# Цели тестирования

|  |  |  |
| --- | --- | --- |
| Цель тестирования | Вид тестирования | DOD (Definition of done) |
| Определить максимальную производительность системы, оценить запас производительности ИС на ПРОД, определить узкое место системы | Поиск максимальной производительности | В ходе теста поиска максимума достигнута точка деградации;  На протяжении интервала стабильной подачи нагрузки теста подтверждения система соответствовала требованиям;  В выводах протокола НТ зафиксирована МП как в % от профиля нагрузки, так и в % запаса производительности относительно промышленной нагрузки; также указывается ограничитель производительности, т.е. причина отказа системы при достижении точки деградации (узкое место) |
| Подтвердить надёжность системы | Тест надёжности | На протяжении интервала стабильной подачи нагрузки система соответствовала требованиям, отсутствуют тренды на ухудшение показателей производительности |
| Подтвердить способность системы автоматически восстанавливаться в стрессовых ситуациях | Стресс-тестирование (все сценарии) | Все сценарии стресс-тестирования пройдены с соответствием критериев успешности для каждого сценария |

# Ограничения тестирования

|  |  |  |
| --- | --- | --- |
| Ограничение | Влияние | Срок устранения / задача на устранение |
| Отсутствие прикладного мониторинга | Невозможность диагностики проблем производительности на уровне внутренних механизмов | 2Q2025 |
| В качестве тестовых данных используются синтетически сгенерированные данные | Возможное отклонение вариативности тестовых данных от ПРОМ среды | 2Q2025 |

# Объект тестирования

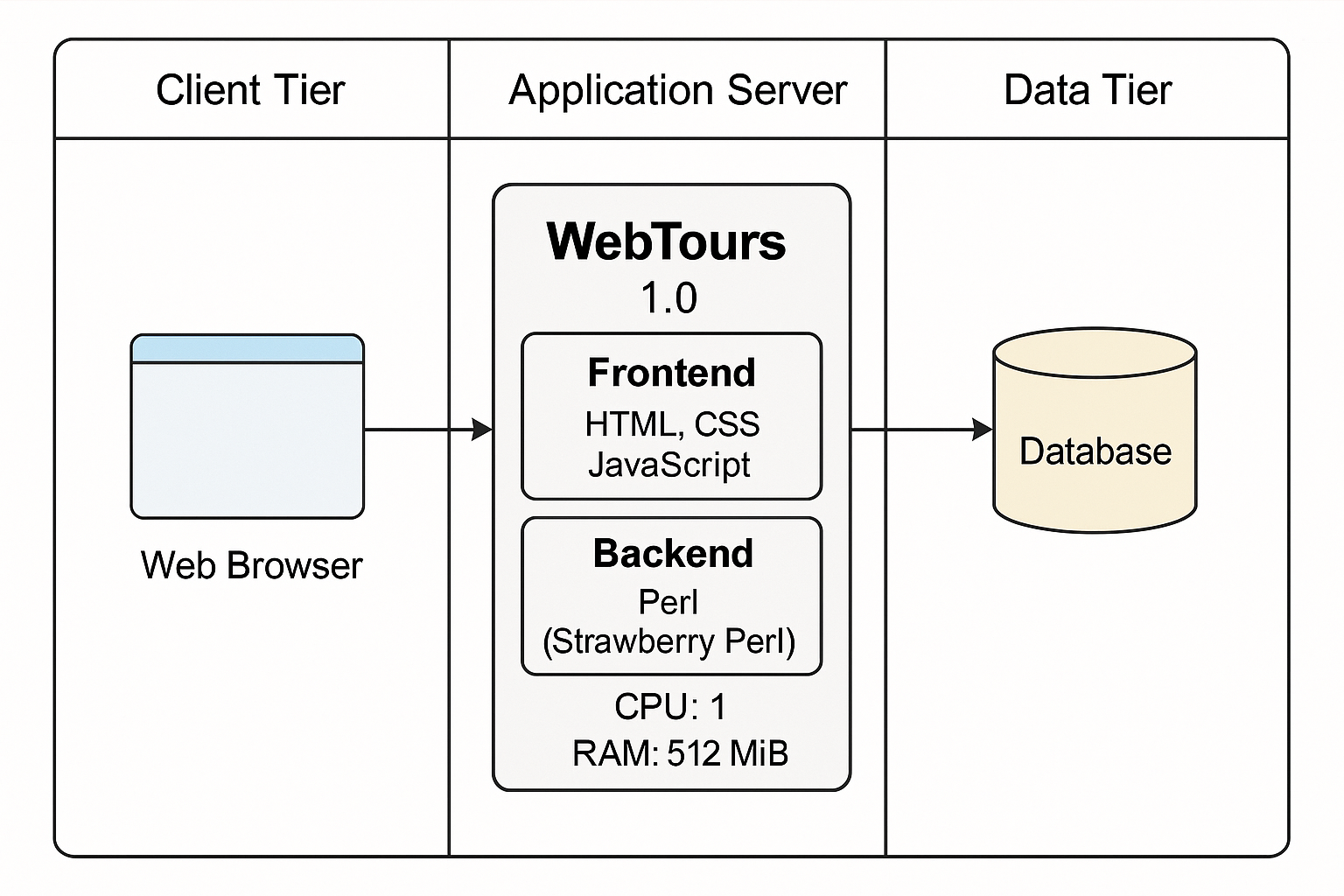
WebTours 1.0 — это учебное веб-приложение, разработанное для демонстрации возможностей инструментов нагрузочного тестирования, таких как JMeter/Gatling. Оно представляет собой онлайн-агентство путешествий, предоставляющее пользователям функциональность для бронирования авиаперелетов.

