# Отчет по Этапу 5: База Данных Интернет-Провайдера (Автоматизация клиентской части и многоклиентское тестирование)

23 мая 2025 г.

## Содержание

# 1 Введение

Данный документ представляет собой отчет о реализации Этапа 5 проекта “База Данных Интернет-Провайдера”. Основной целью данного этапа, согласно `task.pdf`, являлось автоматизация клиентской части, создание клиентов-генераторов для формирования непрерывного потока запросов к базе на основе разных сценариев, а также моделирование и тестирование "независимой"и несогласованной работы нескольких клиентов с одним сервером.

Основные задачи Этапа 5 включали:

- Модификацию клиентского приложения для поддержки параметризованных задержек между запросами.
- Введение идентификации клиентов для анализа их взаимодействия в логах и результатах.
- Улучшение протоколирования времени для трассировки выполнения запросов.
- Разработку файлов сценариев, включающих команды управления задержками.
- Создание инфраструктуры (скриптов) для одновременного запуска нескольких экземпляров клиента.
- Определение методологии анализа результатов многоклиентского тестирования для проверки корректности работы сервера в условиях конкурентного доступа.

Этот этап является развитием предыдущего, где была создана базовая клиент-серверная архитектура, и направлен на проверку ее стабильности и корректности при более реалистичных нагрузках.

# 2 Общая структура проекта

Проект "База Данных Интернет-Провайдера"(соответствует заданию (9) из `task.pdf`) имеет четко выраженную модульную структуру, направленную на разделение ответственности между компонентами и облегчение разработки и поддержки. Ниже представлена детальная иерархия файлов и директорий проекта, основанная на анализе предоставленных файлов и файла `1.md`.

## 2.1 Детальная структура файлов и директорий

```
InternetProviderStage4/                 # Корневая директория проекта
|-- CMakeLists.txt                      # Главный файл конфигурации сборки CMake
|-- README.md                           # Описание Этапа 3 (пример)
|-- 1.md                                # Описание структуры проекта
|-- task.pdf                            # Описание задания
|-- src/                                # Исходный код приложения
|   |-- common_defs.h                   # Общие определения и константы
|   |-- core/                           # Ядро системы
|   |   |-- database.h / .cpp
|   |   |-- provider_record.h / .cpp
|   |   |-- tariff_plan.h / .cpp
|   |   |-- query_parser.h / .cpp
|   |   |-- date.h / .cpp
|   |   `-- ip_address.h / .cpp
|   |-- net/                            # Сетевые компоненты
```

```
|   |   `-- tcp_socket.h / .cpp
|   |-- server/                          # Серверная часть
|   |   |-- server.h / .cpp
|   |   |-- server_main.cpp
|   |   |-- server_command_handler.h / .cpp
|   |   `-- server_config.h / .cpp
|   |-- client/                          # Клиентская часть
|   |   |-- client_main.cpp
|   |   `-- client_processing_logic.h
|   `-- utils/                           # Утилиты
|       |-- logger.h / .cpp
|       |-- file_utils.h / .cpp
|       `-- thread_pool.h / .cpp
|-- tools/                               # Инструменты
|   `-- generator.cpp                    # Генератор тестовых данных
|-- data/                                # Данные по умолчанию/тестовые
|   |-- default_tariff.cfg
|   |-- generated_data.txt
|   `-- comprehensive_test_queries.txt.txt
|-- server_databases/                    # БД сервера
|   `-- initial_data.txt
|-- unit_tests/                          # Модульные тесты
|   |-- CMakeLists.txt
|   |-- test_main.cpp
|   |-- test_database.cpp
|   |-- test_provider_record.cpp
|   |-- test_tariff_plan.cpp
|   |-- test_query_parser.cpp
|   |-- test_date.cpp
|   |-- test_ip_address.cpp
|   |-- test_tcp_socket.cpp
|   |-- test_server.cpp
|   |-- test_server_command_handler.cpp
|   |-- test_server_config.cpp
|   |-- test_logger.cpp
|   |-- test_file_utils.cpp
|   |-- test_thread_pool.cpp
|   |-- test_utils.h
|   `-- test_client_request_processing.cpp
`-- integration_tests/                   # Интеграционные тесты
    |-- run_all_tests.sh
    |-- test_common.sh
    |-- client.log
    |-- server.log
    |-- client_integration_test.log
    |-- server_integration_test.log
    |-- test_server_data_root/
    |   `-- server_databases/
    |       |-- dataset_A_s5.txt
    |       `-- dataset_B_s5.txt
    `-- scenarios/
        |-- scenario1/
        |   |-- commands_s1.txt
        |   |-- expected_db_after_s1.txt
        |   |-- expected_output_s1.txt
        |   |-- initial_data_s1.txt
        |   |-- output_actual_s1.txt
        |   `-- tariff_s1.cfg
        |-- scenario2/                 # ...аналогично...
        |   |-- commands_s2.txt
```

```
|   |-- expected_output_s2.txt
|   |-- initial_data_s2.txt
|   |-- output_actual_s2.txt
|   `-- tariff_s2.cfg
|-- scenario3/                  # ...аналогично...
|   |-- commands_s3.txt
|   |-- expected_output_s3.txt
|   |-- initial_data_s3.txt
|   |-- output_actual_s3.txt
|   `-- tariff_s3.cfg
|-- scenario4/                  # ...аналогично...
|   |-- commands_s4.txt
|   |-- expected_output_s4.txt
|   |-- initial_data_s4.txt
|   |-- output_actual_s4.txt
|   `-- tariff_s4.cfg
`-- scenario5/
    |-- commands_s5.txt
    |-- dataset_A_s5.txt
    |-- expected_dataset_A_after_save_s5.txt
    |-- expected_dataset_B_after_save_s5.txt
    |-- expected_output_s5.txt
    |-- output_actual_s5.txt
    `-- tariff_s5.cfg
```

