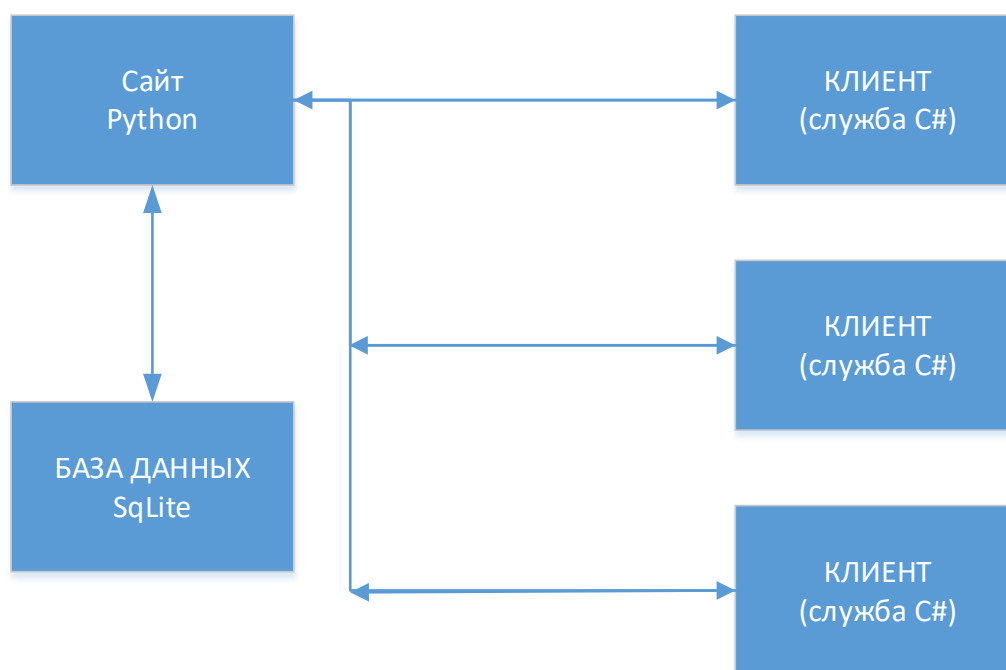


Рисунок 8 — Клиент-серверная архитектура

2.2.1 Сервер

2.2.1.1 Web-портал

На сервере работает web-портал с возможностью регистрации пользователей.

На главной странице для незарегистрированных пользователей выводится инструкция по работе с ресурсом (Рисунок 9).



Рисунок 9 — Вкладка ДОМОЙ для незарегистрированного пользователя

Для зарегистрированных пользователей выводится список клиентов с их статусом готовности (Рисунок 10). Статус готовности определяет то, вошел клиент в систему или нет. Любой зарегистрированный клиент может поставить в очередь задачу, выполнение которой будет осуществляться авторизованными в данный момент пользователями.

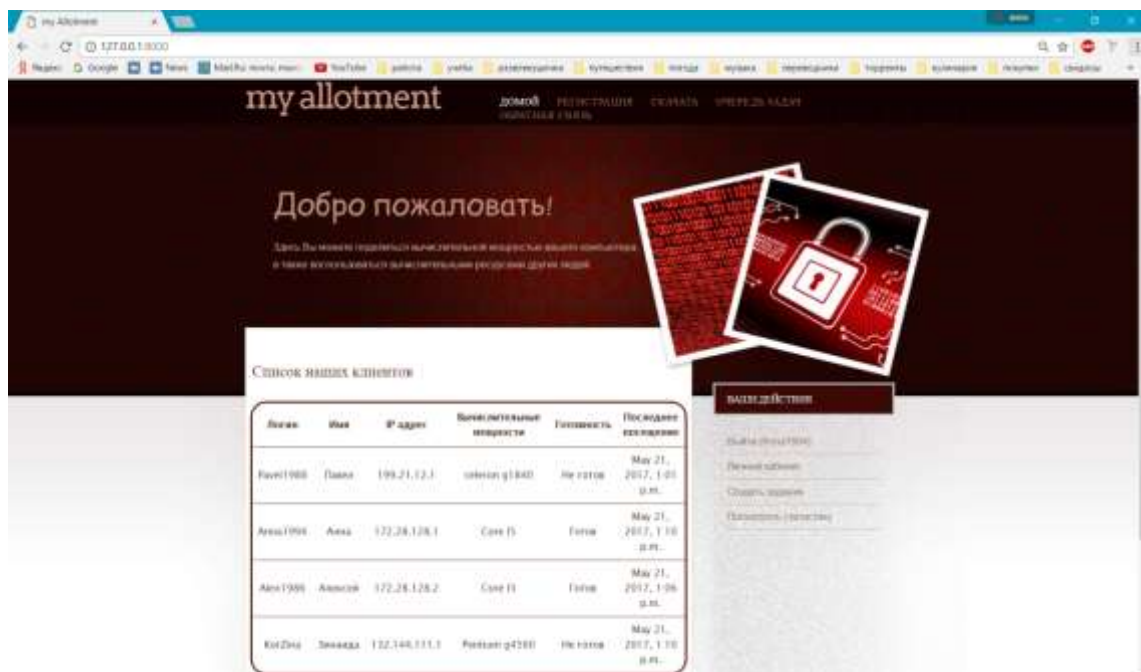


Рисунок 10 — Вкладка ДОМОЙ для зарегистрированного пользователя

Для прохождения регистрации пользователь, в соответствии с инструкцией во вкладке «Домой», должен скачать службу-клиент во вкладке «Скачать» (Рисунок 11) .



Рисунок 11 — Вкладка СКАЧАТЬ для получения клиентской службы

Для регистрации необходимо перейти во вкладку «Регистрация» (Рисунок 12) и заполнить поля. В поле «Хэш-идентификатор клиента» требуется записать содержимое файла YourHash.txt, который появится после установки службы-клиента.

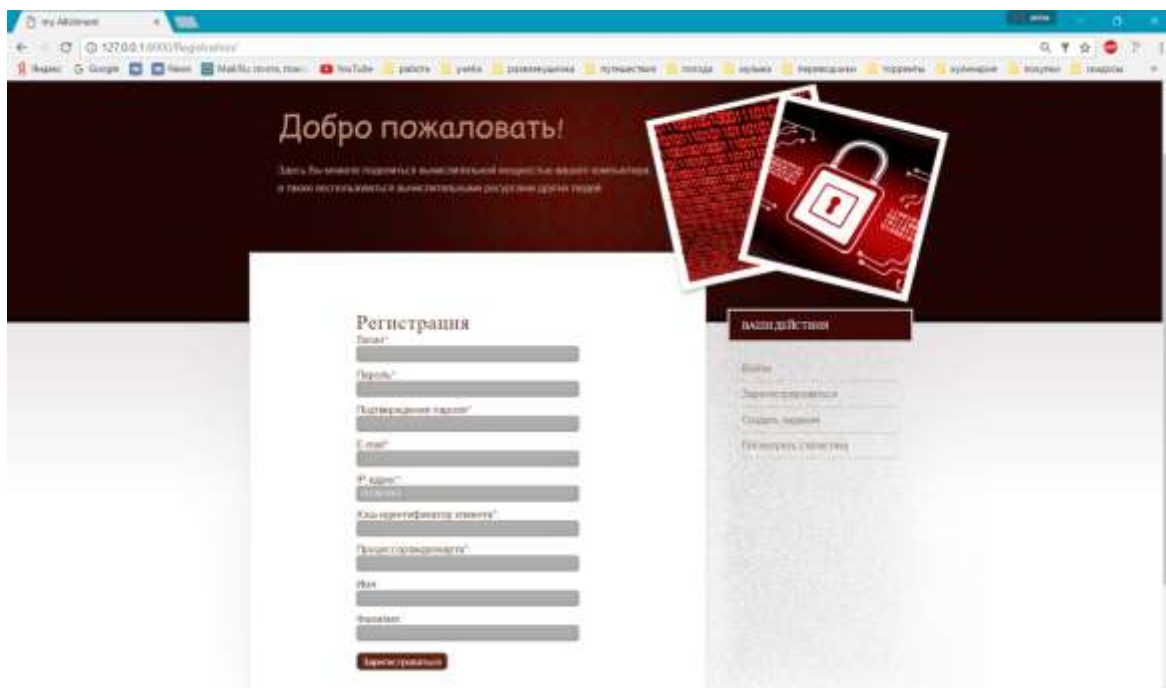


Рисунок 12 — Вкладка РЕГИСТРАЦИЯ

После успешной регистрации для начала работы необходимо войти в систему (Рисунок 13), перейдя во вкладку «Войти».

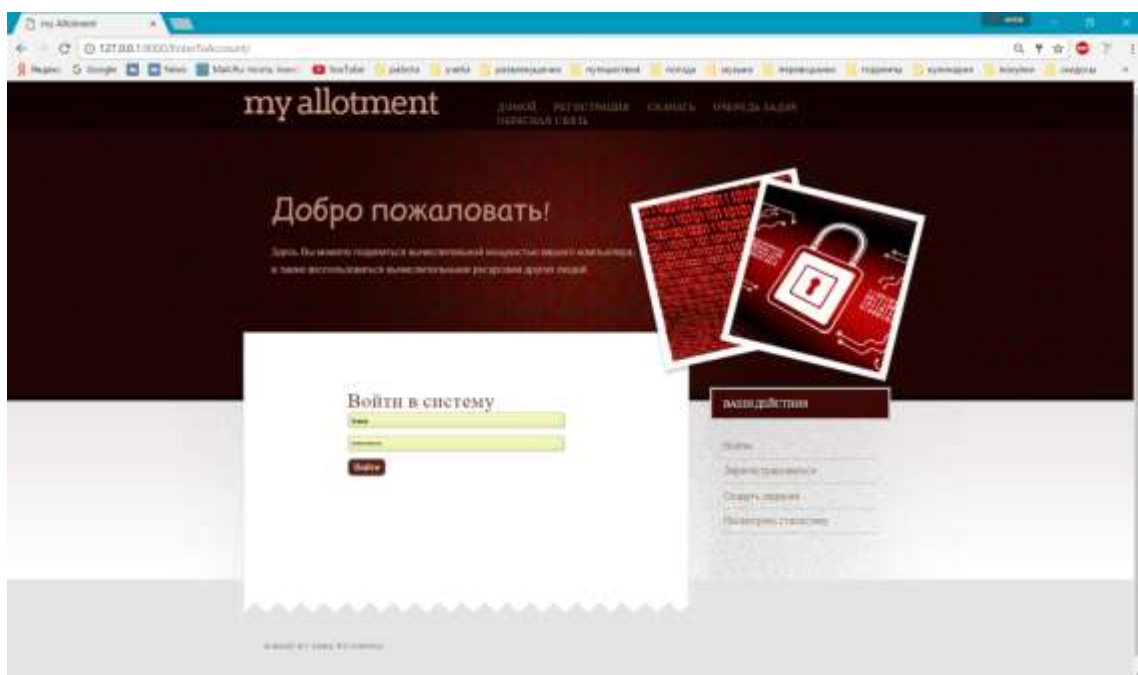


Рисунок 13 — Вкладка ВОЙТИ для входа в систему

После входа в систему, у пользователя есть возможность посмотреть и отредактировать личные данные, проследить историю созданных ранее задач во вкладке «Личный кабинет» (Рисунок 14).

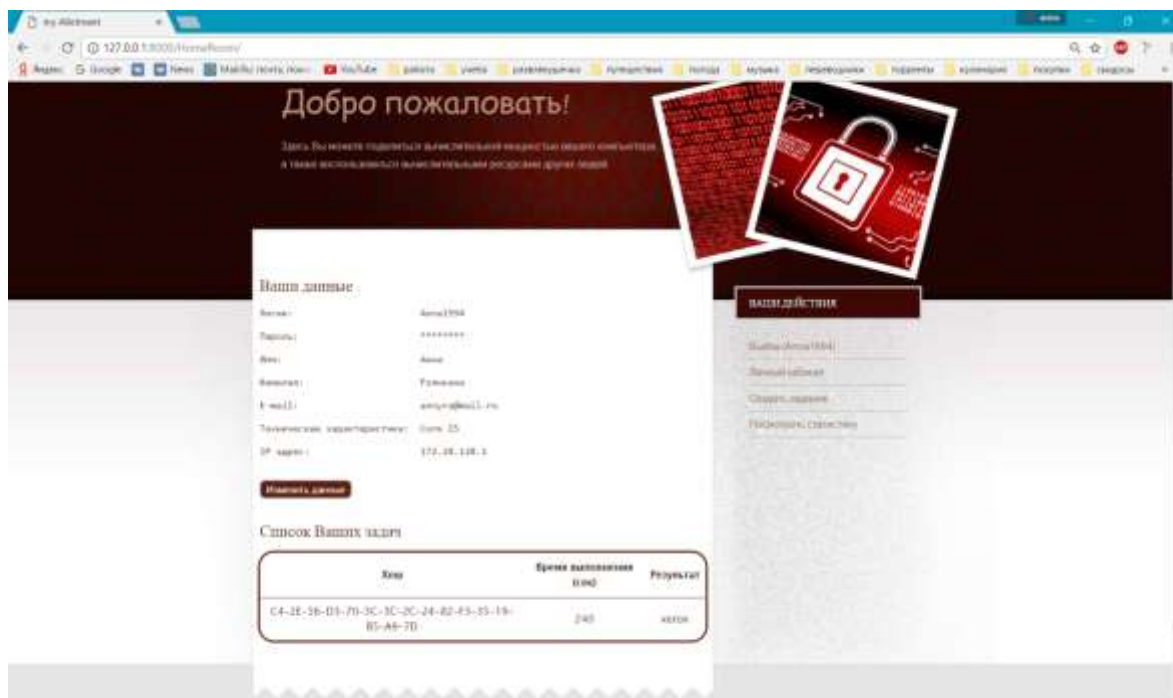


Рисунок 14 — Вкладка ЛИЧНЫЙ КАБИНЕТ

Если у пользователя есть хэш-значение, по которому необходимо найти пароль, следует перейти во вкладку «Создать задание» (Рисунок 15), где указывается хэш, алфавит, и предполагаемая максимальная длина пароля.

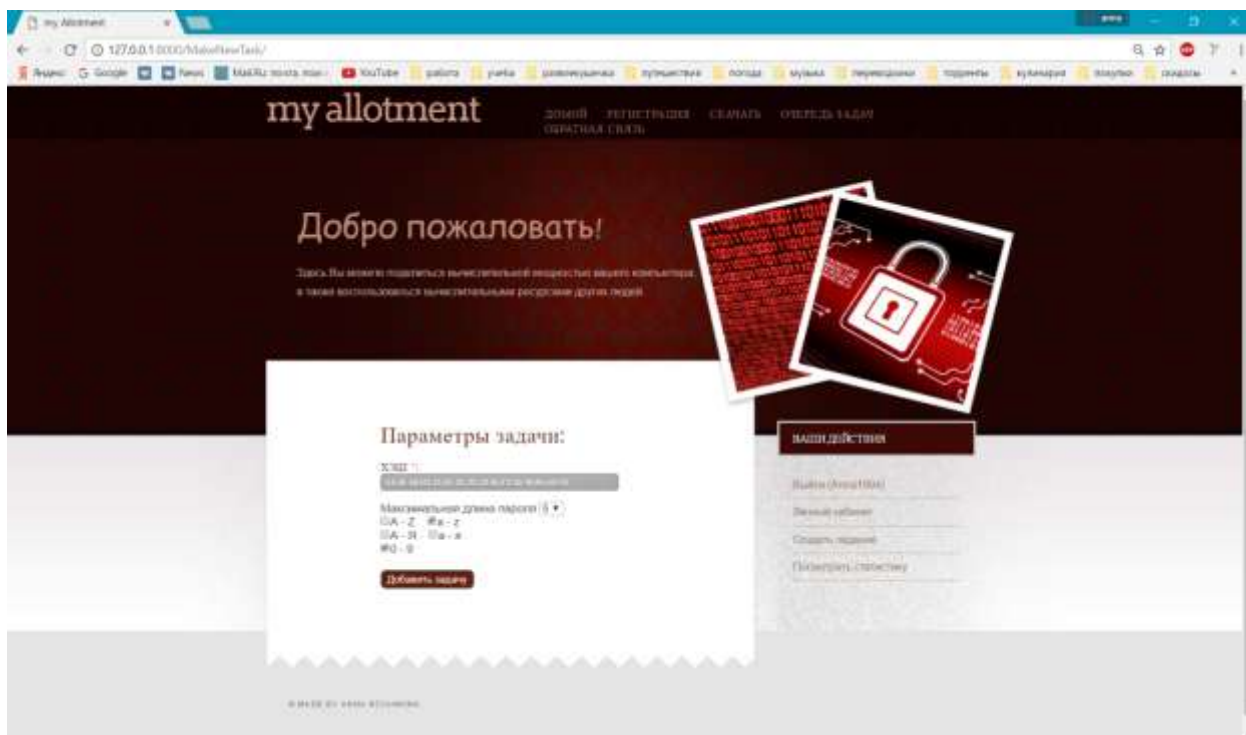


Рисунок 15 — Вкладка СОЗДАТЬ ЗАДАНИЕ

Во вкладке «Статистика» можно посмотреть, какой процент из выполненных подзадач пришелся на конкретного клиента (Рисунок 16).

