Тестирование ПО глазами разработчика

Дисклеймер

Всё сказанное в данном видео является оценочным суждением автора и никаким образом не связано с компаниями, с которыми автор сотрудничает.

Цель видео

- Дать теоретические основы и практические навыки юнит и интеграционного тестирования на примере работы со Spring Boot приложениями с использованием JUnit, Mockito и TestContainers.

О себе

- > 10 лет опыта в разработке
- Пишу код
- Говорю слова

План

1. Введение в курс
2. Создание и настройка учебного проекта
3. Юнит тестирование репозиторного слоя
4. Юнит тестирование сервисного слоя
5. Юнит тестирование слоя контроллеров
6. Интеграционное тестирование с “реальной” БД
7. Интеграционное тестирование с Testcontainers
8. Создание и настройка “реактивного” учебного проекта (WebFlux)
9. Юнит тестирование “реактивных” контроллеров
10. Интеграционное тестирование “реактивного” приложения с TestContainers
11. Заключение

Необходимые знания

- Основы языка Java
- Основы Spring Framework
- Базовые знания JUnit
- Базовые знания Mockito

С чем работаем в рамках курса

- Java 11+
- Spring Boot (MVC, JPA, WebFlux, Test)
- JUnit
- Mockito
- БД H2
- БД PostgreSQL
- Gradle
- Intellij IDEA
- Git
- Вспомогательные библиотеки

Рекомендации по изучению

- Просмотр раздела видео (таймкоды)
- Скачать исходный код учебных проектов
- Повторять действия выполненные в видео
- Каждый этап видео находится в отдельной ветке Git репозитория (step 1, step 2, step 3 и т.д.)
- Проверять работоспособность приложения после выполнения задания каждого модуля
- Вопросы и ответы на них в комментариях к видео

Тестирование ПО

- Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом (ISO/IEC TR 19759:2005).

Тестирование ПО

Для чего проводится тестирование ПО?

- Проверка соответствия заявленным требованиям.
- Выявление проблем на более ранних этапах разработки и предотвращения повышения стоимости продукта.
- Обнаружение вариантов использования, которые не были предусмотрены при разработке.
- Избежание репутационных потерь компании. Любой обнаруженный дефект негативно влияет на лояльность пользователей.

Тестирование ПО

1. Модульное (юнит) тестирование.

Сосредотачиваются на проверке небольших компонентов программы, таких как отдельные методы или функции. Они обычно не требуют значительных усилий для автоматизации и могут быть выполнены быстро сервером непрерывной интеграции.

2. Интеграционное тестирование.

Проверяют взаимодействие между различными частями приложения, такими как база данных или микросервисы. Они требуют запуска различных компонентов и, следовательно, являются более затратными по сравнению с модульными тестами.

3. Функциональное тестирование.
Оценивают соответствие работы программы бизнес-требованиям, проверяя лишь конечный результат действия без учета промежуточных состояний.

Тестирование ПО

4. Сквозное тестирование.
Моделируют поведение пользователя в контексте всего приложения. Они проверяют различные сценарии использования, что может быть как простым действием, так и более сложным взаимодействием.

5. Приемочное тестирование.
Формально проверяет соответствие системы бизнес-требованиям. Оно включает запуск приложения и оценку его поведения с точки зрения пользователя, включая производительность.

6. Тестирование производительности.
Оценивает характеристики работы приложения при определенной нагрузке, такие как скорость, масштабируемость и отзывчивость.

7. Smoke-тестирование.
Проверяют основные функциональные возможности приложения быстро после создания новой сборки или развертывания, чтобы убедиться, что основные функции работают корректно.