## Компоненты объекта тестирования

* **Фронтенд (Frontend):** Интерфейс пользователя реализован с использованием стандартных веб-технологий: HTML, CSS и JavaScript. Эти технологии обеспечивают отображение веб-страниц и взаимодействие с пользователем.​
* **Бэкенд (Backend):** Серверная часть приложения отвечает за обработку бизнес-логики и взаимодействие с базой данных. WebTours использует язык программирования Perl, а именно дистрибутив Strawberry Perl, который позволяет запускать Perl-скрипты на Windows-платформах.​
* **База данных (Database):** Для хранения данных о пользователях и бронированиях используется простая файловая система в виде текстовых файлов.​

## Архитектура стенда НТ

|  |  |
| --- | --- |
| Ресурс | Значение |
| CPU | 1 |
| RAM | 512 MiB |



=================================================================

🌐 WebTours 1.0: Трехзвенная архитектура (Mind-Blowing Edition) 🌐

=================================================================

🔹 Клиентский уровень (Client Tier) 🔹

+-----------------------------+ +-----------------------------+

| Веб-браузер 1 (HTML, CSS) | | Веб-браузер 2 (JavaScript) |

| 🌍 Chrome/Firefox | | 🌍 Safari/Edge |

+-----------------------------+ +-----------------------------+

| HTTP/HTTPS Запросы | | HTTP/HTTPS Запросы |

+---------------------+---------------+---------------------+

| 🔗 Поток данных 🔗 |



🔹 Сервер приложений (Application Server) 🔹

+-----------------------------+

| ⚙️ Perl (Strawberry Perl) |

| 💻 Бизнес-логика: |

| - Поиск рейсов |

| - Бронирование |

| 📊 Ресурсы: |

| - CPU: 1 |

| - RAM: 512 МБ |

+-----------------------------+

| Чтение/Запись 📝 |

+---------------------+

| 💾 Данные |



🔹 Уровень данных (Data Tier) 🔹

+-----------------------------+

| 📂 Текстовые файлы: |

| - users.txt |

| - bookings.txt |

| - flights.txt |

| 💿 Простая файловая система |

+-----------------------------+

=================================================================

🔥 Потоки данных: HTTP-запросы ➡️ Обработка ➡️ Хранение 🔥

=================================================================

WebTours следует классической трехзвенной архитектуре, состоящей из следующих уровней:​

1. **Клиентский уровень (Client Tier):** Включает веб-браузеры пользователей, которые отправляют HTTP-запросы и отображают полученные страницы.​
2. **Сервер приложений (Application Server):** Обрабатывает запросы от клиентов, выполняет бизнес-логику и взаимодействует с базой данных. Реализован с использованием Perl-скриптов, работающих на сервере.​
3. **Уровень данных (Data Tier):** Состоит из текстовых файлов, используемых для хранения данных приложения.