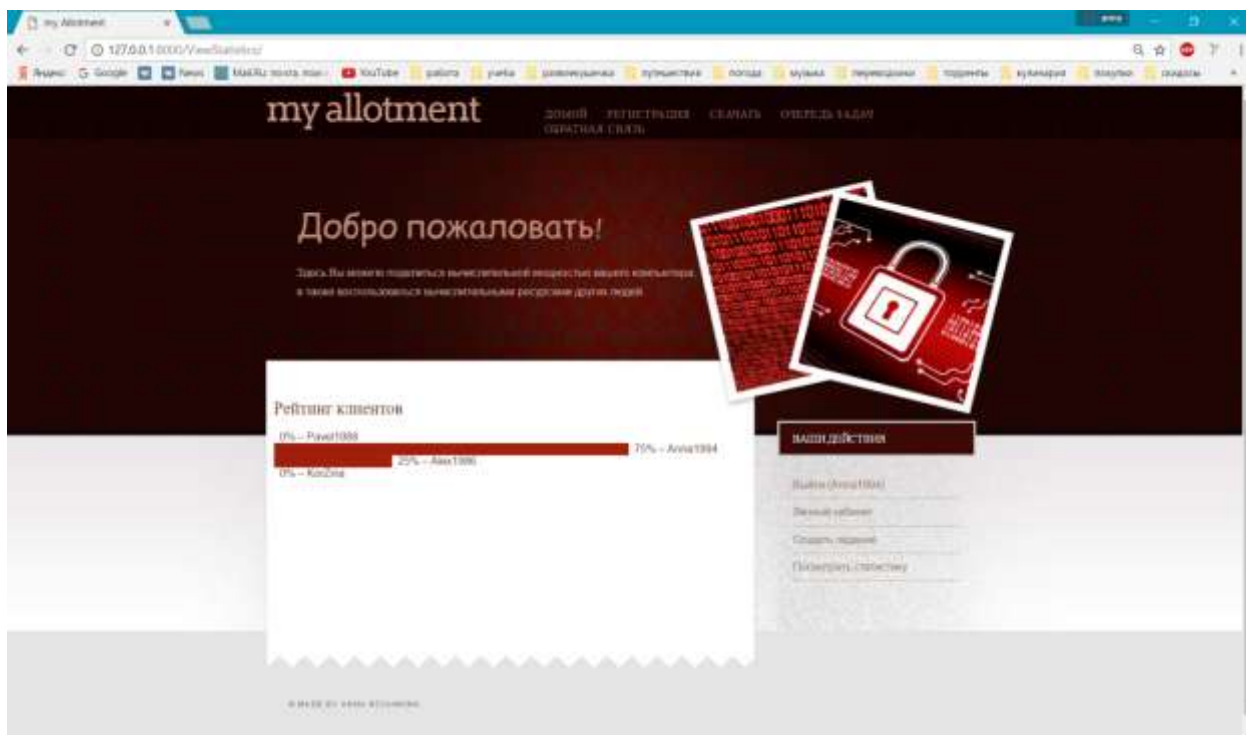


Рисунок 16 — Вкладка СТАТИСТИКА

Если есть пожелания или замечания к работе ресурса, во вкладке «Обратная связь» есть возможность обратиться к разработчику. Сообщение придет на электронную почту и будет обязательно рассмотрено (Рисунок 17 и Рисунок 18).

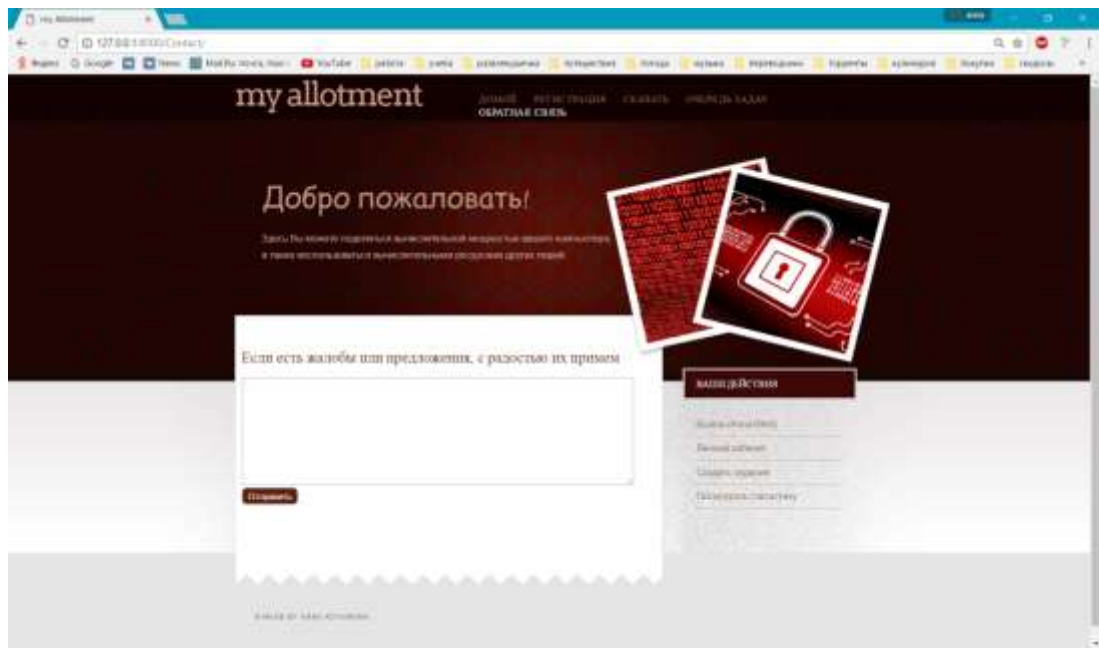


Рисунок 17— Вкладка ОБРАТНАЯ СВЯЗЬ

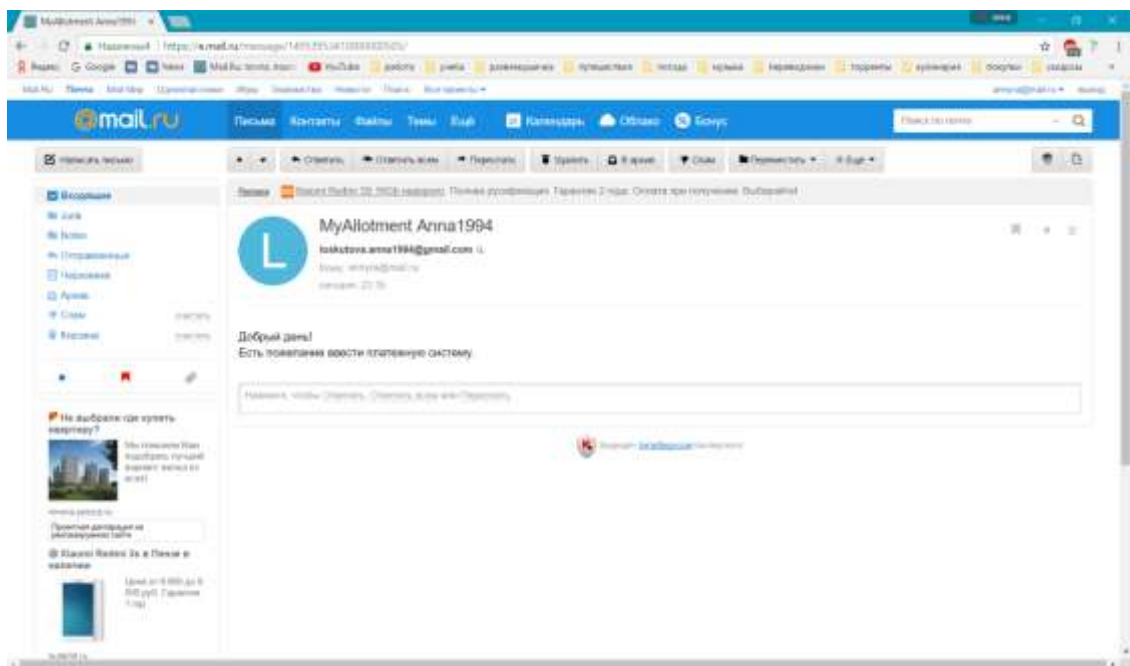


Рисунок 18— Сообщение от Anna1994

2.2.1.2 База данных

Для хранения данных, сервер использует СУБД SQLite.

Для работы сайта было создано 4 таблицы (Рисунок 19):

- Для хранения статических данных зарегистрированных пользователей (SClients - Рисунок 20)
- Для хранения динамически изменяемых данных зарегистрированных пользователей (CclientsParams - Рисунок 21)
- Для хранения списка задач (Ctasks - Рисунок 23)
- Для хранения списка подзадач (CtaskParts - Рисунок 22)

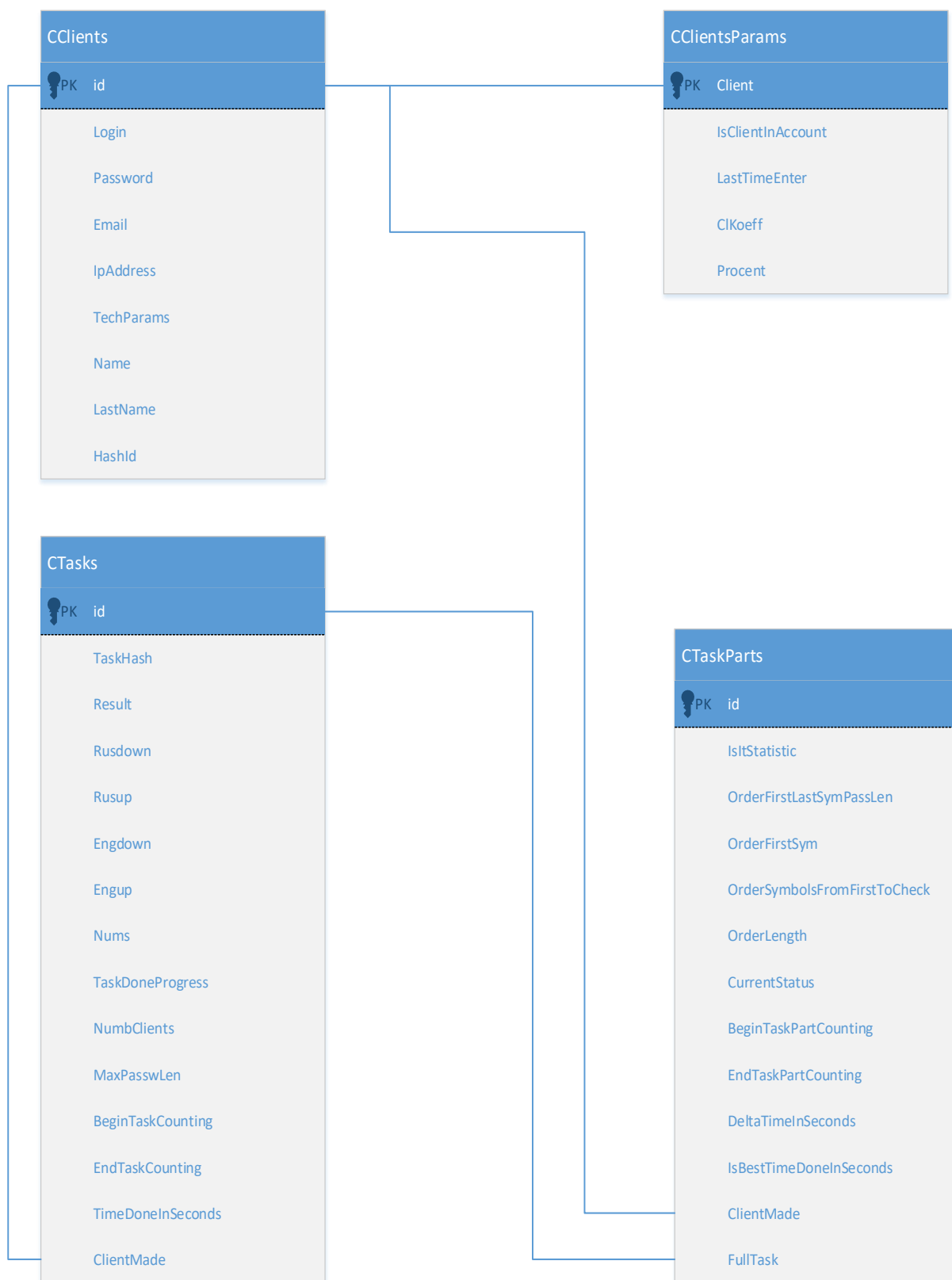


Рисунок 19 — Связи между таблицами в базе данных сервера

Поля таблицы CClients:

- id — первичный ключ
- Login — псевдоним пользователя (CharField)
- Password — пароль пользователя (CharField)
- Email — электронная почта пользователя (CharField)
- IPAddress — IP адрес пользователя (CharField)
- TechParams — технические параметры компьютера клиента (CharField)
- Name — имя клиента (CharField)
- LastName — фамилия клиента (CharField)
- HashId — Идентификатор клиента по hash-значению (CharField)

Поля таблицы CclientsParams:

- Client — связь с полем таблицы CClients, первичный ключ
- IsClientInAccount — признак входа клиента в аккаунт (BooleanField)
- LastTimeEnter — время и дата последнего входа (DateTimeField)
- ClKoeff — коэффициент клиента по производительности (FloatField)
- Procent — Процент участия в выполнении заданий (IntegerField)

Поля таблицы Ctasks:

- id — первичный ключ
- TaskHash — исходный хэш (CharField)
- Result — найденный пароль (CharField)
- Rusdown — признак есть ли русские символы нижнего регистра в пароле (BooleanField)
- Rusup — признак есть ли русские символы верхнего регистра в пароле (BooleanField)
- Engdown — признак есть ли английские символы нижнего регистра в пароле (BooleanField)

- Engur — признак есть ли английские символы верхнего регистра в пароле (BooleanField)
- Nums — признак есть ли цифры в пароле (BooleanField)
- TaskDoneProgress — признак завершена ли задача (IntegerField)
- NumClients — предполагаемое количество клиентов (IntegerField)
- MaxPasswLen — максимальная длина пароля (IntegerField)
- BeginTaskCounting — время и дата начала расчета (IntegerField)
- EndTaskCounting — время и дата конца расчета (DateTimeField)
- ClientMade — связь с полем таблицы CClients (ForeignKey)
- TimeDoneInSeconds — время выполнения задачи (IntegerField)