Такая структура обеспечивает хорошую организацию кода, упрощает его понимание, модификацию и тестирование.

# 3 Реализованный функционал (Этап 5)

## 3.1 Модификации клиентского приложения (`client_main.cpp`)

Клиентское приложение (`database_client`) было доработано для поддержки автоматизированного многоклиентского тестирования и моделирования асинхронного взаимодействия.

### 3.1.1 Идентификация клиента

- Добавлена новая опция командной строки: `-client-id <ID_клиента>`.
- Этот идентификатор используется для:
  - Формирования уникального префикса для сообщений в лог-файле клиента (например, `[ClientMain:client1]`).
  - Автоматического формирования имени файла лога по умолчанию (например, `client1_client.log`), если путь к лог-файлу не указан явно.
  - Автоматического формирования имени выходного файла по умолчанию в пакетном режиме (например, `client1_commands.out.txt`).
  - Включения в строки вывода в пакетном режиме (файл, указанный через `-o`), что позволяет легко сопоставить вывод с конкретным экземпляром клиента при анализе результатов параллельной работы.

### 3.1.2 Механизм задержек

Для имитации реальной работы пользователей и создания асинхронной нагрузки на сервер, в клиент была добавлена возможность выполнения команд задержки, считываемых из файла сценария в пакетном режиме.

- **Команда** `DELAY_MS <миллисекунды>`:
  - При обнаружении этой команды в файле сценария клиент приостанавливает свою работу на указанное количество миллисекунд с помощью `std::this_thread::sleep_for(st`
  - Команда логируется и записывается в выходной файл клиента с указанием выполненной задержки.
  - Отрицательные значения игнорируются (с предупреждением в лог).
- **Команда** `DELAY_RANDOM_MS <мин_мс> <макс_мс>`:
  - Клиент генерирует случайное целое число миллисекунд в указанном диапазоне (включительно) и приостанавливает работу на это время.
  - Используется `std::random_device`, `std::mt19937` и `std::uniform_int_distribution`.
  - Команда, выбранный диапазон и фактическое значение задержки логируются и записываются в выходной файл клиента.
  - Некорректные параметры диапазона (например, `min_ms < 0` или `max_ms < min_ms`) приводят к пропуску команды с предупреждением.
- Эти команды не отправляются на сервер, а обрабатываются локально клиентом.

