Обеспечение качества высоконагруженного Spring-Boot приложения

И.В. Чумачёва, email: chumacheva@cs.vsu.ru
Н. К. Самойлов, nk.samoylov@gmail.com

Воронежский государственный университет

Аннотация. *Рассмотрены основы обеспечения качества, контроля качества и тестирования программного обеспечения, особенности устройства и тестирования высоконагруженного приложения, проведен анализ полученных результатов и сделаны выводы.*

Ключевые слова: *Высоконагруженное приложение, Spring-Boot, обеспечение качества, контроль качества, тестирование*

Введение

Одна из важнейших задач при разработке программного обеспечения – это обеспечение качества программного обеспечения. Качество программного обеспечения – способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям [1] или степень, в которой система, компонент или процесс удовлетворяют потребностям или ожиданиям заказчика или пользователя. В данной работе решается задача выявить основные подходы по обеспечению качества высоконагруженного Spring-Boot приложения, выявить проблемы, возникающие при обеспечении качества и разработать рекомендации по обеспечению качества для подобных приложений на основе полученного опыта.

1. Теоретические аспекты обеспечение качества программного обеспечения

Обеспечение качества (Quality Assurance – QA) (рис.1) – это совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО для обеспечения качества выпускаемого продукта [2].

Рис. 1. Взаимосвязь обеспечения качества, контроля качества и тестирования

Контроль качества (Quality Control – QC) (рис.2) – это совокупность действий, проводимых над объектом тестирования в процессе разработки для получения информации об актуальном состоянии объекта тестирования в разрезах: «готовность Продукта к выпуску», «Соответствие зафиксированным требованиям», «Соответствие заявленному уровню качества продукта». [2]

Рис. 2. Качество программного обеспечения

Тестирование программного обеспечения (Software Testing) – это одна из техник контроля качества, включающая в себя следующие активности [2]:

- планирование работ (Test management);
- наблюдение и контроль (Test monitoring and control);
- написание автотестов и подготовка тест даты (Test implementation);
- проектирование тестов (Test design);
- выполнение тестирования (Test Execution);
- анализ результатов (Test Analysis);

– завершение тестирования (Test completion).

Существует множество разных видов, типов и уровней тестирования программного обеспечения, фактически любой тип тестирования программного обеспечения может быть выполнен как вручную, так и с использованием средства автоматизации. Выделяют следующие уровни тестирования: модульное (компонентное) тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование [2]. Также существуют различные виды тестирования, которые выделяют в зависимости от преследуемых целей, например: функциональное, нефункциональное (нагрузочное, производительности, надежности, безопасности, конфигурации), связанное с изменениями (регрессионное и проверка после исправлений).

Если рассматривать обеспечение качества нагруженных приложений, то необходимо применять данные виды тестирования в следующем порядке: юнит-тестирование (покрытие кода юнит тестами), тестирование интеграций, функциональное тестирование и тестирование производительности.

Контроль качества программного обеспечения основывается на выбранной модели, анализе критериев, заданных технической документацией, и анализе результатов в виде метрик кода и метрик тестирования. При анализе метрик кода было выявлено, что несмотря на богатый выбор существующих видов метрик, подавляющее большинство компаний на практике практически их не используют для оценки качества. Большой популярностью обладают метрики тестирования, такие как Pass Rate (количество успешно пройденных тест-кейсов/тест-сьютов к общему количеству тест-кейсов/тест-сьютов), Execution Rate (количество выполненных тест-кейсов/тест-сьютов к общему количеству тест-кейсов/тест-сьютов), коэффициент ошибок пропущенных в релиз (отношение количества ошибок обнаруженных после выпуска релиза к общему количеству ошибок обнаруженных до и после релиза)[3].

2. Обеспечение качества для высоконагруженного приложения

Высоконаруженные сервисы – это системы, которые могут принимать, обрабатывать и генерировать большие объемы данных разной природы и из разных источников с течением времени, организованные с помощью различных технологий и с целью извлечения ценности из данных для различных типов предприятий [4].

Основная цель приложения, выбранного для исследования – сбор данных с мобильных клиентов по использованию того или иного

мобильного приложения для последующего анализа данных (как единичного использования, так и BigData).

Первая версия приложения (рис.3) состояла из клиента, который отправлял собранные данные в Data Reader, в свою очередь Data Reader отправлял данные в БД (MySQL). Пользователь (аналитик) через веб-интерфейс заходил в систему и мог получить доступ к истории одного просмотра или послать запрос с помощью выбора фильтров на агрегацию данных и получить результат исходя из поставленных задач. Архитектура состояла из нижеперечисленных сервисов.

– Data Reader – Spring-Boot микросервис, который принимает данные от клиентов по REST протоколу и выдает ответ клиенту;

– БД – реляционная база данных;

– Data Processor – Spring-Boot микросервис, который осуществляет постобработку данных и формирует ответы на запросы пользователей-аналитиков;

– Client API – Spring-Boot микросервис, который предоставляет REST API для web client приложения;

– Web Client – реализация на Native JavaScript и jQuery.