Поля таблицы CtaskParts:

- id — первичный ключ
- IsItStatistic — признак определения коэффициента или реальное задание (BooleanField)
- OrderFirstLastSymPassLen — длина пароля из которой мы считаем символы от буквы до буквы (IntegerField)
- OrderFirstSym — порядок алфавита первого символа пароля (IntegerField)
- OrderSymbolsFromFirstToCheck — количество порядков символов мы должна проверить от OrderFirstSym (IntegerField)
- OrderLength — длины паролей на данном отрезке задания (CharField)
- CurrentStatus — текущий статус отрезка заданий (IntegerField)
- BeginTaskPartCounting — время и дата начала расчета (DateTimeField)
- EndTaskPartCounting — время и дата конца расчета (DateTimeField)
- DeltaTimeInSeconds — время, которое заняла данная подзадача (IntegerField)

- IsBestTimeDoneInSeconds — флаг лучшее ли время было в рамках этой задачи (IntegerField)
- ClientMade — связь с полем таблицы CClients (ForeignKey)
- FullTask — связь с полем таблицы CTasks (ForeignKey)

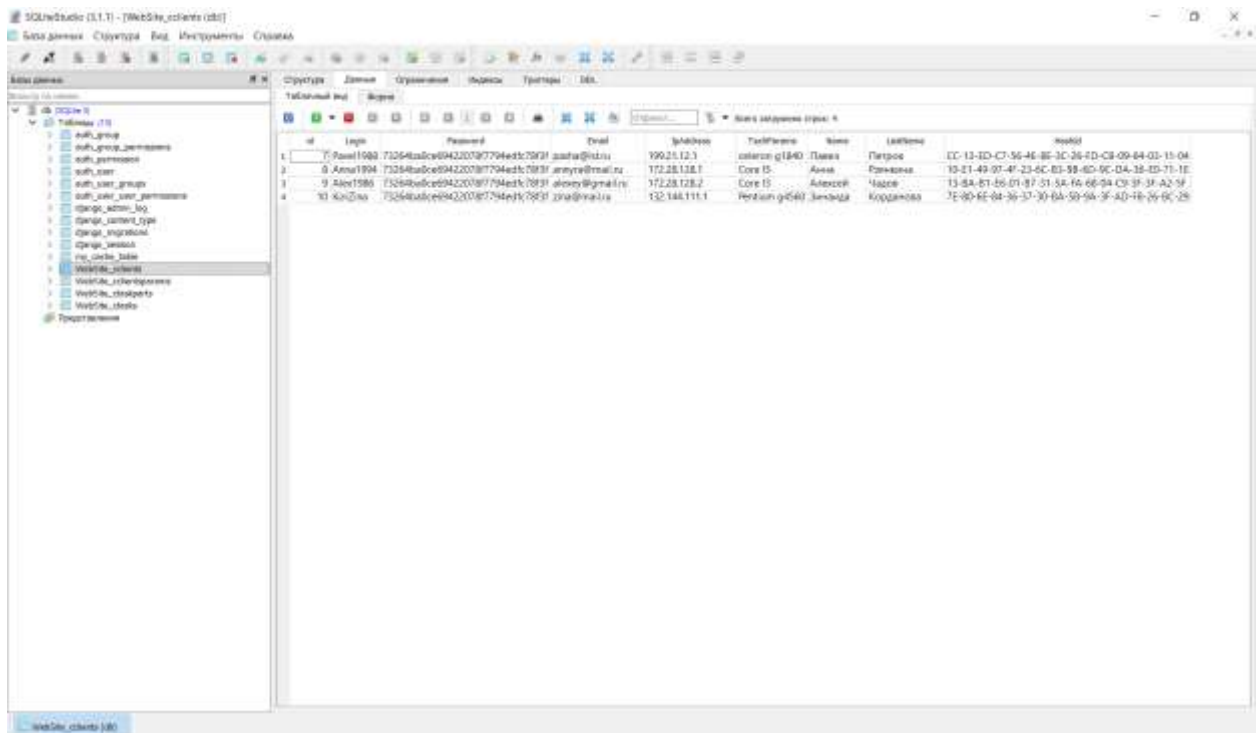


Рисунок 20 — Представление в БД статических данных зарегистрированных пользователей

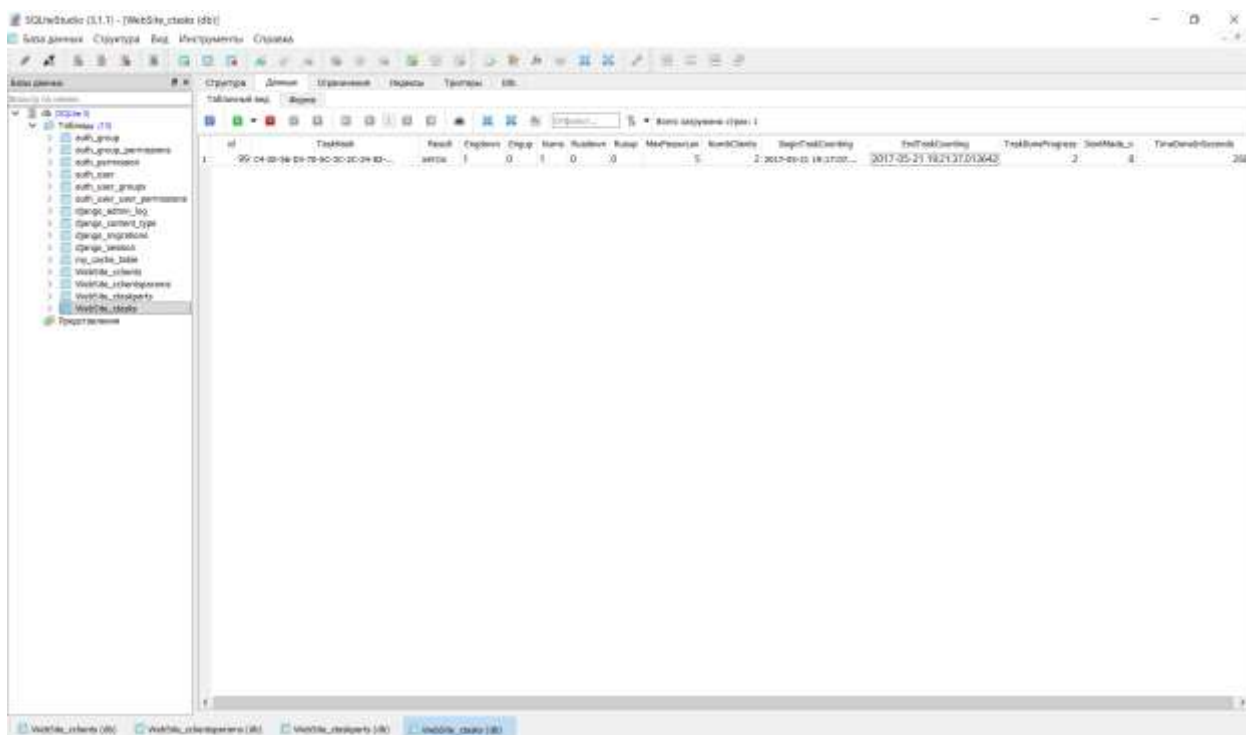


Рисунок 23 — Представление в БД списка задач

2.2.2 Клиент

Клиентская часть представлена службой windows, написанной на языке C#. По умолчанию тип запуска службы автоматический.

Для начала работы с системой, пользователь скачивает во вкладке сайта-сервера «Скачать» архив, содержащий пакет установки со службой-клиентом. В полученном архиве необходимо запустить файл InstallClient.msi.

Ниже представлены этапы установки службы.