### 3.1.3 Улучшенное протоколирование времени в выходном файле

Для анализа хронологии событий при многоклиентской работе, в выходной файл пакетного режима (указанный опцией `-o`) теперь записываются временные метки:

- Перед отправкой каждого запроса на сервер: `[%Y-%m-%d %H:%M:%S.мс] [ID_клиента] ЗАПРОС #N: <текст_запроса>`.
- После успешного получения полного ответа от сервера на запрос: `[%Y-%m-%d %H:%M:%S.мс] [ID_клиента] ОТВЕТ ПОЛУЧЕН для запроса #N`.
- При выполнении команд задержки: `[%Y-%m-%d %H:%M:%S.мс] [ID_клиента] CMD: DELAY_MS ...`.

Временные метки генерируются с помощью статической функции `get_current_timestamp_for_output` в `client_main.cpp`, обеспечивающей формат ГГГГ-ММ-ДД ЧЧ:ММ:СС.мс.

## 3.2 Файлы сценариев для Этапа 5

Файлы команд для пакетного режима были расширены для включения команд управления задержками. Это позволяет создавать более сложные и реалистичные сценарии взаимодействия.

Пример структуры файла команд (`commands_clientX.txt`):

```
# Scenario for a specific client
DELAY_MS 150 # Initial delay for this client

ADD FIO "User Name X" IP "10.0.1.X" DATE "DD.MM.YYYY" END
DELAY_RANDOM_MS 100 500

SELECT IP "10.0.1.X" END
DELAY_MS 200

# Further commands...
EXIT
```

Создание разнообразных таких файлов с различными командами и интервалами задержек позволяет моделировать различные профили нагрузки от нескольких клиентов.

## 3.3 Оркестровка многоклиентского тестирования (`run_stage5_tests.sh`)

Для автоматизации запуска и управления многоклиентскими тестами был создан новый shell-скрипт `integration\_tests/run\_stage5\_tests.sh`. Этот скрипт использует общие функции из `integration\_tests/test\_common.sh`.

Ключевые возможности скрипта:

- **Подготовка окружения**: Очищает предыдущие результаты и данные, создает необходимые директории для логов, результатов и временных файлов сценариев. Создает общий файл тарифов для тестового прогона. Генерирует примеры файлов сценариев для нескольких клиентов.
- **Запуск сервера**: Запускает один экземпляр сервера `database_server` с предопределенными параметрами (порт, файл тарифов, директория данных, настройки логирования). PID сервера сохраняется для последующей остановки.
- **Конфигурация клиентов**: Определяется массив конфигураций для нескольких клиентов. Каждая конфигурация включает уникальный ID клиента и путь к его файлу сценария.
- **Одновременный запуск клиентов**: Для каждой конфигурации клиента:
  - Формируются параметры командной строки для `database_client`, включая `-client-id`, путь к файлу сценария, путь к уникальному файлу вывода и путь к уникальному файлу лога.
  - Клиент запускается в фоновом режиме (`&`). PID каждого запущенного клиента сохраняется.
- **Ожидание завершения**: Скрипт ожидает завершения всех запущенных клиентских процессов с помощью команды `wait`. Анализируются коды завершения каждого клиента.
- **Остановка сервера**: После завершения всех клиентов сервер корректно останавливается.
- **Вывод результатов**: Скрипт выводит информацию о статусе завершения каждого клиента, а также пути к файлам логов и результатам для дальнейшего анализа.

Этот скрипт позволяет моделировать ситуацию, описанную в `task.pdf`, где несколько клиентов независимо и асинхронно обращаются к серверу.

### 3.4 Интерпретация серверных задержек

В `task.pdf` упоминается: "Вызовы этих функций [sleep/usleep] вставляются между отправкой отдельных запросов каждого клиента и между отправкой отдельных ответных сообщений сервера."На данном этапе, основное внимание было уделено реализации задержек на стороне клиента, так как это напрямую соответствует задаче "Автоматизация клиентской части". Внесение искусственных задержек на стороне сервера *при отправке ответа* не было реализовано как постоянная функция сервера. Текущая реализация с клиентскими задержками позволяет тестировать асинхронное взаимодействие и реакцию сервера на конкурентные запросы.

