Уровни тестирования

Курс Вадима Ксендзова
"Тестирование ПО"

Давай вспомним про LEGO.

Это набор разноцветных деталей разной формы и размеров, которые после соединения становятся игрушкой.

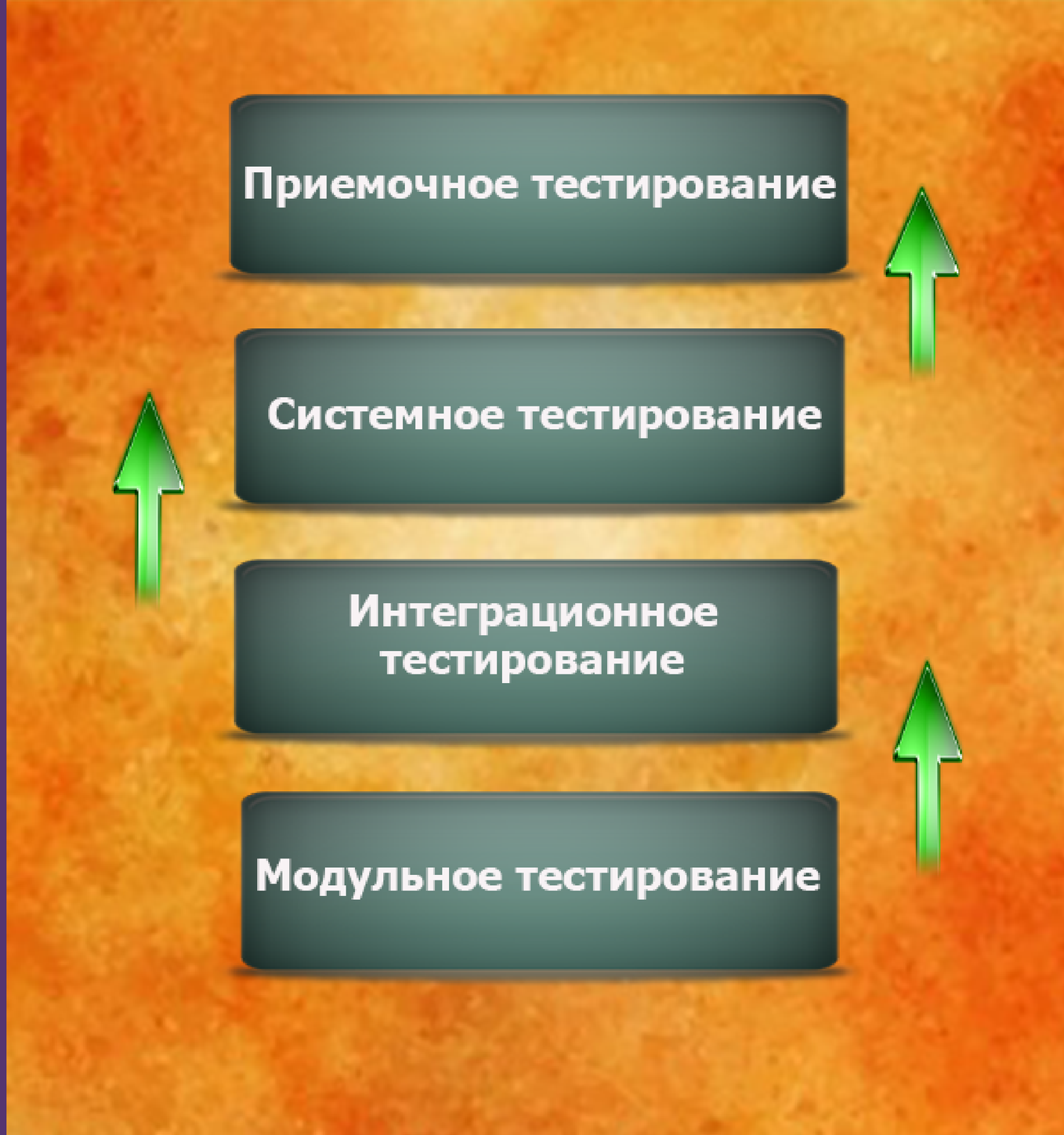
Обычно, процесс сборки игрушки выглядит так:

- Берем детали и инструкцию по сборке, и проверяем, что все на месте (детали правильной формы / размера / цвета)
- Собираем детали в «блоки». Если игрушка — это машинка, то мы соберем несколько блоков: двери, колеса, салон, корпус машины, подвеску и т.п.
- «Блоки» собираем в части побольше (если нужно), а уже из них складываем готовую машинку
- Проверяем, что машинка ездит, двери открывается, ничего от нее не отпадает и т.п.
- В конце мы проверяем, что машинка соответствует изображению и это то, что мы покупали .