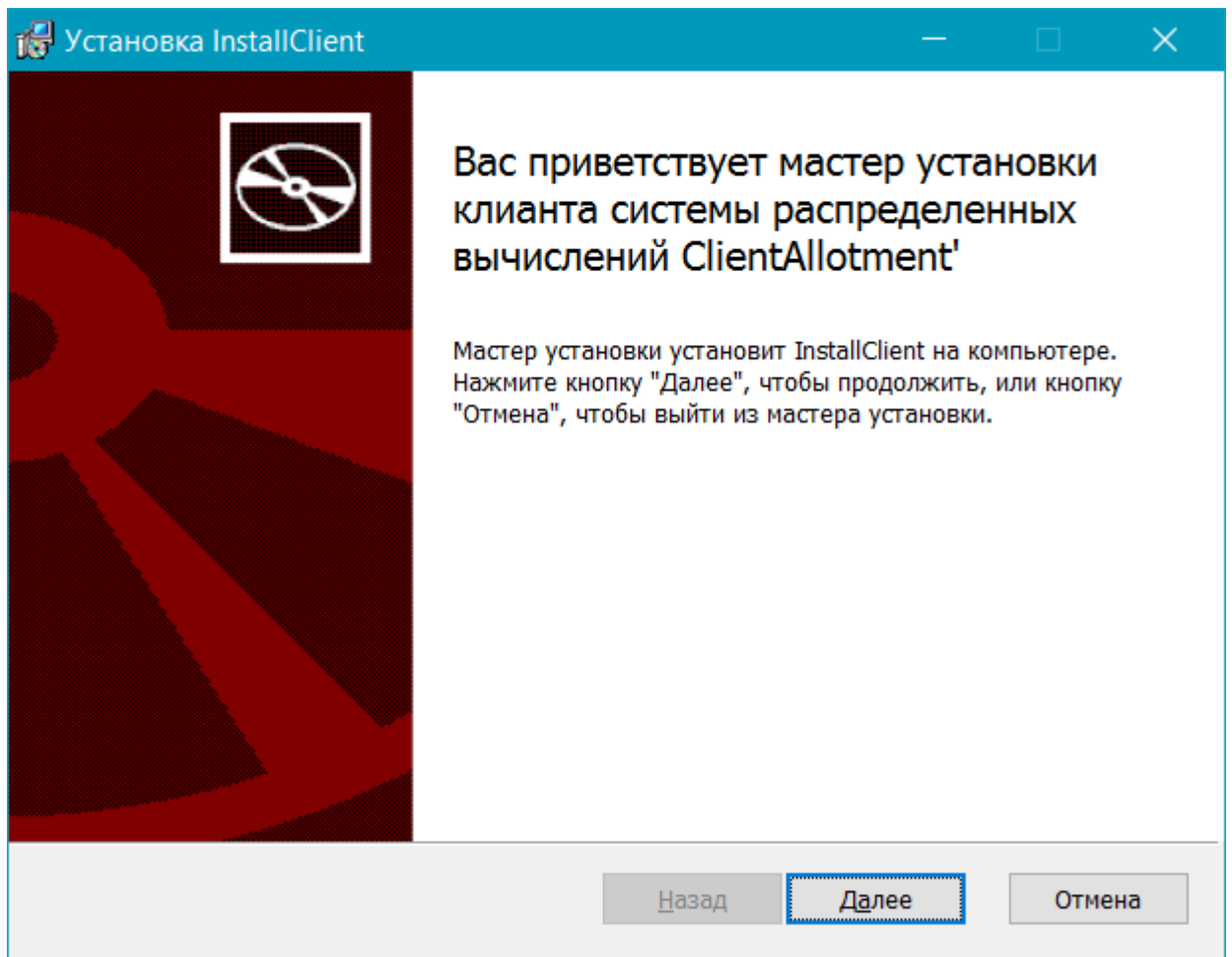


Рисунок 24 — Установка службы-клиента

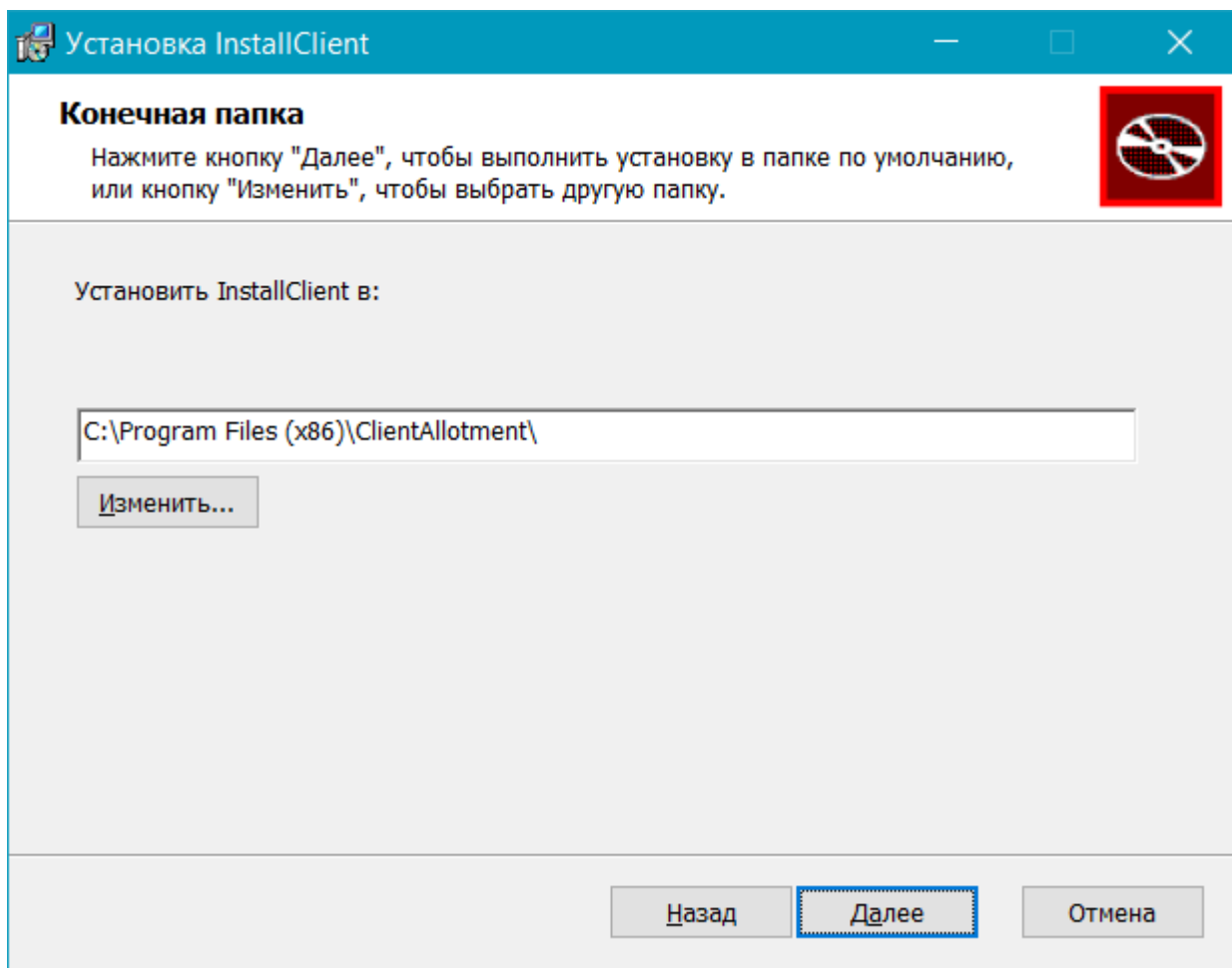


Рисунок 25 — Установка службы-клиента. Выбор папки установки

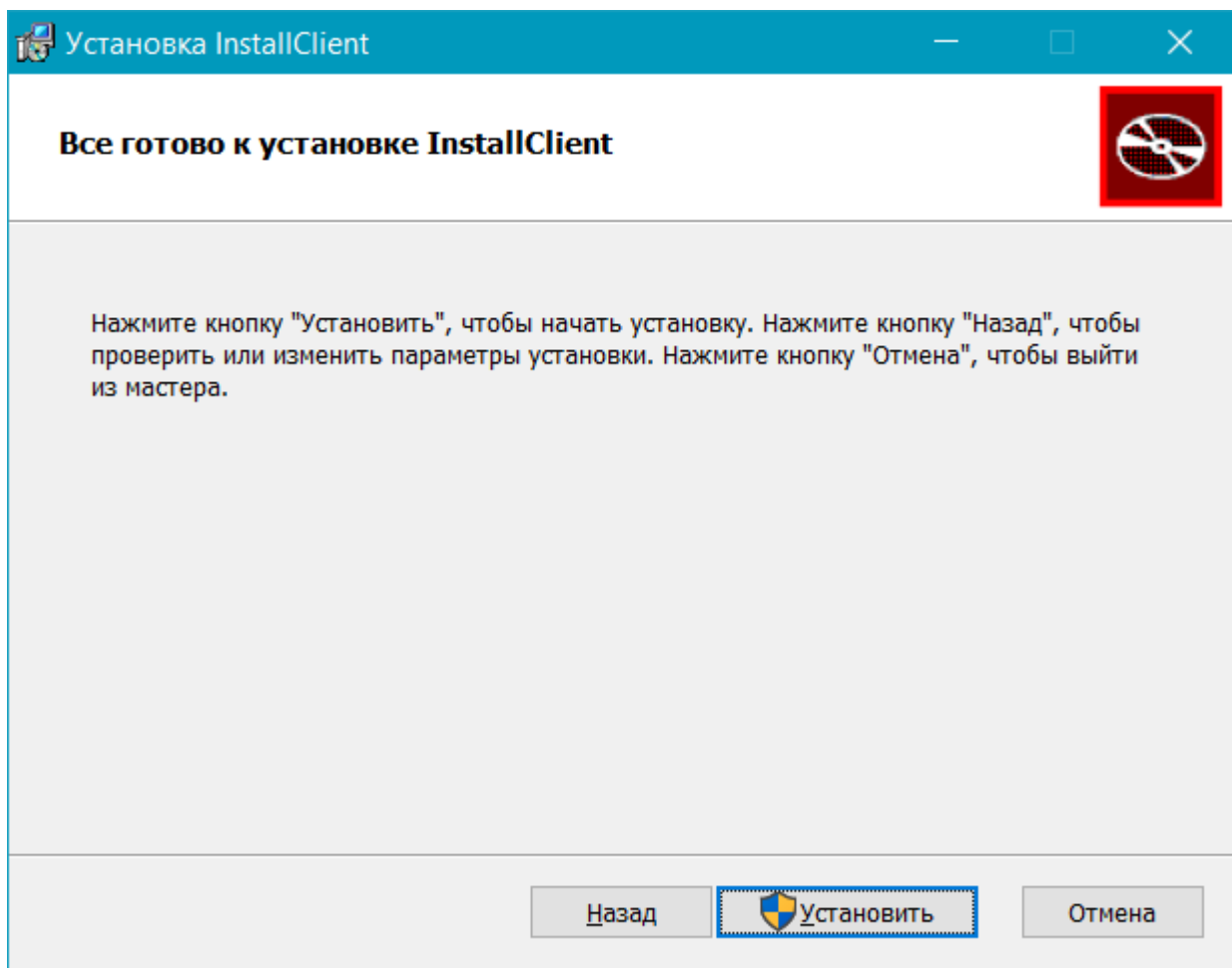


Рисунок 26 — Установка службы-клиента

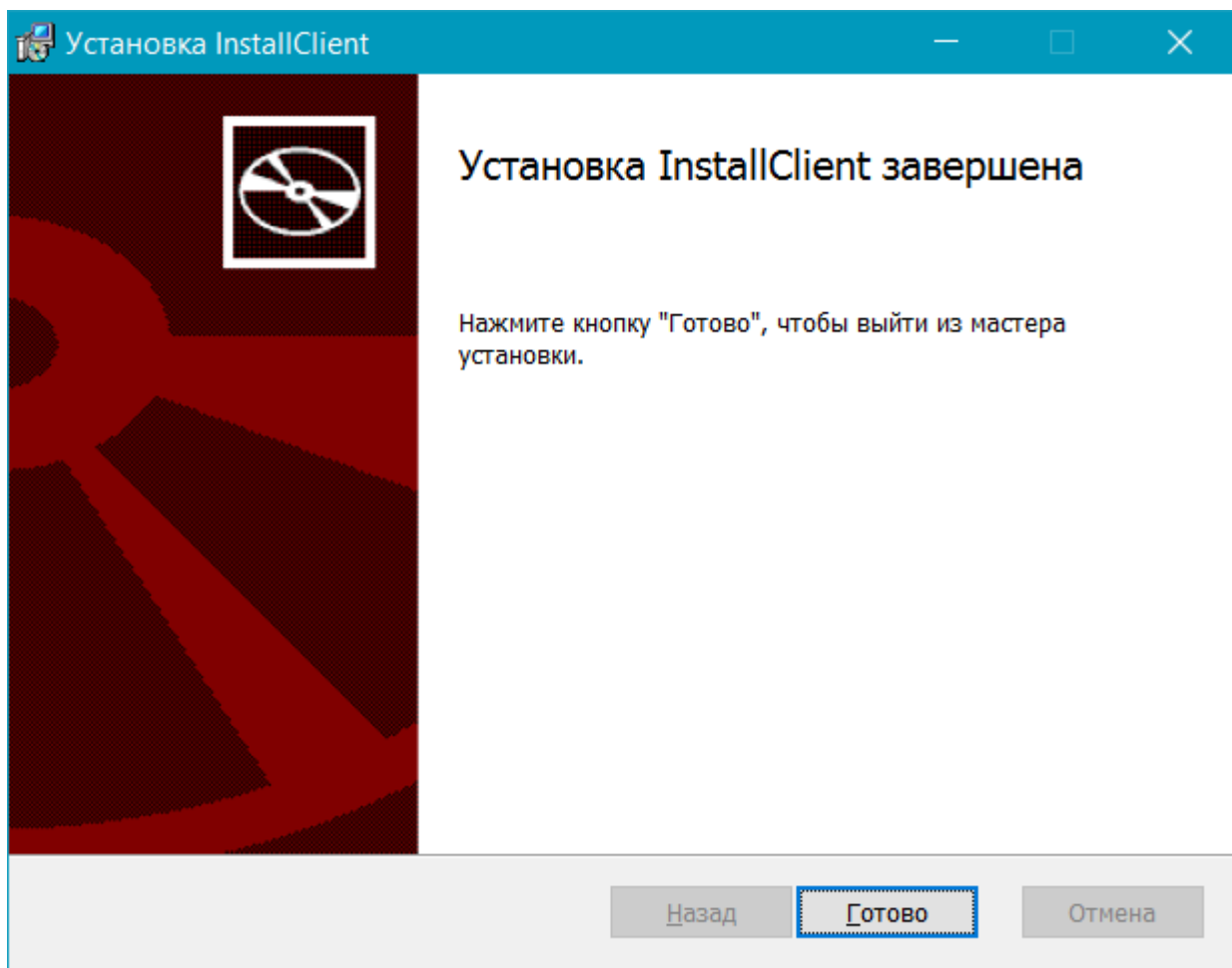


Рисунок 27 — Завершение установки службы-клиента

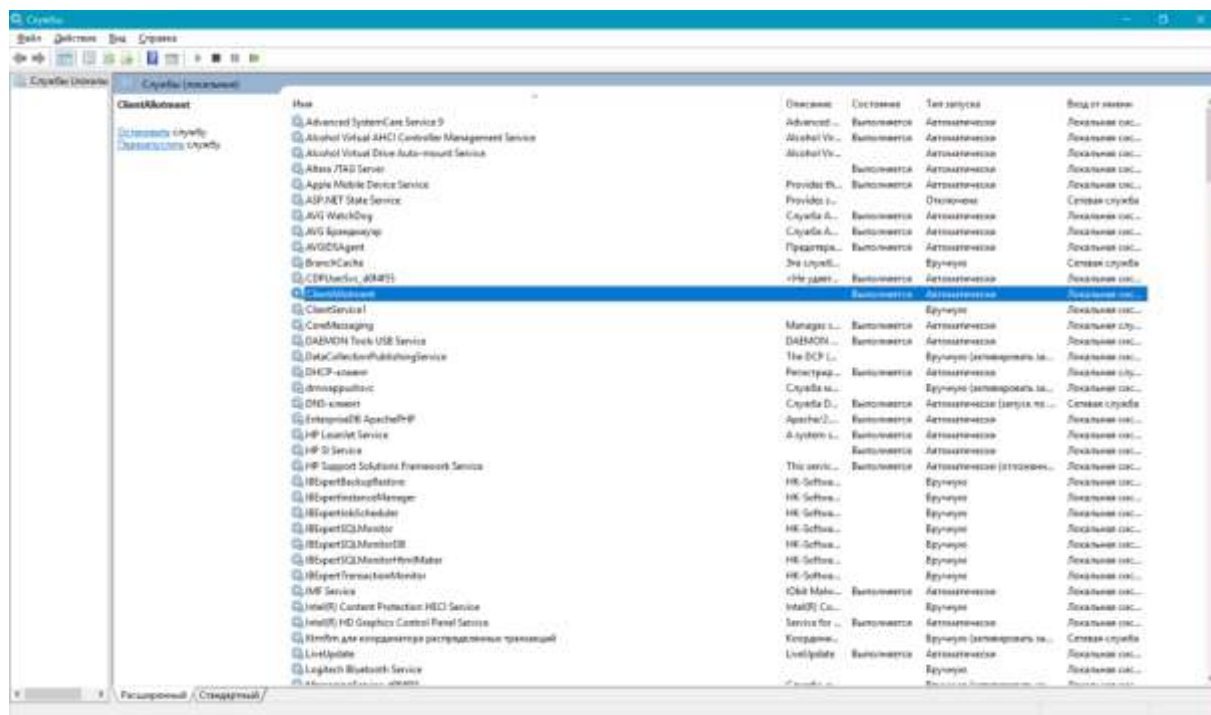


Рисунок 28 — Установленная служба ClientAllotment

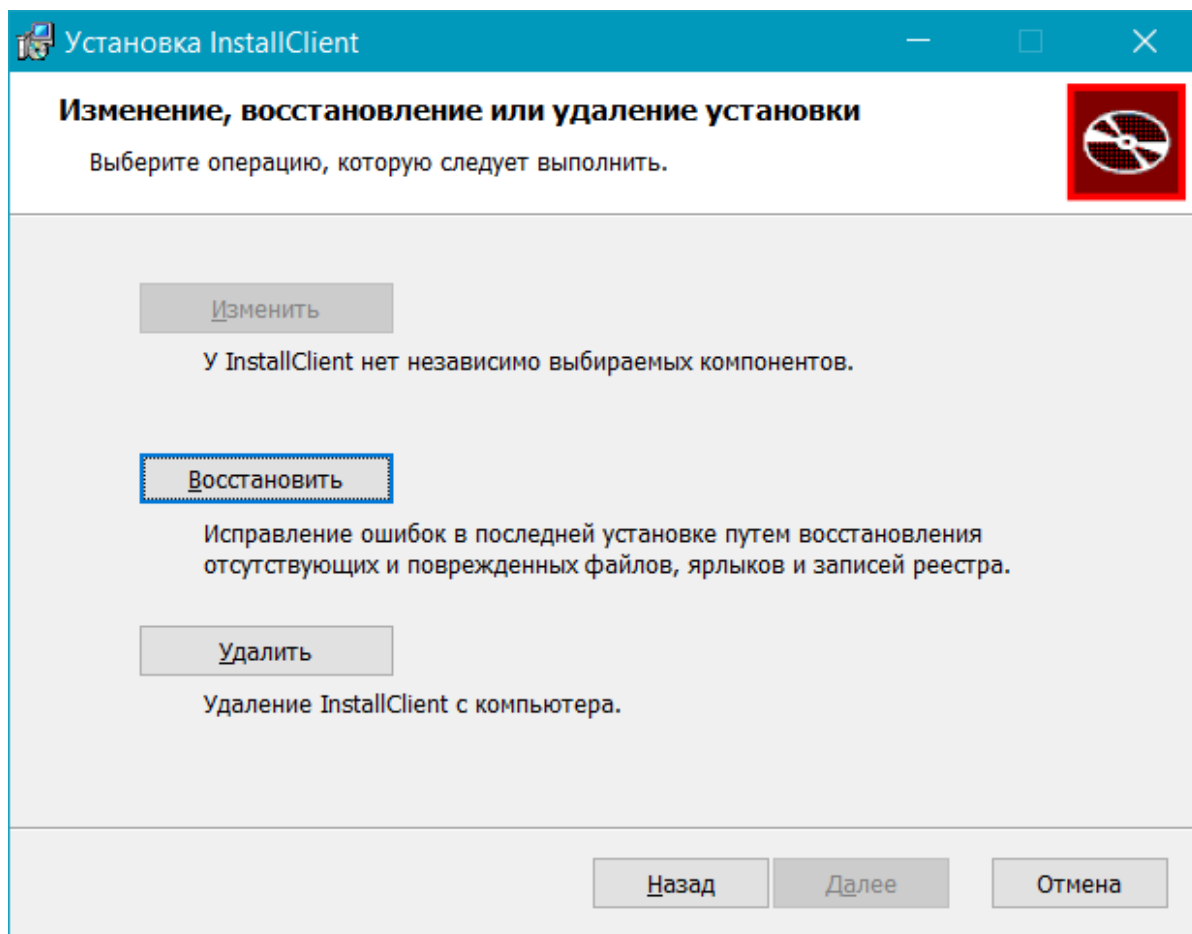


Рисунок 29 — Восстановление или удаление службы-клиента

После установки службы средствами Windows Installer, в списке системных служб будет видна установленная служба ClientAllotment. При старте системы она автоматически начинает работу.

При необходимости администратор домена может установить данную службу удаленно на необходимое количество компьютеров.

2.2.3 Взаимодействие между сервером и клиентами

Клиент с периодичностью в 2 секунды опрашивает сервер с целью получения подзадач на выполнение.

Клиент отправляет серверу:

- HashId – идентификатор клиента
- IsWaitTask – признак, ожидает клиент задание или выполняет его
- IsSuccess – признак, нашел ли клиент в результате выполнения подзадачи пароль
- TaskResult – пароль, если он был найден в выполненной подзадаче
- LastIdTaskPartDone – идентификатор предыдущей выполненной подзадачи

Сервер отправляет клиенту:

- MaxPasswLen – максимальная заданная длина пароля
- Alphabet – алфавит для перебора
- TaskHash – хэш-значение
- OrderLength – длины пароля, которые необходимо проверить
- OrderFirstSym – порядок символа алфавита заданной длины пароля, с которого начнется перебор
- OrderSymbolsFromFirstToCheck – количество символов алфавита, начиная с заданного, нужно задействовать в переборе
- LastIdTaskPartDone – идентификатор текущей подзадачи

Клиент, приступая к выполнению новой задачи, берет статистическую подзадачу, чтобы сервер мог определить его производительность и присвоить коэффициент. Самый быстрый клиент получает коэффициент 1, другие, с более низкими показателями, получают коэффициенты пропорционально меньше (например, 0.7).

После выполнения статистической подзадачи клиенту выдается подзадача, размер которой определяется в соответствии с его коэффициентом. Ес-

ли клиент по каким-либо причинам не откликается, или выполняет подзадачу длительное время, когда остальные уже справились, его работу забирает клиент с самым высоким коэффициентом. Таким образом, необработанных подзадач не остается.

2.2.4 Состояния работы программы

В процессе работы программы есть несколько основных состояний. Ниже представлена работа при авторизации пользователя (Рисунок 30) и работа с задачами (Рисунок 31).

