УДК 004.054

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ МИКРОСЕРВИСОВ ПРИЛОЖЕНИЯ ПО СОЗДАНИЮ РЕЗЕРВНЫХ КОПИЙ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ

Снисаренко С.В., Чех П.С.
БГУИР, г. Минск, Беларусь, kafsu@bsuir.by

Введение

Современный мир все больше требует вовлеченности информационных технологий во все сферы жизнедеятельности человечества, тем самым ускоряя и усовершенствуя свои процессы и технологии. В таком ритме продуктам необходимо оптимизироваться и подстраиваться под потребителя, чтобы занимать лидирующие позиции. Для осуществления такой цели жизненный цикл разработки программного обеспечения (SDLC – Software development lifecycle) должен быть минимизирован на всех его стадиях, но уделяя особое внимание качеству продукта. Программы нельзя считать единственными готовыми продуктам, которые должны быть подвергнуты тестированию в процессе разработки программного обеспечения. В действительности результат каждого промежуточного этапа во всем процессе разработки должен быть проверен на точность, что позволит исключить вероятность ошибки на начальных этапах.

1. Тестирование API

Фаза тестирования играет немаловажную роль в этой иерархии, сопровождаясь проверкой работоспособности системы, выявлением, фиксацией и исправлением багов до тех пор, пока продукт не достигнет необходимых стандартов качества. При планировании и проектировании приложения архитектура имеет большое значение, т.к. именно она определяет качественные атрибуты сервиса: масштабируемость, надежность и безопасность.

Внаши дни важным качеством является также возможность быстрой и безопасной доставки кода. Именно микросервисная архитектура как стиль проектирования делает приложение легко под-

держиваемым, тестируемым и развертываемым [1]. Из этого вытекает необходимость тестирования программного интерфейса приложения (API – Application Programming Interface [2]) которое становится особенно востребованным на первых этапах проектирования, так как именно оно позволяет в полной мере убедиться в правильности работы мискросервиса и интеграции отдельных сервисов, а также в соблюдении логики функционирования системы в целом.

Микросервис – это независимый, автономный ресурс, спроектированный как отдельно выполняемый файл или процесс и взаимодействующий с другими микросервисами через стандартные, но легковесные межпроцессные связи, такие как протокол передачи гипертекста (HTTP), веб-службы RESTful (построенные на архитектуре репрезентативной передачи состояния – Representational State Transfer, REST), очереди сообщений и т. п. [2].

Реализация программного интерфейса приложения может быть как внутренней (приватной), когда программные компоненты находятся внутри приватной сети и нет прямой возможности получить к ним доступ из вне, либо открытой (публичной), когда программные компоненты имеют прямой выход в глобальный интернет, что позволяет внешним пользователям или другим программам получать информацию, которую можно интегрировать в свои приложения. На базе таких архитектурных решений выстраиваются специальные техники тест-дизайна, которые подразумевают под собой этап процесса тестирования программного обеспечения, на котором проектируются и создаются тестовые случаи для покрытия логики, как отдельного модуля, так и интеграции с другими модулями в соответствии с определенными ранее критериями качества и целями тестирования.

По сравнению с монолитными приложениями микросервисная архитектура позволяет облегчить тестирование отдельных сервисов, так как их размер намного меньше. Но в тоже время важно убедиться в том, что отдельные сервисы работают вместе, избегая при этом использования сложных, медленных и ненадежных сквозных тестов [3]. Упростить задачу тестирования сервисов изолированного друг от друга позволяют следующие шаблоны:

– тестирование контрактов с расчетом на потребителя;

– тестирование контрактов на стороне потребителя;

– тестирование компонентов сервиса.

Первый шаблон, тестирование контрактов с расчетом на потребителя, позволяет произвести проверку того, что сервис отвечает ожиданием клиентов. Второй шаблон проводит проверку возможности взаимодействия клиента с сервисом, и третий – тестирование сервиса в изоляции. В рамках данной работы реализовано тестирование программного интерфейса приложения по шаблону тестирования компонентов сервиса.

