**РЕЧЬ**

**Слайд #1**

Здравствуйте уважаемые члены государственной аттестационной комиссии, меня зовут Махмудов Бабур. Тема моей выпускной квалификационной работы – «исследование методов повышения производительности систем веб-шаблонов и разработка системы шаблонизации на их основе».

**Слайд #2**

Актуальность данной работы заключается в том, что существующие решения не эффективны в плане потребления вычислительных ресурсов, также большинство из них предназначены для использования в составе более крупных веб-фреймворков, что понижает их показатель удобства в эксплуатации людьми не технического профиля.

**Слайд #3**

Цель, объект, предмет и задачи исследования представлены на слайде №3. Основной целью работы был анализ основных факторов негативно сказывающихся на производительности систем веб-шаблонов, поиск способов их устранения и разработка высокопроизводительной системы веб-шаблонов с применение данных подходов. Основные проблемы производительности выявленные в ходе исследования показаны на следующем слайде.

**Слайд #4**

1. Подавляющее большинство существующих решений написаны на интерпретируемых языках (PHP, Python, JavaScript).
2. Присутствие механизм автоматического управления памятью, также известного как сборщик мусора.
3. Не всегда присутствует кеширования обработанных шаблонов, что приводит к необходимости повторной обработки.
4. Вычислительные ресурсы не используются полностью, из-за исполнения программы в одном потоке, и наличием блокировок, связанных с чтением и записью с медленных хранилищ или каналов передачи данных.

**Слайд #5**

На слайде №5 представлено схематическое сравнение интерпретируемых и компилируемых языков программирования.

При компиляции, компилятор обрабатывает исходный код и производит оптимизированный машинный бинарный код под целевую архитектуру ЭВМ.

При интерпретации обработка исходного кода происходит построчно, и результатом такого процесса является промежуточный код виртуальной машины, так же известной как байт-код. Далее байт-код подвергается процессу трансляции в машинный код целевой архитектуры.

Такой подход к исполнению программы в несколько раз медленнее по сравнению с компилируемым аналогом.

**Слайд #6**

Для решения ранее описанных проблем, были предложены следующие подходы:

1. Использование компилируемых языков программирования вместо интерпретируемых.
2. В выбранном языке программирования не должно быть механизма сборки мусора, а в целях упрощения управления памятью необходимо чтобы язык поддерживал идиому RAII (объект который получил ресурс, ответственен за его освобождение).
3. Реализация кеширования шаблонов с применением вытесняющих алгоритмов LRU (вытеснение давно не используемых объектов из кеша) или LFU (вытеснение наименее используемых объектов).
4. Реализация системы веб-шаблонов как веб-сервис, который может параллельно и асинхронно обрабатывать большое количество запросов на построение веб-документов или на управление шаблонами.

**Слайд #7**

Практическая часть работы посвящена проектированию и разработке ранее упомянутого веб-сервиса по построению веб-документов.

Укрупнённо модель веб-сервиса выглядит следующим образом:

Множество клиентов делает обращения к сервису посредством HTTP запросов, сервис перенаправляет запросы на соответствующие модули в зависимости от запроса.

**Слайд #8**

На слайде №8 приведена IDEF0 диаграмма обработки таких HTTP запросов, по ней видно, что после предварительной обработки запроса веб-сервис перенаправляет запрос на соответствующую подсистему для дальнейшей обработки. Например, это может быть запрос на построение документа или же на обновление шаблона в хранилище.

**Слайд #9**

Для реализации ранее описанного веб-сервиса построения веб-документов были использованы следующие технологии:

Серверная часть была написана на языке программирования Rust (компилируемый язык, без сборки мусора и с повышенными гарантиями на безопасность выполнения). Для поддержки многопоточного и асинхронного исполнения была использована библиотека tokio. А для непосредственно самой шаблонизации в веб-сервис был интегрирован шаблонизатор tera.

Пользовательский веб-интерфейс был написан с применением javascript-фреймворка vuejs и использованием стандартных html и css.

Основания для выбора той или технологии приведены в пояснительной записке.

Также стоит упомянуть, что в силу клиент-серверной архитектуры, клиентом может выступать любая сущность, поддерживающая HTTP протокол обмена данными.

**Слайд #10**

На данном слайде приведён пример пользовательского веб-интерфейса разработанного сервиса. А также по указанной ссылке можно ознакомиться с демонстративной версией разработанной системы.

**Слайд #11**

По завершению разработки были проведены тесты для оценки производительности.

Тестирование велось на виртуальном сервере от Google в Стокгольме в то время как клиентские запросы исходили из штата Вирджиния, на слайде приведены характеристики этого сервера.

На диаграмме показано соотношение одновременных подключений в конкретные интервалы времени и времени ответа веб-сервиса. А именно по диаграмме видно, что на 550 одновременных подключений среднее время ответа системы составляет чуть более 4-х секунд.

**Слайд #12**

Нагрузка, которая фиксировалась на сервере во время проведения теста показана на слайде №12. По нему видно, что оба ядра процессора нагружены только на 50%, т.е., не смотря на большое количество одновременных запросов, такая нагрузка находится в допустимых пределах для разработанной системы веб-шаблонов.

**Слайд #13**

В заключении можно сказать что все поставленные цели были достигнуты: были проанализированы проблемы низкой производительности веб-шаблонов, предложены способы устранения данных проблем, после чего разработана высокопроизводительная и удобная в использовании система веб-шаблонов.