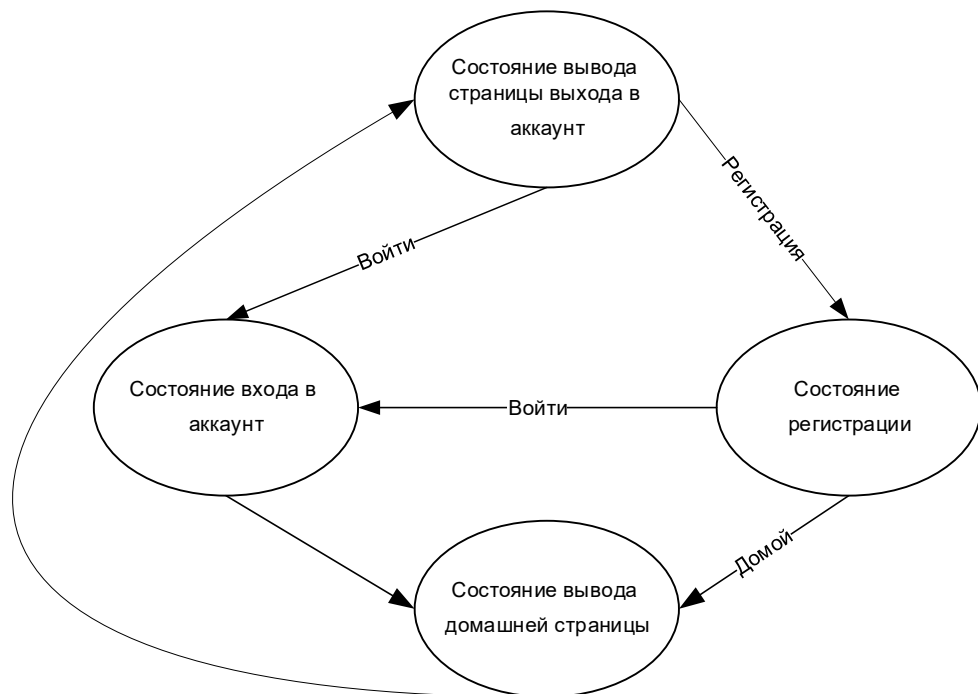


Рисунок 30 — Состояния работы при авторизации на сайте

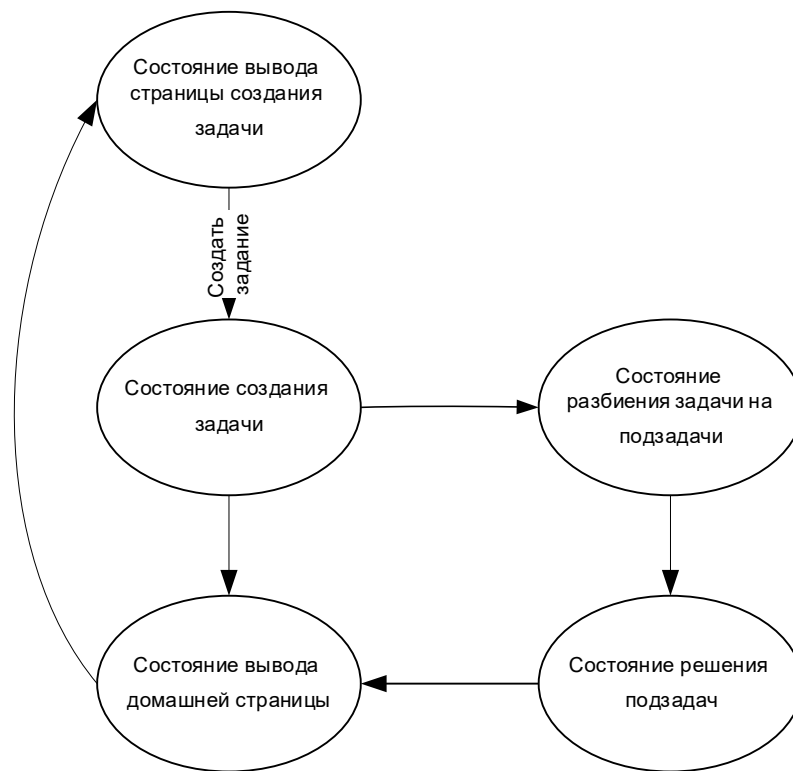


Рисунок 31 — Состояния работы с задачами

3 Результаты работы

3.1 Тестирование

Тестирование проводилось на 3 компьютерах: два с процессором Intel Core i5 и один Intel Core i3.

Для примера возьмем пароли, состоящие из 5 букв английского алфавита нижнего регистра «apple» (Рисунок 33) и «smile» (Рисунок 32). Количество вариантов переборov для нахождения этого пароля составляет 12.356.630.

По результатам вычислений видно, насколько ускоряется процесс поиска с привлечением новых клиентов.

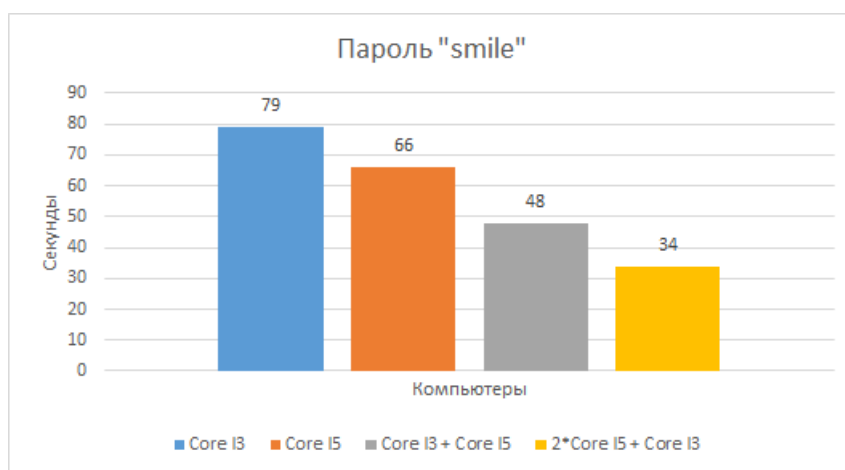


Рисунок 32 — Статистика расчета пароля «smile»

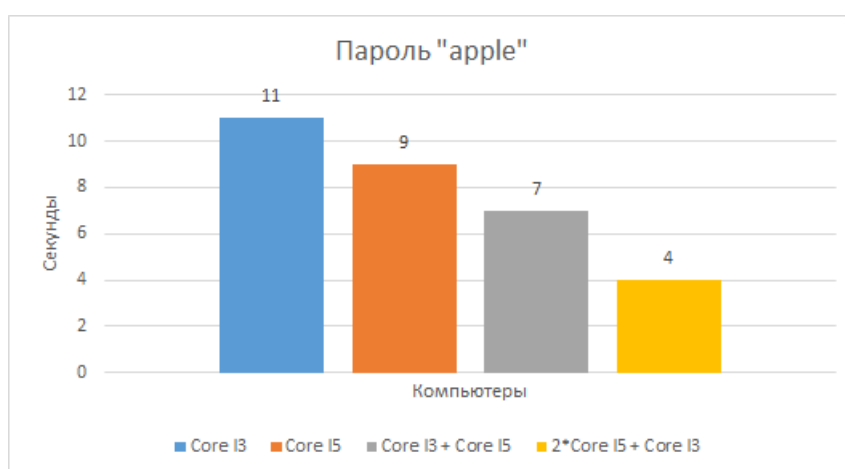


Рисунок 33 — Статистика расчета пароля «apple»

Для вычисления пароля «Apple» выполняется перебор английских символов верхнего и нижнего регистра. Вариантов перебора стало гораздо больше — 19.778.064.644. Ускорение вычислений стало нагляднее (Рисунок 34).

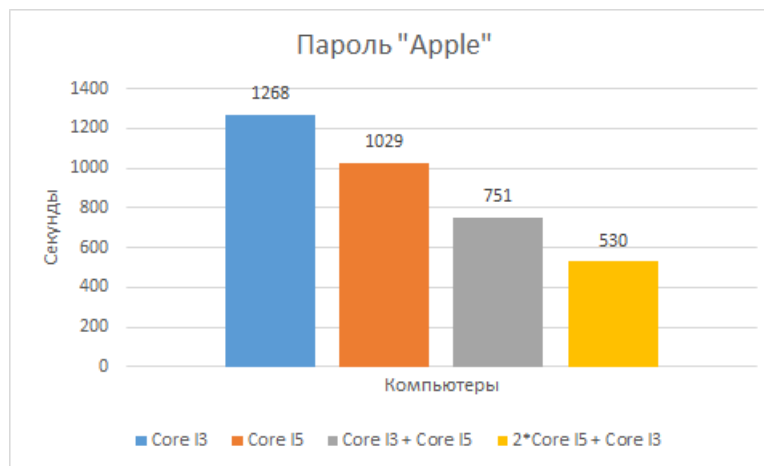


Рисунок 34 — Статистика расчета пароля «Apple»

3.2 Руководство пользователя

Для начала работы Вам необходимо зарегистрироваться.

- Перейти во вкладку **СКАЧАТЬ** и забрать службу-клиент.
- После скачивания Вам необходимо запустить установку службы-клиента, дважды нажав на **InstallClient.msi**. После установки службы, появится файл **YourHash.txt**.
 - Далее зайти во вкладку **РЕГИСТРАЦИЯ** и заполнить личные данные. В поле «Хэш-идентификатор клиента» требуется записать содержимое файла **YourHash.txt**.

После регистрации можно создавать свои задачи, а также помогать другим пользователям решить их.

Заключение

В ходе выполнения дипломной работы была разработана Многопользовательская клиент-серверная Web-система для управления распределенными вычислениями. Благодаря алгоритму распределения подзадач в зависимости от производительности клиента, программа быстро осуществляет поиск пароля по его хэш значению.

В процессе работы над дипломным проектом были изучены и применены технологии Django, SQLite, CryptoAPI.

Поскольку у программы решаемый круг задач ограничен, есть пути ее доработки. В дальнейшем функционал программы можно расширить.

К примеру, если помимо MD5 использовать для вычислений другие алгоритмы хеширования, продукт будет более универсальным. Также можно осуществить подбор пароля по маске, распаковку запароленных архивов.

Для сайта можно добавить платежную систему и, в зависимости от коэффициента клиента, распределить денежное вознаграждение за предоставление вычислительных мощностей.

Список использованных источников

1. Г. Шилдг Полный справочник по C# — изд. Вильнюс, 2004г
2. Г.Р. Эндрюс Основы многопоточного, параллельного и распределенного программирования — изд. Вильнюс, 2003г
3. Джефффри Рихтер CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. — изд. Питер, 2012г
4. Криптография. Базовые знания о шифровании. [Электронный ресурс], URL: <http://www.furfur.me/furfur/culture/culture/166567-kriptografiya> (дата обращения: 15.05.2017).
5. Распределенные вычисления на платформе .Net. [Электронный ресурс], URL: <http://habrahabr.ru/post/97292/> (дата обращения: 10.05.2017).
6. Токарев А.Н. Стратегия размещения подзадач в распределенных вычислительных системах кластерно-метакомпьютерного типа — Пенза, 2005г
7. В.В.Корнеев Параллельные вычислительные системы — изд. Нолидж, 2009г
8. Самоучитель Python. [Электронный ресурс], URL: <https://pythonworld.ru/samouchitel-python> (дата обращения: 28.04.2017).
9. Django. [Электронный ресурс], URL: <https://djbook.ru/> (дата обращения: 5.05.2017).

ИСХОДНЫЙ КОД ПРОГРАММЫ

Приложение А

Сервер

Файл models.py

```
from django.contrib.auth.models import User
from django.db import models
from datetime import datetime, timedelta

# Полный список того, что мы знаем о клиенте
class CClients(models.Model):
    # псевдоним
    Login = models.CharField(max_length=256)

    # пароль
    Password = models.CharField(max_length=256)

    # почта
    Email = models.CharField(max_length=256)

    # IP адрес
    IPAddress = models.CharField(max_length=256)

    # технические параметры компьютера клиента
    TechParams = models.CharField(max_length=256)

    # имя клиента
    Name = models.CharField(max_length=256)

    # фамилия клиента
    LastName = models.CharField(max_length=256)

    # Идентификатор клиента по hash
    HashId = models.CharField(max_length=256, default='')

    def __str__(self):
        return (self)

class CTasks(models.Model):
    # исходный хэш
    TaskHash = models.CharField(max_length=256)

    # расшифрованный пароль
    Result = models.CharField(max_length=256)

    # есть ли русские символы нижнего регистра в пароле
    Rusdown = models.BooleanField(default=False)

    # есть ли русские символы верхнего регистра в пароле
    Rusup = models.BooleanField(default=False)
```

```

# есть ли английские символы нижнего регистра в пароле
Engdown = models.BooleanField(default=False)

# есть ли английские символы верхнего регистра в пароле
Engup = models.BooleanField(default=False)

# есть ли цифры в пароле
Nums = models.BooleanField(default=False)

# завершена ли задача 0 - не тронута 1 - начата 2 завершена
TaskDoneProgress = models.IntegerField(default=0)

# количество клиентов заявленных изначально
NumbClients = models.IntegerField(default=0)

# максимальная длина пароля
MaxPasswLen = models.IntegerField(default=0)

# время и дата начала расчета
BeginTaskCounting = models.DateTimeField(null=True,
blank=True)

# время и дата конца расчета
EndTaskCounting = models.DateTimeField(null=True,
blank=True)

ClientMade = models.ForeignKey(CClients, null=True,
default=None)

# Время выполнения задачи
TimeDoneInSeconds = models.IntegerField(default=0)

def __str__(self):
    return (self)

class CTaskParts(models.Model):
    # для определения коэффициента или реальное задание
    IsItStatistic = models.BooleanField(default=False)

    # длина пароля из которой мы считаем символы от буквы до
    буквы
    OrderFirstLastSymPassLen = models.IntegerField(default=None)

    # порядок алфавита первого символа пароля
    OrderFirstSym = models.IntegerField(default=None)

    # Сколько порядков символов мы должна проверить от
    OrderFirstSym
    OrderSymbolsFromFirstToCheck =
models.IntegerField(default=None)

```

```

    # длины паролей на данном отрезке задания
    OrderLength = models.CharField(max_length=256) # например
5_4_3_2_1

    # текущий статус отрезка заданий
    CurrentStatus = models.IntegerField(default=0) # 0 - не
обрабатывался 1 - обрабатывается 2 - обработан

    # время и дата начала расчета
    BeginTaskPartCounting = models.DateTimeField(null=True,
blank=True)

    # время и дата конца расчета
    EndTaskPartCounting = models.DateTimeField(null=True,
blank=True)

    # Время которое заняла данная подзадача
    DeltaTimeInSeconds = models.IntegerField(default=0)

    # флаг лучшее ли время было в рамках данной задачи
    IsBestTimeDoneInSeconds = models.IntegerField(default=0)

    ClientMade = models.ForeignKey(CClients, null=True,
default=None)

    FullTask = models.ForeignKey(CTasks, default=None)

    def __str__(self):
        return (self)

class CClientsParams(models.Model):
    Client = models.OneToOneField(CClients, primary_key=True)

    # признак входа клиента в аккаунт
    IsClientInAccount = models.BooleanField(default=False)

    # время и дата последнего входа
    LastTimeEnter = models.DateTimeField(null=True, blank=True)

    # коэффициент клиента по производительности
    ClKoeff = models.FloatField(default=0.0)

    # Процент участия в заданиях
    Procent = models.IntegerField(default=0)

    def __str__(self):
        return (self)