Тестирование ПО

Как разработчики, мы сосредоточены (в большинстве случаев) на:

- Юнит тестах
- Интеграционных тестах

Юнит тестирование

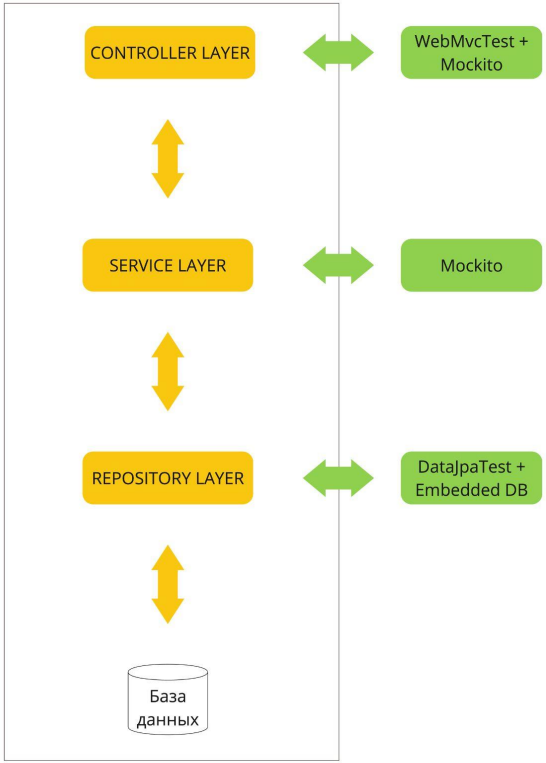
- Тестирование отдельных модулей (методов, классов, функций, объектов, процедур) приложения.
- Основная цель - убедиться, что каждый юнит программы работает в соответствии с ожиданиями.
- Выполняется на этапе разработки и чаще всего является обязательным этапом для инженеров.

Юнит тестирование

Инструменты:

- JUnit
- Mockito

Юнит тестирование



Интеграционное тестирование

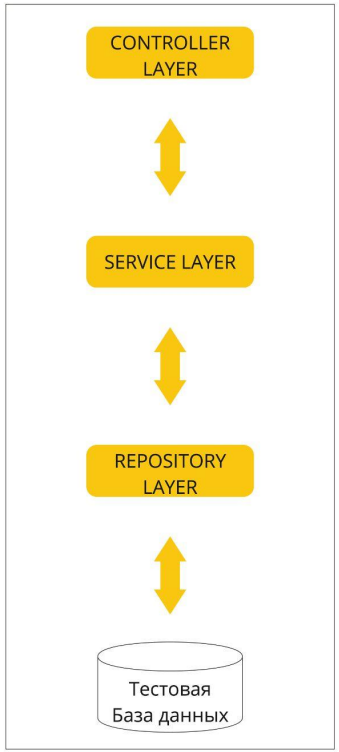
- Интеграция разных компонентов приложения
- НЕ ИСПОЛЬЗУЮТСЯ “ЗАГЛУШКИ” в виде Mock’ов

Интеграционное тестирование

Инструменты:

- Testcontainers
- Реальные тестовые БД

Интеграционное тестирование



Настройка учебного проекта

- Скачать проект по ссылке
- Открыть и запустить локально
- Разбор структуры проекта

Настройка учебного проекта

- В рамках этого раздела ключевую роль играет зависимость spring-boot-starter-test
- Данный стартер включает в себя:
  - Авто конфигурацию
  - “спринговые” зависимости
  - JUnit, Mockito, AssertJ, JsonPath, Hamcrest
- Стартер решает все проблемы совместимости

Настройка учебного проекта

- Hamcrest - это фреймворк, который позволяет нам работать с предикатами (matchers).
- Часто используется в связке с JUnit для проверки утверждений.
- assertThat(<АКТУАЛНОЕ_ЗНАЧЕНИЕ>, is(<ОЖИДАЕМОЕ_ЗНАЧЕНИЕ>))
- JsonPath позволяет нам работать с JSON посредством DSL