## Требования к производительности

### Критерии оценки работоспособности системы

Система должна иметь запас 200% от ПРОМа (100% + 200%).

## Мониторинг

|  |  |  |
| --- | --- | --- |
| Назначение мониторинга | Инструмент | Ссылка |
| Мониторинг метрик инструмента НТ (JMeter/Gatling) | JMeter/Gatling (с подключением к Grafana) | http://localhost:3000 (Grafana)  <http://localhost:8888> (Gatling) |
| Мониторинг утилизации ресурсов контейнеров (Docker) | Grafana (стандартный дашборд для Docker) | http://localhost:3000 (Grafana) |
| Сбор метрик контейнеров (cAdvisor) | cAdvisor | http://localhost:8080 |
| Хранение и обработка метрик (Prometheus) | Prometheus | http://localhost:9090 |
| Визуализация метрик (Grafana) | Grafana | http://localhost:3000 |

### Дополнительные средства мониторинга

# Моделирование нагрузки

## Виды тестирования

### Поиск максимальной производительности

 Тест проводится с **пошаговым увеличением нагрузки** для выявления предельного уровня производительности системы.

 Каждая ступень нагрузки увеличивается на **100% от базового профиля** предыдущей ступени.

 **Длительность стабильного отрезка ступени** – **22 минуты** (включая 2 минуты для стабилизации и 20 минут для анализа).

 Завершается тест при достижении точки деградации системы.

### Подтверждение максимальной производительности

### Проводится на **100% от уровня максимальной производительности**, выявленного в тесте поиска.

* **Длительность стабильного отрезка ступени** – **1 час 2 минуты** (включая 2 минуты для стабилизации и 20 минут для анализа).

 **Цель:** Подтвердить работоспособность системы при длительной нагрузке без деградации ключевых метрик.

### Тест стабильности

 Проводится на **70% от уровня подтвержденного максимума производительности**.

 **Цель:** Оценка стабильности работы системы в длительном интервале времени.

 **Критерии успешности:** отсутствие ухудшения производительности, стабильность использования ресурсов.

### Стресс-тест Пиковая производительность

 **Метод нагрузки:** Резкое повышение нагрузки до уровня, превышающего **пиковую производительность системы**.

 **Длительность удержания пиковой нагрузки:** **5 минут после фиксации деградации**.

 **Фаза спада нагрузки:** Плавное снижение нагрузки по ступеням с шагом **-25% от пиковой нагрузки**.

 **Длительность каждой ступени в фазе снижения:** **2 минуты**.

 **Цель:** Проверка поведения системы в условиях экстремальной нагрузки и восстановления после перегрузки.

### Стресс-тест объёмное тестирование

 **Метод нагрузки:** Для **30% пользователей** **увеличивается объем данных в 10 раз** (увеличение количества купленных билетов).

### Проводится на **100% от уровня максимальной производительности**, выявленного в тесте поиска.

* **Длительность стабильного отрезка ступени** – **1 час 2 минуты** (включая 2 минуты для стабилизации и 20 минут для анализа).

 **Цель:** Оценка влияния увеличенного объема данных на производительность и корректность обработки операций.

 **Критерии успешности:** Отсутствие значительной деградации времени отклика и корректное выполнение операций.

## Профиль нагрузки

Профиль актуализирован на данных статистики промышленной среды за 14.04.2025, пиковый час с 20:00 до 21:00

|  |  |  |  |
| --- | --- | --- | --- |
| Статистика с ПРОДа | | Профиль | |
| Название запроса | Интенсивность по статистике запросов / час | Расчетная интенсивность запросов / час | % Соотвествия расчетанной интенсивности статистики |
| Главная Welcome страница | 520 | 522 | 0% |
| Вход в систему | 422 | 422 | 0% |
| Переход на страницу поиска билетов | 305 | 321 | 5% |
| Заполнение полей для поиска билета | 282 | 274 | -3% |
| Выбор рейса из найденных | 270 | 274 | 2% |
| Оплата билета | 175 | 171 | -2% |
| Просмотр квитанций | 280 | 291 | 4% |
| Отмена бронирования | 73 | 73 | 0% |
| Выход из системы | 326 | 321 | -1% |
| Перход на страницу регистрации | 97 | 100 | 3% |
| Заполнение полей регистарции | 97 | 100 | 3% |
| Переход на следуюущий эран после регистарции | 97 | 100 | 3% |
| **Итого** | 2944 | 2970 | 1% |