```

Файл views.py

```
from django.shortcuts import render
from django.template import loader
from django.http import HttpResponse
from django.shortcuts import render_to_response, redirect
from django.contrib import auth
from django.contrib.auth.models import User

from WebSite.models import CClients, CTaskParts, CClientsParams
from WebSite.models import CTasks

from django.core.mail import send_mail
from django.conf import settings
from django.core.context_processors import csrf
import hashlib
import time
import threading
from datetime import datetime, timedelta, timezone
import http.client
import socket

import sys
from PyQt5 import QtCore

from enum import Enum

EtalonTaskLength = 20000000
CountRUScharsUP = 33
CountEngcharsUp = 26
CountRUScharsDn = 33
CountEngcharsDn = 26
CountNUMBchars = 10

t1 = time.time()

lockMarkReady = threading.Lock()

def __init__(self):
    # noinspection PyArgumentList
    super().__init__()

#####
#####
##### ОТРИСОВКА СТРАНИЦ
#####
#####
```

```

def home_page(request):
    template = loader.get_template('WebSite/Homepage.html')
    first = CClients.objects.all()
    second = CClientsParams.objects.all()
    mylist = zip(first, second)
    return HttpResponse(
        template.render({'itemsCl': mylist,
                        'itemsTasks': CTasks.objects.all(),
                        'username':
auth.get_user(request).username}, request))

def contact_page(request):
    template = loader.get_template('WebSite/Contact.html')
    return HttpResponse(template.render({'username':
auth.get_user(request).username}, request))

def download_page(request):
    template = loader.get_template('WebSite/Download.html')
    return HttpResponse(template.render({'username':
auth.get_user(request).username}, request))

def enter_page(request):
    template =
loader.get_template('WebSite/EnterToAccount.html')
    return HttpResponse(
        template.render({'username':
auth.get_user(request).username}, request))

def registration_page(request):
    return render_to_response('WebSite/Registration.html',
                              {'username':
auth.get_user(request).username,
                              'tryIp':
socket.gethostbyname(socket.getfqdn())})

def queue_page(request):
    template = loader.get_template('WebSite/Queue.html')
    return HttpResponse(
        template.render({'itemsTasks': CTasks.objects.all(),
'username': auth.get_user(request).username}, request))

def newtask_page(request):
    return render_to_response('WebSite/MakeNewTask.html',
{'username': auth.get_user(request).username})

```

```

def get_statistics(request):
    allClients = CClients.objects.all()
    allTaskparts = CTaskParts.objects.filter(CurrentStatus=2)
    if len(allTaskparts) > 0:
        allTaskparts = allTaskparts.exclude(ClientMade_id=None)

    counttasks = len(allTaskparts)

    # забиваем процент участия
    for itemCl in allClients:
        currProc = 0
        for itemTP in allTaskparts:
            if itemCl == itemTP.ClientMade:
                currProc = currProc + 1

        if counttasks == 0:
            pr = 0
        else:
            pr = int(currProc / counttasks * 100)

        cl = CClientsParams.objects.get(Client=itemCl)

        cl.Procent = pr
        cl.save()

    first = CClients.objects.all()
    second = CClientsParams.objects.all()
    mylist = zip(first, second)

    return render_to_response('WebSite/Statistics.html',
{'username': auth.get_user(request).username,
'itemsCl': mylist})

def editaccount_page(request):
    template = loader.get_template('WebSite/EditAccount.html')
    login = 'Anich'
    name = 'Anna0'
    lastName = 'Rzyankina'
    try:
        q = CClients.objects.get(Login=login) # Name=name,
        LastName=lastName
    except:
        return HttpResponse('FAIL PARAMETER: NO SUCH THING');

    return HttpResponse(
        template.render({'itemCl': q, 'username':
auth.get_user(request).username}, request))

def homeroom_page(request):

```

```

user = auth.get_user(request)
template = loader.get_template('WebSite/HomeRoom.html')
login = user.username
try:
    q = CClients.objects.get(Login=login)
except:
    return HttpResponse('FAIL PARAMETER: NO SUCH THING');

TaskOfClient = CTasks.objects.filter(ClientMade=q)

return HttpResponse(
    template.render({'itemCl': q, 'username':
auth.get_user(request).username, 'itemsTasks': TaskOfClient},
    request))

class ALPHABETH(Enum):
    RUSDOWN = 1
    RUSUP = 2
    ENGDOWN = 3
    ENGUP = 4
    NUMS = 5

#####
#####
##### ФУНКЦИИ #####
#####

#####
#####

# вход клиента в систему
def EnterClient(request):
    args = {}
    args.update(csrf(request))
    if request.POST:
        try:
            login = request.POST.get("username", "")
            password = request.POST.get("password", "")
            m = hashlib.md5()
            m.update(bytes(password, "UTF-8"))
        except:
            return HttpResponse("Что-то пошло не так. Попробуйте
еще раз!")

        user = auth.authenticate(username=login,
password=password)
        if user is not None:
            auth.login(request, user)
            p = CClients.objects.get(Login=login)

```

```

        q = CClientsParams.objects.get(Client=p)
        # ставим отметку что вход в систему осуществлен
        q.IsClientInAccount = True
        q.LastTimeEnter = datetime.now(timezone.utc)
        q.save()
        return redirect("/")
    else:
        args['login_error'] = "Пользователь не найден"
        return HttpResponse("Пользователь не найден")
else:
    HttpResponse("request.POST")

# добавление клиента в БД
def ExitOfAccount(request):
    args = {}
    args.update(csrf(request))
    try:
        user = auth.get_user(request)
        p = CClients.objects.get(Login=user.username)
        q = CClientsParams.objects.get(Client=p)
        # ставим отметку что вход в систему осуществлен
        q.IsClientInAccount = False
        #q.LastTimeEnter = datetime.now(timezone.utc)
        q.save()
        auth.logout(request)
        return redirect("/")
    except:
        user = auth.get_user(request)
        auth.logout(request)
        args['login_error'] = "Не удалось осуществить выход из
аккаунта " + user.username
        return render_to_response("WebSite/ExitOfAccount.html",
args)

def AddClientToBD(request):
    args = {}
    args.update(csrf(request))
    if request.POST:
        try:
            login = request.POST.get("login", "")
            password = request.POST.get("password", "")
            password2 = request.POST.get("password2", "")

            if password != password2:
                return HttpResponse('Пароли не совпали')

            email = request.POST.get("email", "")
            ipaddr = request.POST.get("ipaddr", "")
            techparams = request.POST.get("techparams", "")
            name = request.POST.get("name", "")

```

```

        lastname = request.POST.get("lastname", "")
        hashClientId = request.POST.get("hashClientId", "")
    except:
        args['login_error'] = 'Ошибка параметра'
        #return
render_to_response("WebSite/Registration.html", args)
        return render_to_response("Ошибка параметра")

    q = CClients.objects.filter(Login=login)
    if len(q) != 0:
        args['login_error'] = 'Пользователь с таким логином
уже есть'
        #return
render_to_response("WebSite/Registration.html", args)
        return render_to_response("Пользователь с таким
логином уже есть")

    q = CClients.objects.filter(IpAddress=ipaddr)
    if len(q) != 0:
        args['login_error'] = 'Пользователь с IP адресом
уже есть'
        #return
render_to_response("WebSite/Registration.html", args)
        return render_to_response("Пользователь с IP
адресом уже есть")

    try:
        user = User.objects.create_user(login, email,
password)
        user.last_name = lastname
        user.save()
    except:
        args['login_error'] = "Не удалось зарегистрировать
пользователя"
        # return
render_to_response("WebSite/Registration.html", args)
        return render_to_response("Не удалось
зарегистрировать пользователя")

    m = hashlib.md5()
    m.update(bytes(password2, "UTF-8"))

    # добавление нового клиента в БД
    q = CClients.objects.create(Login=login,
Password=m.hexdigest(), Email=email, IpAddress=ipaddr,
TechParams=techparams,
Name=name,
LastName=lastname,
HashId=hashClientId)

    CClientsParams.objects.create(Client=q,
LastTimeEnter=datetime.now(timezone.utc))

```

```

        return redirect("/")
    else:
        args['login_error'] = "Не удалось отправить форму"
        #return render_to_response("WebSite/Registration.html",
args)
        return render_to_response("Не удалось отправить форму")

# Добавление новой задачи
def AddNewTask(request):
    try:
        # забираем параметры с формы
        rusdown = request.POST.get("rusdown", "")
        rusup = request.POST.get("rusup", "")
        engdown = request.POST.get("engdown", "")
        engup = request.POST.get("engup", "")
        nums = request.POST.get("nums", "")
        taskhash = request.POST.get("taskhash_in", "")
        smaxpasswlength = request.POST.get("maxpasslength", "")

    except:
        return HttpResponse('Ошибка параметра');

    maxpasswlength = int(smaxpasswlength, 10)

    # time.strftime("%H:%M:%S")

    NumbOfAlphabethSym = 0

    if rusdown == 'on':
        NumbOfAlphabethSym = CountRUScharsDn +
NumbOfAlphabethSym
    if rusup == 'on':
        NumbOfAlphabethSym = CountRUScharsUP +
NumbOfAlphabethSym
    if engdown == 'on':
        NumbOfAlphabethSym = CountEngcharsDn +
NumbOfAlphabethSym
    if engup == 'on':
        NumbOfAlphabethSym = CountEngcharsUp +
NumbOfAlphabethSym
    if nums == 'on':
        NumbOfAlphabethSym = CountNUMBchars + NumbOfAlphabethSym

    q = CTasks.objects.filter(TaskHash=taskhash)
    if len(q) > 0:
        return HttpResponse('TaskAlreadyMade')

    if NumbOfAlphabethSym == 0:
        return HttpResponse('NoAlphabeth')

```

```

    arrayClientsReady =
CClientsParams.objects.filter(IsClientInAccount=True)

    # количество залогиненных клиентов
    NumbClientsReady = len(arrayClientsReady)

    # создаем запись в БД о новом задании
    CTasks.objects.create(TaskHash=taskhash, Result='0',
Rusdown=rusdown, Rusup=rusup, Engdown=engdown,
                                Engup=engup, Nums=nums,
TaskDoneProgress=0, NumbClients=NumbClientsReady,
                                MaxPasswLen=maxpasswlength)

    currTask = CTasks.objects.get(TaskHash=taskhash)

    user = auth.get_user(request)
    q = CClients.objects.get(Login=user.username)
    currTask.ClientMade = q
    currTask.save()

    i = 1
    FullNumbOfCombs = 0

    while i <= maxpasswlength:
        FullNumbOfCombs = NumbOfAlphabethSym ** i +
FullNumbOfCombs
        i = i + 1

    tempEtalonTaskLength = EtalonTaskLength / NumbOfAlphabethSym

    # сохраняем новый эталон чтобы он делился на размер алфавита
    tempEtalonTaskLength = NumbOfAlphabethSym *
tempEtalonTaskLength

    # Количество переборов для статистики X / N
    volumeForStatistic = FullNumbOfCombs / NumbClientsReady

    # Размер одной задачи
    taskValue = volumeForStatistic / NumbClientsReady

    # считаем объем статистической задачи
    if taskValue > tempEtalonTaskLength:
        taskValue = tempEtalonTaskLength
    else:
        # должна быть кратна алфавиту
        taskValue = int(taskValue - taskValue %
NumbOfAlphabethSym)

#####
    # Случай когда все задания одному клиенту

```

```
#####
# если вариантов меньше чем клиентов или статистическое
значение слишком мало или клиент один
if FullNumOfCombs < NumbClientsReady or taskValue < 1 or
NumbClientsReady == 1:
    # задаем в списке numBuffer все длины пароля в
нисходящем порядке
    numBuffer = []
    i = maxpasswlength

    while i > 0:
        list.append(numBuffer, i)
        # numBuffer[maxpasswlength - i] = i
        i = i - 1
        strNumBuffer = ""

        strNumBuffer = "-".join(str(x) for x in numBuffer)

        CTaskParts.objects.create(IsItStatistic=True,
OrderFirstLastSymPassLen=numBuffer[0],
                                OrderFirstSym=0,

OrderSymbolsFromFirstToCheck=NumOfAlphabethSym,
                                OrderLength=strNumBuffer,
CurrentStatus=0, ClientMade=None, FullTask=currTask)
        template =
loader.get_template('WebSite/Homepage.html')

        first = CClients.objects.all()
        second = CClientsParams.objects.all()
        mylist = zip(first, second)

        return HttpResponse(
            template.render({'itemsCl': mylist,
'itemsTasks': CTasks.objects.all()}, request))

#####
# Случай когда несколько клиентов

#####

# длина пароля с которой мы начинаем давать статистические
задачи

startIndex = maxpasswlength - 1
orderSymbolsFromFirstToCheck = 1
i = 0
while i < maxpasswlength:
    i = i + 1
    if NumOfAlphabethSym ** (i + 1) > taskValue:
        startIndex = i
```

```

        if i > 0:
            orderSymbolsFromFirstToCheck = int(taskValue /
(NumbOfAlphabethSym ** i))
            break

# обработанные длины
doneLengthes = []
list.append(doneLengthes, startIndex)

# текущий индекс алфавита
alphabethIndex = 0

FullNumbVariantsDone = 0

CurrNumbVariantsDone = 0
ii = 0
while ii < NumbClientsReady:
    ii = ii + 1
    i = ii
    numBuffer = []
    list.append(numBuffer, doneLengthes[len(doneLengthes) -
1])

    CurrNumbVariantsDone = (NumbOfAlphabethSym **
(numBuffer[0] - 1)) * orderSymbolsFromFirstToCheck

    if ((alphabethIndex + orderSymbolsFromFirstToCheck) >=
NumbOfAlphabethSym):
        PartNumbVariantsDone = (NumbOfAlphabethSym **
            (numBuffer[0] - 1)) *
(NumbOfAlphabethSym - alphabethIndex - 1)
        FullNumbVariantsDone = PartNumbVariantsDone +
FullNumbVariantsDone
        alphabethIndex = NumbOfAlphabethSym - 1 -
orderSymbolsFromFirstToCheck
    else:
        PartNumbVariantsDone = (NumbOfAlphabethSym **
(numBuffer[0] - 1)) * orderSymbolsFromFirstToCheck
        FullNumbVariantsDone = PartNumbVariantsDone +
FullNumbVariantsDone

    if (alphabethIndex + orderSymbolsFromFirstToCheck) >
NumbOfAlphabethSym:
        orderSymbolsFromFirstToCheck = NumbOfAlphabethSym -
alphabethIndex

    strNumBuffer = "-".join(str(x) for x in numBuffer)

# отдаем все варианты одному - делаем одну запись в базу
данных
CTaskParts.objects.create(IsItStatistic=True,

```

```

OrderFirstLastSymPassLen=numBuffer[0],
                                OrderFirstSym=alphabethIndex,

OrderSymbolsFromFirstToCheck=orderSymbolsFromFirstToCheck,
                                OrderLength=strNumBuffer,
CurrentStatus=0, ClientMade=None, FullTask=currTask)

    alphabethIndex = alphabethIndex +
orderSymbolsFromFirstToCheck

    # Статистические задачи определены, определяем оставшиеся
    isItFirst = True

    numBuffer = []
    currLengthPass = startIndex

    while FullNumbVariantsDone < FullNumbOfCombs:
        CurrNumbVariantsDone = 0

        if isItFirst == True:
            isItFirst = False
            numBuffer = []
            CurrNumbVariantsDone = NumbOfAlphabethSym ** (
                (startIndex - 1)) * (NumbOfAlphabethSym -
alphabethIndex + 1)

            FullNumbVariantsDone = FullNumbVariantsDone +
CurrNumbVariantsDone
            list.append(numBuffer, startIndex)
            j = startIndex - 1
            while j > 0:
                list.append(doneLengthes, j)
                list.append(numBuffer, j)
                j = j - 1

            j = startIndex - 2
            while j > 0:
                CurrNumbVariantsDone = NumbOfAlphabethSym ** j
                FullNumbVariantsDone = FullNumbVariantsDone +
CurrNumbVariantsDone
                j = j - 1

            alphabethIndex = alphabethIndex;

            strNumBuffer = "-".join(str(x) for x in numBuffer)

            # отдаем все варианты одному - делаем одну запись в
базу данных
            CTaskParts.objects.create(IsItStatistic=False,
OrderFirstLastSymPassLen=numBuffer[0],

OrderFirstSym=alphabethIndex,