## 4 Методология тестирования Этапа 5

Основная цель тестирования на Этапе 5 — проверить корректность работы сервера при одновременном обращении нескольких независимых клиентов и убедиться, что "результаты ответов сервера для каждого отдельного клиента не должны зависеть от количества других клиентов и от сценариев их работы".

- **Сценарии нагрузки**: Используются различные файлы команд для каждого клиента, содержащие разные последовательности операций и разные команды задержек (`DELAY_MS`, `DELAY_RANDOM_MS`). Это создает непредсказуемую, асинхронную нагрузку на сервер.
- **Анализ выходных данных клиентов**:
  - Каждый клиент сохраняет свой диалог с сервером (отправленные запросы с временными метками, полученные ответы сервера) в уникальный файл вывода.
  - Эти файлы анализируются на предмет корректности ответов сервера на запросы конкретного клиента.
  - Важно проверить, что операции одного клиента корректно отражаются в ответах на запросы других клиентов.
  - Сравнение выходных файлов одного и того же клиентского сценария, запущенного изолированно и в составе группы, может помочь оценить влияние других клиентов, хотя полная идентичность вывода (особенно для команд типа `PRINT_ALL`) не всегда является критерием из-за параллельных модификаций.
- **Анализ состояния базы данных**: Если сценарии клиентов включают команды `SAVE`, можно анализировать сохраненные файлы БД для проверки целостности данных.
- **Анализ логов**: Логи сервера и клиентов анализируются на предмет ошибок, предупреждений, порядка обработки запросов и проблем синхронизации. Идентификаторы клиентов и временные метки помогают в этом.
- **Проверка отсутствия взаимоблокировок (Deadlocks)**: Отслеживается по логам и таймаутам.

Критерием успешного прохождения теста является корректное завершение работы всех клиентов, получение логически правильных ответов от сервера и сохранение базой данных целостного состояния.

# 5 Инструкции по запуску тестов Этапа 5

1. **Сборка проекта**: Убедитесь, что сервер (`database_server`) и клиент (`database_client`) скомпилированы с последними изменениями (см. Раздел 6 отчета по Этапу 4, пути к исполняемым файлам: `build/bin/`). 2. **Подготовка сценариев**:

- Создайте директорию `integration\_tests/scenarios\_stage5/`.
- Внутри нее создайте поддиректории для каждого тестового клиента (`client1/`, `client2/` и т.д.).
- В каждую такую поддиректорию поместите файл команд (например, `commands.txt`) с запросами и командами задержек.

3. **Настройка и запуск скрипта `run_stage5_tests.sh`**:

- Убедитесь, что пути к исполняемым файлам и другие параметры в скрипте `integration\_tests/run\_stage5\_tests.sh` (предоставлен ранее) указаны корректно.
- В массиве `client_scenarios_config` укажите идентификаторы клиентов и пути к их файлам сценариев.
- Сделайте скрипт исполняемым: `chmod +x integration_tests/run_stage5_tests.sh`.
- Запустите скрипт из директории `integration_tests/`: `./run_stage5_tests.sh`.

4. **Анализ результатов**:

- Изучите консольный вывод скрипта.
- Проверьте содержимое директории результатов (например, `integration\_tests/stage5\_results/`).
- Изучите лог-файлы сервера и клиентов.
- Проверьте файлы базы данных, если они сохранялись.