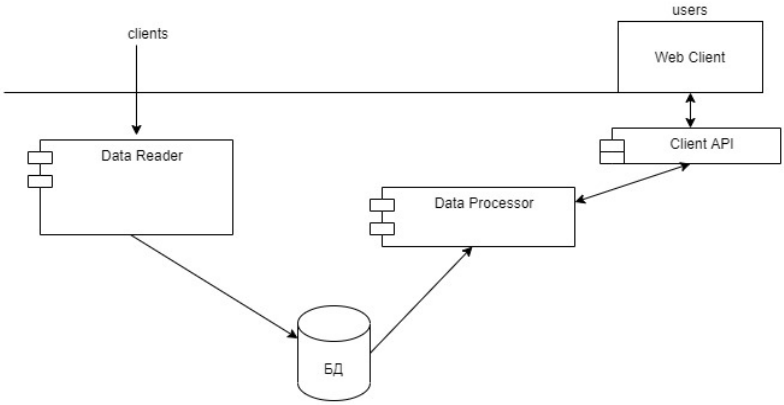

Рис. 3. Первая версия приложения

Были выполнены следующие виды тестирования:

– покрытие отделом разработчиков кода Unit тестами;

– тестирование интеграций: в том числе анализ джейсона полученных данных;

– функциональное тестирование;

– тестирование производительности.

По результатам проведенного тестирования были найдены следующие проблемы, приведенные ниже.

– Потеря данных: если передача данных была прервана нестабильной связью или проблемами у провайдера, данные не хранились у клиента и были потеряны.

– Проблема поврежденных данных: данные могли передаваться не полностью и были сохранены в БД также не полностью, соответственно большая часть полученных данных была “грязными” данными и не пригодными для сбора аналитики.

– Медленная обработка запросов базой данных, которая не удовлетворяла технические требования: обработка агрегирующего запроса должна выполняться не больше минуты, а загрузка истории одной сессии – не более 5 секунд.

– Система не прошла тестирование производительности.

Для устранения подобных проблем необходимо:

– хранить данные о сессиях у клиента в течение суток, чтобы при проблемах со связью, они не потерялись;

– изменить маршруты передачи параметров, любая передача содержала в себе переменную “sessionID”, чтобы устранить потери при порционной пересылке данных, возникающих при проблемах связи или провайдера, и идентифицировать и объединить данные об одной сессии;

– изменить архитектуру микросервисов, улучшить показатели “железа” – оперативную память в том числе;

– добавить еще одну базу данных и перепрофилировать назначение баз данных: чтобы одна отвечала за сбор данных, а другая за их агрегацию.

Помимо вышеуказанных проблем были найдены функциональные дефекты, которые тоже были устранены к выходу второго релиза. Мы повторили цикл тестирования и обнаружили проблемы с производительностью иного характера. Например, DataReader не справился с нагрузкой и в нем переполнялась очередь. Поэтому было принято решение сделать некоторые сервисы масштабируемыми. Также было обнаружено, что для безопасности данных необходимо иметь копии баз данных, а для обработки запросов необходимо добавить очередь. После второй итерации изменений архитектура приложения изменилась в соответствии с рис. 4.

В третьей версии приложения была реализована следующая структура:

– Load Balancer – Netflix Ribbon балансировщик нагрузки;