```

```

OrderSymbolsFromFirstToCheck=(NumbOfAlphabethSym -
alphabethIndex),
                                OrderLength=strNumBuffer,
CurrentStatus=0, FullTask=currTask)

    currLengthPass = currLengthPass + 1
    alphabethIndex = 0

else: # если не первый пришел
    if alphabethIndex >= (NumbOfAlphabethSym - 1):
        alphabethIndex = 0
        currLengthPass = 1 + currLengthPass
        # когда последние индексы алфавита, добавляем в
сделанные
        list.append(doneLengthes, currLengthPass)
        if maxpasswlength < currLengthPass:
            break

    numBuffer = []
    list.append(numBuffer, currLengthPass)

    # сколько букв надо отсылать
    partToCheck = 1
    partToCheck = min(partToCheck, NumbOfAlphabethSym -
alphabethIndex)

    CurrNumbVariantsDone = (NumbOfAlphabethSym **
(currLengthPass - 1)) * partToCheck

    FullNumbVariantsDone += CurrNumbVariantsDone

    strNumBuffer = "-".join(str(x) for x in numBuffer)

    CTaskParts.objects.create(IsItStatistic=False,
OrderFirstLastSymPassLen=numBuffer[0],

OrderFirstSym=alphabethIndex,

OrderSymbolsFromFirstToCheck=partToCheck,
                                OrderLength=strNumBuffer,
CurrentStatus=0, FullTask=currTask)
    alphabethIndex = partToCheck + alphabethIndex

    template = loader.get_template('WebSite/Homepage.html')
    first = CClients.objects.all()
    second = CClientsParams.objects.all()
    mylist = zip(first, second)

    return HttpResponse(
        template.render({'itemsCl': mylist, 'itemsTasks':
CTasks.objects.all()}),

```

```

        'username':
auth.get_user(request).username}, request))

# удаление клиента
def DeleteClient(request):
    try:
        ipAddr = request.POST("ipAddr", "")
    except:
        return HttpResponse('FAIL PARAMETER')
    CClients.objects.filter(IpAddress=ipAddr).delete()
    return HttpResponse('DeleteClient')

# удаление всех заданий

def DeleteAllTasks(request):
    CTasks.objects.all().delete()
    return HttpResponse('DeleteAllTasks')

def timer_action(currClient):
    time.sleep(10)
    currClient.IsClientOnline = False
    return

# обработка сообщения клиента о готовности решать задачи
def MarkReady(request):
    lockMarkReady.acquire()

    NumOfParts = 0
    try:
        try:
            ipClient = request.POST.get("ip", "")
            isWaitTask = request.POST.get("waittask", "") #
ожидает задание или выполняет его
            isSuccess = request.POST.get("success", "") #
успешно ли выполнил задание (нашел ли ответ)
            taskResult = request.POST.get("result", "") #
результат если есть
            lastidtaskpartdone =
int(request.POST.get("lastidtaskpartdone", "")) # результат
если есть
            if isWaitTask == 0:
                return HttpResponse('Working')

        except:
            return HttpResponse('FAIL PARAMETER INIT')

    # выполняет задание и мы его не трогаем

```

```

post_str = ""

# ищем в БД текущего клиента по ip
currClient = CClients.objects.filter(IpAddress=ipClient)
if len(currClient) != 1:
    return HttpResponse('NO SUCH CLIENT IN DATABASE')

myClient = CClients.objects.get(IpAddress=ipClient)

myClientParams =
CClientsParams.objects.get(Client=myClient)

if not myClientParams.IsClientInAccount:
    return HttpResponse('CLIENTOFFLINE')

# проверяем на наличие заданий
TaskToDoRest =
CTasks.objects.exclude(TaskDoneProgress=2)
if len(TaskToDoRest) == 0:
    lastidtaskpartdone = 0
    return HttpResponse('NO TASK')

# задание которое висит в обработке
TaskNotFinishRest =
CTasks.objects.filter(TaskDoneProgress=1)

CurrClientConnect =
CClients.objects.get(id=currClient[0].id)

# текущий клиент для обработки конкретного задания
CurrClientConnectTech =
CClientsParams.objects.get(Client=currClient[0])

# выбираем части, принадлежащие конкретно этому заданию
CurrTaskPart =
CTaskParts.objects.get_queryset().filter(FullTask=TaskToDoRest[0]
])

ReadyTask = CTasks.objects.get(id=TaskToDoRest[0].id)

# если только получили задачу, выставаем флаг что она
начала обрабатываться
if ReadyTask.TaskDoneProgress == 0:
    try:
        ReadyTask.BeginTaskCounting =
datetime.now(timezone.utc)
        ReadyTask.TaskDoneProgress = 1

        ReadyTask.save()
        myClientParams.objects.all().update(ClKcoeff=0.0)
    except:
        return HttpResponse('ERR')

```

```

        # если посчитали успешно
        if isSuccess == '1':
            TaskPartDone =
CTaskParts.objects.get(id=lastidtaskpartdone)
            TaskPartDone.EndTaskPartCounting =
datetime.now(timezone.utc)
            TaskPartDone.save()

            TaskDoneRest =
CTasks.objects.get(id=TaskNotFinishRest[0].id)
            TaskDoneRest.EndTaskCounting =
datetime.now(timezone.utc)
            TaskDoneRest.Result = taskResult
            TaskDoneRest.TaskDoneProgress = 2

            DeltaTimeEnd =
datetime.datetimeuple(TaskDoneRest.EndTaskCounting)
            DeltaTimeEndInSec = DeltaTimeEnd.tm_hour * 60 * 60 +
DeltaTimeEnd.tm_min * 60 + DeltaTimeEnd.tm_sec
            DeltaTimeBegin =
datetime.datetimeuple(TaskDoneRest.BeginTaskCounting)
            DeltaTimeBeginInSec = DeltaTimeEnd.tm_hour * 60 * 60
+ DeltaTimeBegin.tm_min * 60 + DeltaTimeBegin.tm_sec
            if DeltaTimeEndInSec > DeltaTimeBeginInSec:
                TaskDoneRest.TimeDoneInSeconds =
DeltaTimeEndInSec - DeltaTimeBeginInSec

            lastidtaskpartdone = 0
            TaskDoneRest.save()
            countUpdate = CurrTaskPart.update(CurrentStatus=2)
            return HttpResponse('Done')

IsFirstTaskPartDone =
CTaskParts.objects.filter(FullTask=TaskToDoRest[0]).exclude(
    CurrentStatus=0)

# если первый, берем его коэффициент за основу
if len(IsFirstTaskPartDone) == 1 and lastidtaskpartdone
> 0:
    try:
        CurrClientConnectTech.ClKcoeff = 1.0
        CurrClientConnectTech.save()
        TaskPartDone =
CTaskParts.objects.get(id=lastidtaskpartdone)

        # DeltaTime = EndTaskPartCountingDelta -
BeginTaskPartCountingDelta
        # TaskPartDone.DeltaTime = DeltaTime
        # TaskPartDone.save()
        TaskPartDone.EndTaskPartCounting =
datetime.now(timezone.utc)

```

```

        TaskPartDone.save()
        DeltaTimeEnd =
datetime.datetime.now(TaskPartDone.EndTaskPartCounting)
        DeltaTimeEndInSec = DeltaTimeEnd.tm_hour * 60 *
60 + DeltaTimeEnd.tm_min * 60 + DeltaTimeEnd.tm_sec
        DeltaTimeBegin =
datetime.datetime.now(TaskPartDone.BeginTaskPartCounting)
        DeltaTimeBeginInSec = DeltaTimeBegin.tm_hour * 60
* 60 + DeltaTimeBegin.tm_min * 60 + DeltaTimeBegin.tm_sec

        if DeltaTimeEndInSec > DeltaTimeBeginInSec:
            TaskPartDone.IsBestTimeDoneInSeconds =
DeltaTimeEndInSec - DeltaTimeBeginInSec

        TaskPartDone.DeltaTimeInSeconds =
DeltaTimeEndInSec - DeltaTimeBeginInSec
        TaskPartDone.save()

    except BaseException as e:
        return HttpResponse('ERROROROR' + str(e))

    else:
        if lastidtaskpartdone > 0:
            TaskPartDone =
CTaskParts.objects.get(id=lastidtaskpartdone)
            TaskPartDone.EndTaskPartCounting =
datetime.datetime.now(timezone.utc)
            TaskPartDone.save()

            DeltaTimeEnd =
datetime.datetime.now(TaskPartDone.EndTaskPartCounting)
            DeltaTimeEndInSec = DeltaTimeEnd.tm_hour * 60 *
60 + DeltaTimeEnd.tm_min * 60 + DeltaTimeEnd.tm_sec
            DeltaTimeBegin =
datetime.datetime.now(TaskPartDone.BeginTaskPartCounting)
            DeltaTimeBeginInSec = DeltaTimeBegin.tm_hour * 60
* 60 + DeltaTimeBegin.tm_min * 60 + DeltaTimeBegin.tm_sec
            TaskPartDone.DeltaTimeInSeconds =
DeltaTimeEndInSec - DeltaTimeBeginInSec

            TaskPartDone.save()
            if lastidtaskpartdone > 0 and
CurrClientConnectTech.ClKoeff == 0.0:
                # если не первый считаем коэффициент
                TaskPartToCompare =
CurrTaskPart.filter(CurrentStatus=2).filter(IsItStatistic=True).
filter(
                    FullTask_id=TaskToDoRest[0].id)
                for item in TaskPartToCompare:
                    if item.IsBestTimeDoneInSeconds > 0:
                        try:
                            TaskPartDone =

```

```

CTaskParts.objects.get(id=lastidtaskpartdone)
TaskPartDone.EndTaskPartCounting =
datetime.now(timezone.utc)
TaskPartDone.save()
CurrClientConnectTech.ClKoeff =
TaskPartDone.DeltaTimeInSeconds / item.IsBestTimeDoneInSeconds
# return HttpResponse('ERROROR1:' +
str(CurrFientConnect.ClKoeff))
CurrClientConnectTech.save()
except BaseException as e:
    return HttpResponse('ERROROR1' +
str(e))
break

i = 0
zzz = 0
NumbOfParts = 0
# если уже было тестовое задание, дельше берем не из
ТЕСТОВЫХ а ИЗ ОСНОВНЫХ
# если основных не осталось, добираем тестовые если есть
if CurrClientConnectTech.ClKoeff > 0:
    # есть коэффициент - берем основную задачу
    TaskPartToChange =
CurrTaskPart.filter(CurrentStatus=0).exclude(IsItStatistic=True)
    CurrK = 0
    CurrK = int(CurrClientConnectTech.ClKoeff)
    NumbOfParts = CurrK * 10
    if NumbOfParts == 0:
        NumbOfParts = 1

    TaskPartToChangeNumbsSameInLen =
TaskPartToChange.filter(OrderLength=TaskPartToChange[0].OrderLen
gth)
    if NumbOfParts >
len(TaskPartToChangeNumbsSameInLen):
        NumbOfPart = len(TaskPartToChangeNumbsSameInLen)
        NumbOfParts = NumbOfPart

    zzz = len(TaskPartToChangeNumbsSameInLen)
    FirstTaskPartToChange =
CTaskParts.objects.get(id=TaskPartToChangeNumbsSameInLen[0].id)
    while i < (NumbOfParts - 1):
        i = i + 1
        TaskPartToChangeNumbsSameInLen =
TaskPartToChange.filter(OrderLength=TaskPartToChange[0].OrderLen
gth)
        PartToDelete =
CTaskParts.objects.filter(id=TaskPartToChangeNumbsSameInLen[1].i
d)

FirstTaskPartToChange.OrderSymbolsFromFirstToCheck =
FirstTaskPartToChange.OrderSymbolsFromFirstToCheck + \

```

```

PartToDelete[0].OrderSymbolsFromFirstToCheck
    FirstTaskPartToChange.save()
    PartToDelete[0].delete()

    TaskPartToChange =
CurrTaskPart.filter(CurrentStatus=0).exclude(IsItStatistic=True)
    else:
        # нет коэффициента - забираем статистическую задачу
        TaskPartToChange =
CurrTaskPart.filter(CurrentStatus=0).filter(IsItStatistic=True)

        # если не попали в количество - забираем что осталось
        if len(TaskPartToChange) == 0:
            TaskPartToChange =
CurrTaskPart.filter(CurrentStatus=0)

        # когда нам пришел ответ что мы можем закрывать часть
задачи
        if lastidtaskpartdone > 0:
            TaskPartDone =
CTaskParts.objects.get(id=lastidtaskpartdone)
            TaskPartDone.CurrentStatus = 2
            TaskPartDone.ClientMade = CurrClientConnect
            # TaskPartDone.EndTaskPartCounting =
datetime.now(timezone.utc)
            TaskPartDone.save()

            # если нам пока нечего обрабатывать т.е. провальный
результат
            if len(TaskPartToChange) == 0:
                ReadyTask.Result = 'FAIL'
                ReadyTask.TaskDoneProgress = 2

                ReadyTask.EndTaskCounting =
datetime.now(timezone.utc)

                DeltaTimeEnd =
datetime.datetime.strptime(ReadyTask.EndTaskCounting)
                DeltaTimeEndInSec = DeltaTimeEnd.tm_min * 60 +
DeltaTimeEnd.tm_sec
                DeltaTimeBegin =
datetime.datetime.strptime(ReadyTask.BeginTaskCounting)
                DeltaTimeBeginInSec = DeltaTimeBegin.tm_min * 60 +
DeltaTimeBegin.tm_sec
                if DeltaTimeEndInSec > DeltaTimeBeginInSec:
                    ReadyTask.TimeDoneInSeconds = DeltaTimeEndInSec
- DeltaTimeBeginInSec

                ReadyTask.save()
                return HttpResponse('FAIL' + ' ' +
str(lastidtaskpartdone))

```

```

        CurrTaskPartToChange =
CTaskParts.objects.get(id=TaskPartToChange[0].id)
        CurrTaskPartToChange.CurrentStatus = 1
        CurrTaskPartToChange.BeginTaskPartCounting =
datetime.now(timezone.utc).isoformat()
        CurrTaskPartToChange.ClientMade = CurrClientConnect
        CurrTaskPartToChange.save()

        lastidtaskpartdone = TaskPartToChange[0].id

        currTask = TaskToDoRest

        post_str = str(currTask[0].MaxPasswLen)

        # смотрим клиент онлайн или нет
        try:
            # connectedClient =
CClients.objects.get(IpAddress=ipClient)

            post_str = post_str + "_"
            if currTask[0].Rusdown == True:
                post_str = post_str + "1"
            if currTask[0].Rusup == True:
                post_str = post_str + "3"
            if currTask[0].Engdown == True:
                post_str = post_str + "2"
            if currTask[0].Engup == True:
                post_str = post_str + "4"
            if currTask[0].Nums == True:
                post_str = post_str + "5"

            post_str = post_str + "_"
            post_str = post_str + currTask[0].TaskHash

        except:
            return HttpResponse("Alphabeth_error")

        post_str = post_str + "_"

        post_str = post_str + CurrTaskPartToChange.OrderLength

        post_str = post_str + "_"

        post_str = post_str +
str(CurrTaskPartToChange.OrderFirstSym)

        post_str = post_str + "_"

        post_str = post_str +
str(CurrTaskPartToChange.OrderSymbolsFromFirstToCheck)