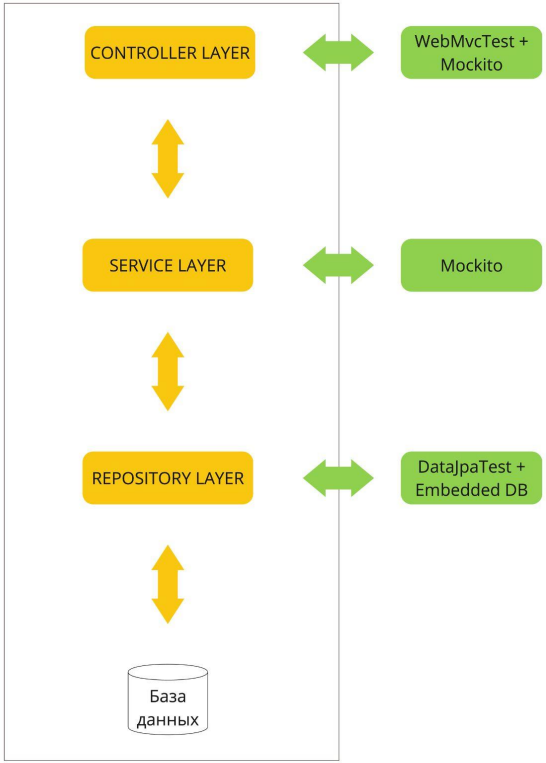
Настройка учебного проекта

- Создание базовой структуры проекта

Тестирование репозитория

- @DataJpaTest используется для тестирования репозиторного слоя
- НЕ ПОДГРУЖАЕТ в контекст компоненты с аннотациями @Component, @Service, @Controller, @RestController, @Bean
- Загружает @Entity, @Respository и JPARepository имплементации
- Отвечает за авто-конфигурацию встроенной БД (H2)
- GIVEN-WHEN-THEN подход

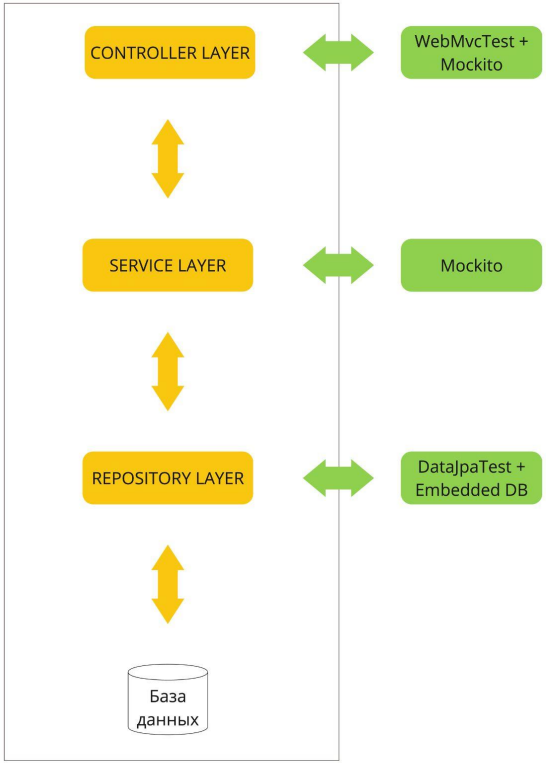
Юнит тестирование



Тестирование сервиса

- Вместо репозитория используется “заглушка” Mockito
- Мы можем создать “заглушку” для класса или интерфейса с помощью метода Mockito.mock(Clazz.class);
- Мы можем создать “заглушку” для класса или интерфейса с помощью аннотации @Mock

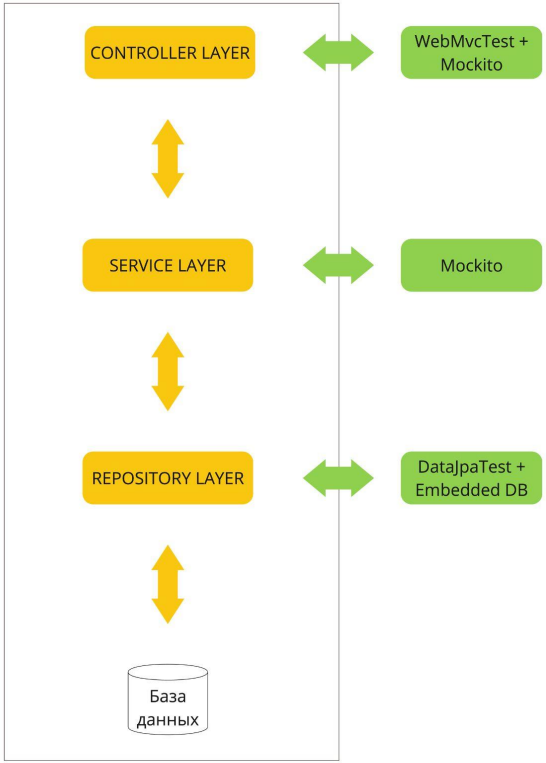
Юнит тестирование



Тестирование контроллера

- Аннотация @WebMvcTest используется для тестирования MVC контроллеров.
- Загружает только указанные контроллеры, вместо всего контекста.