# 6 Общая функциональность проекта (обзор)

Помимо нововведений Этапа 5, проект включает в себя всю функциональность, реализованную на предыдущих этапах, которая составляет основу для клиент-серверного взаимодействия:

- **Структуры данных** (`core/`):
  - `Date`: Представление и валидация дат (`date.h`, `date.cpp`).
  - `IPAddress`: Представление и валидация IPv4-адресов (`ip_address.h`, `ip_address.cpp`).
  - `ProviderRecord`: Структура записи об абоненте, включая ФИО, IP, дату и почасовой трафик (`provider_record.h`, `provider_record.cpp`). Включает валидацию данных трафика.
  - `TariffPlan`: Управление почасовыми тарифами на входящий и исходящий трафик, загрузка из файла (`tariff_plan.h`, `tariff_plan.cpp`).
- **База данных** (`Database`) (`database.h`, `database.cpp`):

- Хранение записей `ProviderRecord` в векторе.
- Операции CRUD: добавление, получение по индексу, редактирование, удаление по различным критериям и индексам.
- Поиск записей по имени, IP, дате или комбинации критериев.
- Загрузка (`loadFromFile`) и сохранение (`saveToFile`) базы данных в файл с обработкой ошибок формата.
- Расчет начислений (`calculateChargesForRecord`) на основе данных записи и тарифного плана.

- **Разбор запросов (`QueryParser`)** (`query_parser.h`, `query_parser.cpp`):
  - Преобразование текстовых строк запросов в структурированные объекты `Query` (содержащие `QueryType` и `QueryParameters`).
  - Токенизация запросов с учетом строк в кавычках.
  - Поддержка команд: `ADD`, `SELECT`, `DELETE`, `EDIT`, `CALCULATE_CHARGES`, `PRINT_ALL`, `LOAD`, `SAVE`, `EXIT`, `HELP`.
  - Разбор параметров для каждой команды, включая критерии фильтрации и данные для модификации.
  - Валидация синтаксиса запросов и генерация исключений при ошибках.

- **Сетевое взаимодействие (`TCPSocket`)** (`net/tcp_socket.h`, `net/tcp_socket.cpp`):
  - Кросс-платформенная обертка для TCP-сокетов (Windows Winsock и POSIX Berkeley sockets).
  - Управление жизненным циклом сокета (создание, закрытие, перемещение).
  - Серверные операции: `bindSocket`, `listenSocket`, `acceptSocket`.
  - Клиентские операции: `connectSocket`.
  - Отправка и получение данных, включая реализацию протокола "длина + данные"(`sendAllDataWithLengthPrefix`, `receiveAllDataWithLengthPrefix`) для сообщений переменной длины.
  - Установка опций сокета (неблокирующий режим, таймауты).
  - Обработка ошибок сокетов и логирование.

- **Утилиты (`utils/`)**:
  - `Logger`: Статический класс для логирования сообщений различных уровней в консоль и/или файл (`logger.h`, `logger.cpp`).
  - `FileUtils`: Утилиты для работы с файловой системой (C++17 `<filesystem>`), включая определение путей к проекту/данным и безопасное формирование путей к файлам на сервере (`file_utils.h`, `file_utils.cpp`).
  - `ThreadPool`: Класс для управления пулом рабочих потоков для асинхронного выполнения задач (`thread_pool.h`, `thread_pool.cpp`).

- **Общие определения (`common_defs.h`)**:
  - Подключение стандартных библиотек C++.
  - Глобальные константы проекта (`HOURS_IN_DAY`, `MAX_MESSAGE_PAYLOAD_SIZE` и др.).

– Константы для сетевого протокола ответа сервера (коды статуса, типы полезной нагрузки, параметры чанкования, ключи заголовков).

Вся эта функциональность используется и проверяется в рамках Этапа 5 при многоклиентском взаимодействии.

# 7 Заключение по Этапу 5

На Этапе 5 была успешно реализована автоматизация клиентской части для проведения многоклиентского тестирования. Клиентское приложение было доработано для поддержки идентификаторов экземпляров и команд управления задержками, что позволяет моделировать асинхронное и несогласованное взаимодействие нескольких клиентов с сервером. Создан новый скрипт для оркестровки таких тестов, включая запуск сервера и параллельный запуск нескольких клиентов с различными сценариями.

Эта доработка позволяет более тщательно проверить надежность сервера, корректность механизмов синхронизации доступа к данным и общую производительность системы в условиях, приближенных к реальным. Полученные инструменты и методология тестирования закладывают основу для дальнейшей оценки масштабируемости и стабильности серверного приложения. Задачи, поставленные в `task.pdf` для данного этапа, в основном выполнены.