```

```

        post_str = post_str + "_"

        post_str = post_str + str(lastidtaskpartdone)

        post_str = post_str + "_"

        post_str = post_str + str(zzz)

        post_str = post_str + "_"

        post_str = post_str + str(NumbOfParts)

        return HttpResponse(post_str)

    finally:
        lockMarkReady.release()    # Изменение данных аккаунта

def EditCurrAccount(request):
    try:
        login = request.POST.get("login_out", "")
        password = request.POST.get("password_in", "")
        email = request.POST.get("email_inout", "")
        ipaddr = request.POST.get("ipaddr_inout", "")
        techparams = request.POST.get("techparams_inout", "")
        name = request.POST.get("name_inout", "")
        lastname = request.POST.get("lastname_inout", "")
    except:
        return HttpResponse('Ошибка параметра')

    q = CClients.objects.get(Login=login)
    # ставим отметку что вход в систему осуществлен
    if password != '':
        q.Password = password
    q.Email = email
    q.IpAddress = ipaddr
    q.TechParams = techparams
    q.Name = name
    q.LastName = lastname
    q.save()

    template = loader.get_template('WebSite/HomeRoom.html')
    login = 'Anich'
    try:
        q = CClients.objects.get(Login=login)
    except:
        return HttpResponse('FAIL PARAMETER: NO SUCH THING');

    return HttpResponse(
        template.render({'itemCl': q}, request))

def ContactToDeveloper(request):

```

```

        message = request.POST.get("mess_send", "")
        user = auth.get_user(request)
        send_mail('MyAllotment ' + user.username, message,
settings.EMAIL_HOST_USER, ['annyra@mail.ru'])
        return render_to_response('WebSite/Contact.html')

```

```

# текущее время
def default_start_time():
    now = datetime.now(timezone.utc)
    return now

```

Файл settings.py

```

import os

import mimetypes

mimetypes.add_type("text/css", ".css", True)

# Build paths inside the project like this:
os.path.join(BASE_DIR, ...)
BASE_DIR =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/1.9/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = '5n0)92y2qi!%&7*tsi888-dp_)egw8wbo_xq_p4-2220yysral'

# SECURITY WARNING: don't run with debug turned on in
production!
DEBUG = True

ALLOWED_HOSTS = [
    '*'
]

EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = "loskutova.anna1994@gmail.com"
EMAIL_HOST_PASSWORD = "89050151516"
EMAIL_USE_TLS = True

SERVER_EMAIL = EMAIL_HOST_USER
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER

```

```

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'WebSite',
]

MIDDLEWARE_CLASSES = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    # 'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.auth.middleware.SessionAuthenticationMiddleware'
    ,
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'SiteAllotment.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, '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',
            ],
        },
    ],
]

WSGI_APPLICATION = 'SiteAllotment.wsgi.application'

# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

```

```

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

CACHES = {
    'default': {
        'BACKEND':
'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
        'TIMEOUT': 'None',
    }
}

# Password validation
# https://docs.djangoproject.com/en/1.9/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarity
Validator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator'
,
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator'
,
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidato
r',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/1.9/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

```

```

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC_URL = "/static/"
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "WebSite/static/css")
]

```

Клиент

Файл Service.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net;
using System.Security.Cryptography;
using System.ServiceProcess;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ClientService
{
    public partial class Service1 : ServiceBase
    {
        public static Thread workThread;
        public static char[] finalRes = null;

        public static List<char> RUScharsUP;
        public static List<char> EngcharsUP;
        public static List<char> RUScharsDn;
        public static List<char> EngcharsDn;
        public static List<char> NUMBchars;
        public static List<char> AlphabethChars;

        public static int MaxNumbOfSymbInPass = 0;

        public static bool IsWaitTask = true;

        public static bool SuccessCounting = false;
    }
}

```

```

public static string ResultCounting = "NO";

public static int LastIdTaskPartDone = 0;

public Service1()
{
    InitializeComponent();
    AlphabethChars = new List<char>();
    RUScharsUP = new List<char>();
    EngcharsUP = new List<char>();
    RUScharsDn = new List<char>();
    EngcharsDn = new List<char>();
    NUMBchars = new List<char>();
    // формирование алфавита
    try
    {
        for (int i = 1040; i < 1072; i++)
        {
            RUScharsUP.Add((char)i);

            //добавляем Ё
            if (i == 1045)
                RUScharsUP.Add((char)1025);
        }

        for (int i = 65; i < 91; i++)
        {
            EngcharsUP.Add((char)i);
        }

        for (int i = 1072; i < 1104; i++)
        {
            RUScharsDn.Add((char)i);

            //добавляем Ё
            if (i == 1077)
                RUScharsUP.Add((char)1105);
        }

        for (int i = 97; i < 123; i++)
        {
            EngcharsDn.Add((char)i);
        }

        for (int i = 48; i < 58; i++)
        {
            NUMBchars.Add((char)i);
        }
    }
    catch (Exception ex)
    {

```

```

        Log(String.Format("MyFirstService exception:
{0}", ex.Message));
    }

}

public static void Log(string message)
{

File.AppendAllText("d:\\Anna\\Diplom\\Srv\\MyService.log",
message + "\n");
}

protected override void OnStart(string[] args)
{

    System.Timers.Timer timer = new
System.Timers.Timer();
    timer.Interval = 2000; // 2 seconds
    timer.Elapsed += new
System.Timers.ElapsedEventHandler(this.WorkConnection);
    timer.Start();
}

private void WorkConnection(object sender,
System.Timers.ElapsedEventArgs e)
{
    //Log("WorkConnection запустился\n");

    if (Convert.ToInt32(IsWaitTask) == 0)
        return;

    try
    {
        TimerCount();
    }
    catch (Exception ex)
    {
        Log("WorkConnection завершился\n");
    }
}

protected override void OnStop()
{
    try
    {
        workThread.Abort();
        Log("MyFirstService остановился\n");
    }
}

```

```

        }
        catch (System.Exception ex)
        {
            Log(String.Format("MyFirstService exception:
{0}", ex.Message));
        }
    }

    private static string POST(string Url, string Data)
    {
        try
        {
            System.Net.WebRequest req =
System.Net.WebRequest.Create(Url);
            Log(Data);
            req.Method = "POST";
            req.Timeout = 100000;
            req.ContentType = "application/x-www-form-
urlencoded";

            byte[] sendData =
Encoding.GetEncoding(1251).GetBytes(Data);
            req.ContentLength = sendData.Length;

            System.IO.Stream sendStream =
req.GetRequestStream();
            sendStream.Write(sendData, 0, sendData.Length);
            sendStream.Close();

            System.Net.WebResponse res = req.GetResponse();
            ;
            string Out = "";
            using (Stream stream = res.GetResponseStream())
            {
                using (StreamReader reader = new
StreamReader(stream))
                {
                    string line = "";
                    while ((line = reader.ReadLine()) !=
null)
                    {
                        Log("Результат: " + line);
                        Out += line;
                    }
                }
            }

            Log("Ответ получен: " + Out + "\n");
        }
    }

```

```

        res.Close();

        return Out;

    }
    catch (System.Exception ex)
    {
        Log(String.Format("POST exception: {0}",
ex.Message));
        return "";
    }
}

public static void TimerCount()
{
    Log("TimerCount\n");
    // Получение имени компьютера.
    String host = System.Net.Dns.GetHostName();
    // Получение ip-адреса.
    IPAddress ip =
    Dns.GetHostByName(host).AddressList[0];
    int num = 0;

    string Answer = "";

    try
    {
        Answer =
        POST("http://127.0.0.1:8000/MarkReady/", "ip=" + ip.ToString() +
"&waittask=" + Convert.ToInt32(IsWaitTask).ToString() +
"&success=" + Convert.ToInt32(SuccessCounting).ToString() +
"&result=" + ResultCounting + "&lastidtaskpartdone=" +
LastIdTaskPartDone);

        // если нам нечего больше обрабатывать или мы
успешно завершили задачу
        if (((SuccessCounting == true) && (Answer ==
"Done")) || (Answer == "SUCCESS") || (Answer == "NO TASK"))
        {
            ResultCounting = "NO";
            SuccessCounting = false;

            return;
        }

        int k = Answer.IndexOf("_");

        // определяем максимальное количество символов в
пароле
        MaxNumbOfSymbInPass =
        Convert.ToInt32(Answer.Substring(0, k), 10);
    }
}

```

```

Answer = Answer.Substring(k + 1);

k = Answer.IndexOf("_");

// определяем алфавит для перебора
string alphabeth = Answer.Substring(0, k);

Answer = Answer.Substring(k + 1);

k = Answer.IndexOf("_");

// определяем хеш
string hash = Answer.Substring(0, k);

Answer = Answer.Substring(k + 1);

k = Answer.IndexOf("_");

string strNumBuffer = Answer.Substring(0, k);

int[] numBuffer = strNumBuffer.Split('-
').Select(x => int.Parse(x)).ToArray(); ;

Answer = Answer.Substring(k + 1);

k = Answer.IndexOf("_");

string OrderFirstSym = Answer.Substring(0, k);

Answer = Answer.Substring(k + 1);

k = Answer.IndexOf("_");

string OrderSymbolsFromFirstToCheck =
Answer.Substring(0, k);

Answer = Answer.Substring(k + 1);
Log(Answer);

k = Answer.IndexOf("_"); ////

LastIdTaskPartDone =
Convert.ToInt32(Answer.Substring(0, k), 10);

if (hash == "")
{
    // продолжаем ждать задание
    IsWaitTask = true;
    return;
}
else

```

```

        {
            // дождались задание и выполняем его
            IsWaitTask = false;
        }

        // определяем ХЭШ
        hash = hash.Replace("-", string.Empty);

        Log("Максимальное число " +
MaxNumbOfSymbInPass.ToString() + "\n");
        Log("Алфавит " + alphabeth + "\n");
        Log("ХЭШ " + hash + "\n");
        //Log(strNumBuffer + " " + OrderFirstSym + " " +
OrderSymbolsFromFirstToCheck + LastIdTaskPartDone + "\n");

        // инициализация алфавита
        MakeNewAlphabeth(alphabeth);
        string str = new
string(AlphabethChars.ToArray());
        Log(str + "\n");

        DoWork(AlphabethChars, numBuffer,
Int32.Parse(OrderFirstSym),
Int32.Parse(OrderSymbolsFromFirstToCheck), 1, hash);

    }
    catch (System.Exception ex)
    {
        Log(String.Format("MyFirstService exception:
{0}", ex.Message));
    }

    Log("End TimerCount\n");
}

private static void MakeNewAlphabeth(string alphabeth)
{
    // чистим предыдущий алфавит
    AlphabethChars.Clear();

    // если есть английские символы
    if (alphabeth.Contains("2"))
    {
        foreach (char a in EngcharsDn)
        {
            AlphabethChars.Add(a);
        }

        if (alphabeth.Contains("4"))
        {

```

```

        foreach (char a in EngcharsUP)
        {
            AlphabethChars.Add(a);
        }
    }

    // если есть русские символы
    if (alphabeth.Contains("1"))
    {
        foreach (char a in RUScharsDn)
        {
            AlphabethChars.Add(a);
        }

        if (alphabeth.Contains("3"))
        {
            foreach (char a in RUScharsUP)
            {
                AlphabethChars.Add(a);
            }
        }
    }

    // если есть цифры
    if (alphabeth.Contains("5"))
    {
        foreach (char a in NUMBchars)
        {
            AlphabethChars.Add(a);
        }
    }
}

/// <summary>
/// Рекурсивный метод перебора
/// </summary>
/// <param name="arAlphabet"> массив символов
полученного алфавита </param>
/// <param name="arBuffer"> массив куда пишется
результат</param>
/// <param name="order"> порядок длины пароля
arBuffer с которого надо начинать считать</param>
/// <param name="hash"> хэш для сравнения</param>
static void
RecursionGenerateCombinationsToFile(List<char> arAlphabet,
char[] arBuffer, int order, string hash)
{
    try
    {

```

```

        // если число рассматриваемых порядков меньше
максимального, генерим рекурсию
        if (order < arBuffer.Length)
            for (int i = 0; i < arAlphabet.Count; i++)
            {
                arBuffer[order] = arAlphabet[i];

                RecursionGenerateCombinationsToFile(arAlphabet, arBuffer, order
+ 1, hash);
            }
        else
        {
            // если уже число порядков дошло до
максимума и мы можем начинать расчет
            string str = "";
            for (int u = 0; u < arBuffer.Length; u++)
                str += arBuffer[u].ToString();

            // тут считаем хэш и сравниваем с пришедшим

            var data = Encoding.Unicode.GetBytes(str);

            // Получаем экземпляр MD5
            var md5 = MD5Cng.Create();

            // Вычисляем хеш
            byte[] result = md5.ComputeHash(data);

            string resultStr =
            BitConverter.ToString(result).Replace("-", string.Empty);

            // освобождаем ресурсы

            md5.Clear();
            //string currRes = new string(arBuffer);

            if (resultStr.Equals(hash))
            {
                SuccessCounting = true;
                string finalRes = new string(arBuffer);
                Log(hash + " " + resultStr);
                ResultCounting = finalRes;
                Log("FIND! " + finalRes.ToString());
                return;
            }
        }
    }
}
catch (System.Exception ex)
{

```

```

        Log(String.Format("
RecursionGenerateCombinationsToFile {0}", ex.Message));
    }

}

/// <summary>
/// обрабатывается массив заданных символов - главная
функция
/// </summary>
/// <param name="arAlphabet"></param> алфавит
/// <param name="numBuffer"> массив сколько значные
пароли надо проверять</param>
/// <param name="orderFirstSym"> порядок в алфавите
первого символа в пароле</param>
/// <param name="order"></param>
/// <param name="hash"></param> хэш
/// <returns></returns>
public static char[] DoWork(List<char> arAlphabet, int[]
numBuffer, int orderFirstSym, int orderLastSym, int order,
string hash)
{
    Log("DoWork begin");
    try
    {

        Thread[] th = new Thread[1];

        // счетчик по количеству созданных нитей
        int SchFotThreads = 0;

        // пробегаем по количеству всех заявленных
вариантов длины пароля
        for (int i = 0; i < numBuffer.Length; i++)
        {

            char[] arBuffer = new char[numBuffer[i]];
            Log("numBuffer i = " + i.ToString());

            // если рассматриваемое кол-во символов
максимальное смотрим на порядок
            if ((i == 0) && (order > 0))
            {
                // рассматриваем заданный отрезок
алфавита (только для первого символа пароля)
                //for (int k = 1; k <= orderLastSym -
orderFirstSym; k++)
                for (int k = 0; k < orderLastSym; k++)
                {
                    try
                    {

```

```

// устанавливаем первый символ
arBuffer[0] =
arAlphabet[orderFirstSym + k];
    }
    catch (System.Exception ex)
    {
        Log("arBuffer " + ex.Message +
orderFirstSym.ToString() + " " + arAlphabet.Count + " " +
k.ToString());
    }

RecursionGenerateCombinationsToFile(arAlphabet, arBuffer, order,
hash);
    }
    order = 0;
}
else
{
    Array.Resize(ref th, th.Length + 1);
    th[SchFotThreads] = new Thread(() => {
RecursionGenerateCombinationsToFile(arAlphabet, arBuffer, order,
hash); });

    th[SchFotThreads].Start();

    SchFotThreads++;
}
}

// ждем завершение потоков
for (int i = 0; i < SchFotThreads; i++)
{
    if (finalRes != null)
    {
        th[i].Abort();
    }
    else
    {
        th[i].Join();
    }
}
IsWaitTask = true;
}
catch (System.Exception ex)
{
    Log("DoWork exception: " + ex.Message);
}
IsWaitTask = true;

```

```
        Log("DoWork end");  
        return finalRes;  
    }  
}  
}
```