Юнит тестирование



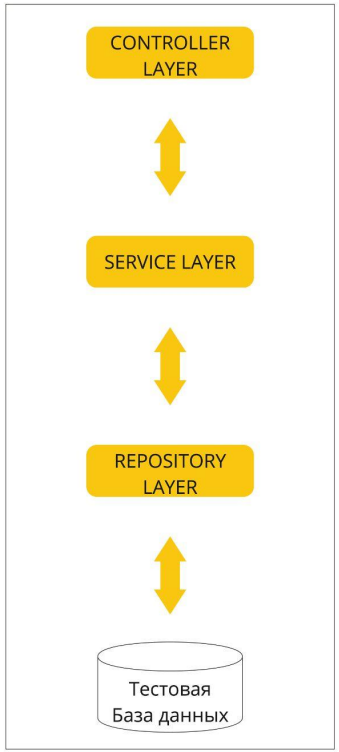
Интеграционное тестирование

- Реальные тест кейсы зависят от нескольких слоев.
- Если мы хотим проверить взаимодействие этих слоев, нам необходимо выполнить интеграционное тестирование.
- Использование аннотации @SpringBootTest
- Она создает и загружает ПОЛНЫЙ ApplicationContext.
- Мы можем подтянуть ЛЮБОЙ bean из ApplicationContext.

Интеграционное тестирование

- @SpringBootTest стартует встроенный сервер и позволяет вызывать нам нужные ендпоинты.
- Мы можем задать порт, на котором будет запущен сервер
- Реализация интеграционных тестов (код).

Интеграционное тестирование



Testcontainers

- Работа с реальной БД не всегда удобно и не всегда безопасна.
- Работа нескольких разработчиков на проекте может быть затруднена при работе с общей тестовой БД.
- Расходы на отдельный инстанс БД
- Testcontainers позволяют использовать Docker контейнеры для замены сторонних сервисов.
- Testcontainers удобное решение, которое предоставляет легковесные инстансы баз данных, брокеров сообщений, облачных сервисов и т.д.

Testcontainers

Преимущества:

- Нет необходимости поддерживать тестовую инфраструктуру.
- Отсутствие конфликтов данных при параллельном исполнении пайплайнов.
- Возможность запуска на локальной машине.
- Автоматическое удаление и очистка контейнеров.
- Пример кода

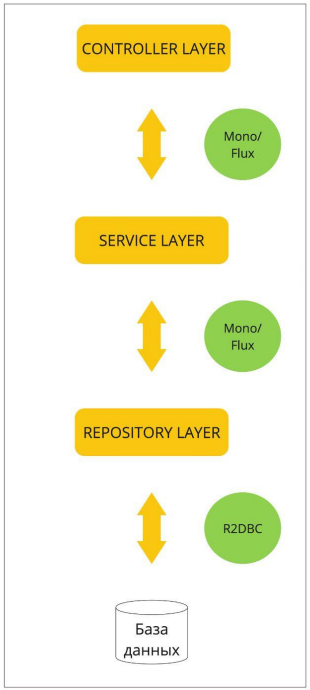
Spring WebFlux

- Фреймворк для асинхронного/реактивного программирования
- Основан на Projectreactor
- Вводятся концепции Publisher: Mono/Flux
- Поддерживается современными веб-серверами
- Многие современные БД имеют реактивный драйверы (Postgres, MySQL, Redis, Cassandra, etc.)

Spring WebFlux тестирование

- Для юнит тестов используется аннотация @WebFluxTest
- Для интеграционных - @SpringBootTest
- Пример кода

Реактивное приложение



Вывод

- Тестирование ПО является неотъемлемым этапом цикла разработки, который помогает снизить расходы на исправление ошибок.
- Помогает в развитии продукта (сценарии использования).
- Снижает риск репутационных потерь компании.

Обратная связь

- Комментарии под видео
- Email: proselytear@yahoo.com