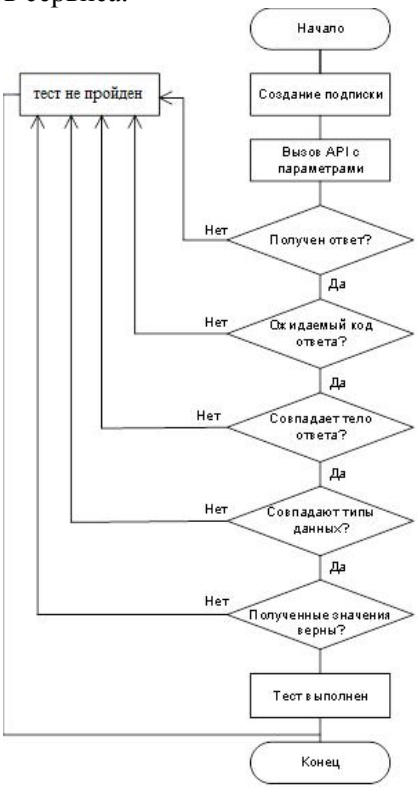

Рисунок 1 – Схема алгоритма по созданию теста

В рамках данной работы разработан алгоритм создания подписки с целью – получить скомпрометированные данные пользователей организации (рис. 1). При включении сервиса для организации создается подписка по названию продукта, которым пользователь

пользуется. После этих действий подключается сторонний сервис, который забирает новую созданную заявку с целью обработки всех пользователей для нахождения всех скомпрометированных email адресов организации. Вся логика происходит на серверной части, только в случае положительного результата организация увидит список данных в веб-интерфейсе.

Заключение

Целесообразность внедрения автоматизации тестирования обусловлена, в первую очередь, возможностью ускорить релизы, позволяя доставить новый функционал либо исправление критической ошибки намного быстрее. Во-вторых, автоматизация рутинных и частых проверок с большим количеством устройств, версий браузеров и операционных систем позволит снизить нагрузку на QA специалистов. При этом, в долгосрочной перспективе, снизит как расходы на тестирование, так и риски, связанные с человеческим фактором. Также не исключением является автоматизация тестирования производительности приложения в условиях одновременной работы с большим количеством данных и пользователей. Спецификой тестирования API является то, что для программного интерфейса приложения необходима базовая техническая подготовка и необходимость использования дополнительных инструментов. На практике для API тестирования применяются такие средства как язык программирования JavaScript [4] и сервис Postman [5]. На языке JavaScript реализуется алгоритмы тестирования миксросервисов, которые имеют структуру вида: входные параметры, реализация вызова API интерфейса, получение результата, сравнивание полученного результата с ожидаемым. Postman является бесплатным сервисом с открытым исходным кодом для запуска реализованных алгоритмов тестирования с возможностью группировки результатов и создания отчетов.

Список литературы:

1. Электронный ресурс: https://ru.qaz.wiki/wiki/Direct_ multiple_shooting_method. Дата обращения 12.11.2020

2. Tamer Basar Francesco Bullo Gregory J. Toussaint. Motion planning for nonlinear under actuated vehicles using h-infinity techniques. IEEE American ControlConference, pages 4097 – 4103, 2001.

3. Weizhong Zhang, Tamer Inanc, and Jerrold E. Marsden. Dmoc approach to real-time trajectory generation for mechanical systems. IEEE Conference on Control, Automation, Robotics and Vision, pages 2192 – 2195, 2008.

4. W. Murray P. E. Gill and M. A. Saunders. Snopt: An sqp algorithm for largescalecon-strained optimization. Report NA 97-2, Department of Mathematics, University of California, San Diego USA, 1997

5. Christopher V. Roa, Stephen J. Wright, and James B. Rawlings. On the application of interior point methods to model predictive control. Journal of optimization theory and application, 99(3):723 – 757, 1998