## Критерии корректности проведенных тестов

|  |  |  |
| --- | --- | --- |
| Название критерия | Метрика | Максимальное значение |
| Время отклика транзакций | 90 percentile response time | 2,5 сек |
| Количество ошибок по транзакциям | Summary Report fail / pass + fail | 5% |
| Отклонение фактической нагрузки от профиля | Summary Report Errort % | - |
| Утилизация CPU контейнера WebTours | container\_cpu\_usage\_seconds\_total | 90% |
| Утилизация оперативной памяти контейнера WebTours | container\_memory\_rss | 90% |
| Утилизация диска WebTours | Disk IO | 90% |

# Наполнение БД

5.1. Перенос данных с промышленной среды

5.1.1. Требования к объемам данных

* Пользователи: 50 карточек с полями: username, password, firstName, lastName, address1, address2.
* Билеты: 250 билетов (50 пользователей × 5 билетов) с полями: flightID, outboundFlight, returnFlight (опционально), привязка к username.
* Данные переносятся из промышленной среды с сохранением структуры и связей.

5.2. Методология генерации данных

5.2.1. Требования к генерации данных

* Уникальность: Уникальные username и flightID.
* Реалистичность: Правдоподобные имена (firstName, lastName), адреса США (address1, address2), рейсы в формате "<Авиакомпания>;<Номер>;<YYYY-MM-DD>".
* Объем: 50 пользователей, 250 билетов (5 на пользователя).
* Безопасность: Случайные пароли (12+ символов, буквы+цифры).
* Консистентность: Билеты привязаны к username, returnFlight заполнен для roundtrip.
* Масштабируемость: Автоматизация, настройка объемов через параметры.
* Ошибки: Логирование ошибок, пропуск некорректных данных.

5.2.2. Способ генерации данных

Генерация автоматизирована через JMeter с Groovy-скриптами:

* Пользователи:
  + Источник: API randomuser.me для firstName, lastName (например, "José", "García").
  + Логин: username из очищенной комбинации firstName+lastName (например, "josegarcia"), уникальность через HashSet (usedCombinations), максимум 1000 попыток.
  + Пароль: Случайный, 12 символов (например, "Xy7kP9mW2nJ4").
  + Адрес: Случайный address1 из списка улиц (например, "Main St"), address2 из городов (например, "New York, NY 10001").
  + Запрос: application/x-www-form-urlencoded (например, "username=josegarcia&password=Xy7kP9mW2nJ4..."), сохраняется в vars.dynamicRegistering.
  + Логи: Данные пользователя в vars.registeredUserData.
* Билеты:
  + Рейсы: Случайный outboundFlight из vars (например, "Blue Sky Air;123;2025-04-25"), returnFlight для roundtrip.
  + Пассажиры: Имена из users.csv (например, "John Doe"), минимум 6 полей.
  + Оплата: Случайная кредитная карта (16 цифр, "4xxx..."), дата "MM/YY" (например, "04/27"), saveCC ("on"/"").
  + Запрос: multipart/form-data с данными рейсов, пассажиров, оплаты, сохраняется в vars.paymentBody.
  + Привязка: Билеты связаны с username, flightID от сервера (например, "123").
  + Объем: 5 билетов на пользователя (50 × 5 = 250).
* Автоматизация:
  + JMeter: Циклы для 50 пользователей и 5 билетов, параметры для масштабирования (${\_\_P(users, 50)}).
  + Логи: log.info для данных, log.error для ошибок (например, API-сбои, пустой users.csv).
  + Ошибки: Пропуск пустых данных, исключения при сбоях (например, "No users found in users.csv").
* Пример:
  + Пользователь: username="josegarcia", password="Xy7kP9mW2nJ4", firstName="José", lastName="García", address1="Main St", address2="New York, NY 10001".
  + Билет: flightID="123", outboundFlight="Blue Sky Air;123;2025-04-25", returnFlight="Blue Sky Air;456;2025-05-01", pass1="José García".