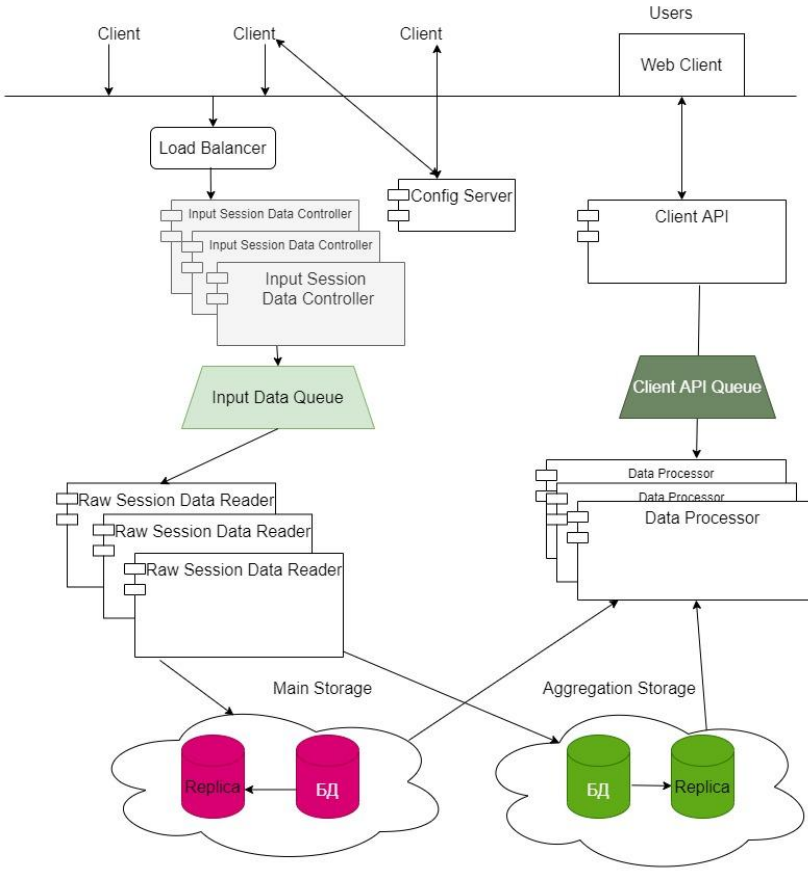

Рис. 4. Третья версия приложения

– Input Session Data Controller – Spring-Boot микросервис, который осуществляет валидацию входных JSON данных и помещает их в промежуточную очередь (Input Data Queue);

– Input Data Queue – очередь для передачи данных между Input Session Data Controller и Raw Session Data Reader;

– Raw Session Data Reader – Spring-Boot микросервис, который сохраняет данные в долговременное хранилище (Main Storage) и в агрегационное хранилище (Aggregation Storage) и производит постобработку данных;

– Main Storage – долговременное хранилище, NoSQL документно-ориентированная СУБД;

– Aggregation Storage – агрегационное хранилище СУБД ClickHouse;

– Data Processor – Spring-Boot микросервис, который формирует ответы на запросы пользователей-аналитиков;

– Client API Queue – очередь для передачи данных между Data Processor и Client Api;

– Client Api – Spring-Boot микросервис, который предоставляет REST API для web client приложения;

– Web Client – реализация на ReactJS.

Клиенты отправляют данные в Input Session Data Controller, где происходит валидация формата присланных данных по флагу “json=yes”, после чего данные о сессии поступают в очередь, оттуда DataReader по параметру “sessionID” находит данные одной сессии и отправляет их в базу данных на хранение. В свою очередь, пользователь через веб интерфейс формирует запрос на получение сессии или отчета из базы данных, этот запрос также сначала отправляется в очередь, а уже потом его обрабатывает сервис обработки данных, взаимодействуя с базами данных. “Main Storage” – база данных предназначена для хранения данных, а “Aggregation Storage” – база данных для обработки агрегирующих запросов.

По результатам тестирования третьей версии приложения были получены следующие метрики:

– Pass Rate=80%;

– Execution Rate=96%;

– Коэффициент ошибок, пропущенных в релиз=5%.

3. Выводы, полученные при обеспечении качества высоконагруженных приложений

Высоконагруженные приложения обретают все большую популярность и должны отвечать определенным требованиям, которые описаны в спецификации системы. Помимо этого, подобные приложения должны обладать следующими возможностями: масштабируемостью, низкой задержкой запросов/загрузок, достоверностью данных, управляемостью (программное обеспечение должно быть простым в обслуживании), экономической эффективностью.

На основе этого практического опыта были сделаны следующие выводы:

– необходимо проектировать систему с учетом будущего масштабирования, а также определить эффективный набор

инструментов, позволяющего решать типовые задачи эффективно по аппаратным ресурсам (базы данных, очереди сообщений, языки программирования, библиотеки и т.д.);

– тестирование – важный этап в обеспечении качества, поэтому необходимо тестировать модули по мере их готовности и применять все возможные виды тестирования для данного приложения;

– разработать систему метрик качества приложения, мониторинга и логирования как инструментов диагностики ошибок и причин сбоев;

– иметь наготове инструменты по восстановлению системы в случае сбоя в кратчайшие сроки, чтобы свести к минимуму последствия.

Заключение

В данной статье кратко рассмотрены основные аспекты обеспечения качества программного обеспечения для высоконагруженных приложений. В результате проделанной работы были решены следующие задачи:

– предотвращение потерь данных: если передача данных была прервана нестабильной связью или проблемами у провайдера, данные не хранились у клиента и были потеряны;

– исправление поврежденных данных: данные могли передаваться не полностью и были сохранены в БД также не полностью, соответственно большая часть полученных данных была “грязными” данными и не пригодными для сбора аналитики;

– стабилизация скорости обработки запросов во время пиковых нагрузок. Обработка агрегирующего запроса должна выполняться не больше минуты, а загрузка истории одной сессии – не более 5 секунд.

Кроме того, были предложены рекомендации для обеспечения качества сервисов с высокой нагрузкой.

Список литературы

1. ISO/IEC 25000:2014(en). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. [Электронный ресурс]. – Режим доступа: https://www.iso.org/ru/standard/64764.html

2. Certified Tester Foundation Level Syllabus, Version 2018, International Software Testing Qualifications Board [Электронный ресурс]. – Режим доступа: https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html

3. Kan Stephen H.: Metrics and Models in Software Quality Engineering/S.H. Kan – 2nd Edition. – Boston: Addison Wesley, 2002. – 448 p.

4. M. Felderer, B. Russo, F. Auer, “On Testing Data-Intensive Software Systems”, 2019. [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/337138666_On_Testing_Data-Intensive_Software_Systems