Если посмотреть на процесс сборки с точки зрения тестирования, его можно описать так:

1. Проверка каждой отдельной детали на соответствие инструкции = Модульное тестирование.
2. Проверка «блоков», состоящих из отдельных деталей соединённых определенным образом = Интеграционное тестирование.
3. Проверка собранной игрушки = Системное тестирование.
4. Собранная игрушка — это именно та игрушка, которую мы хотели = Приемочное тестирование.

Уровень тестирования определяет то, над чем производятся тесты: над отдельным модулем, группой модулей или системой, в целом. Проведение тестирования на всех уровнях системы - это залог успешной реализации и сдачи проекта.

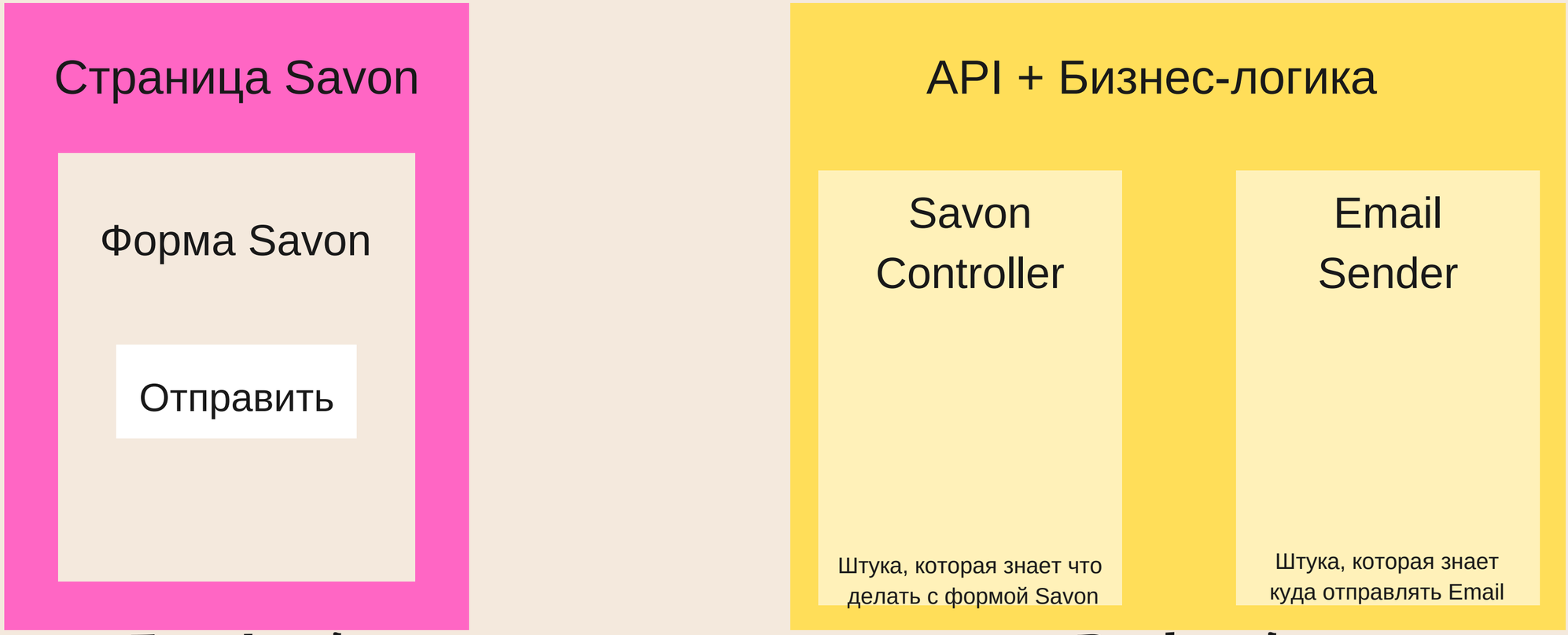


Мы рассмотрим на каждом уровне:

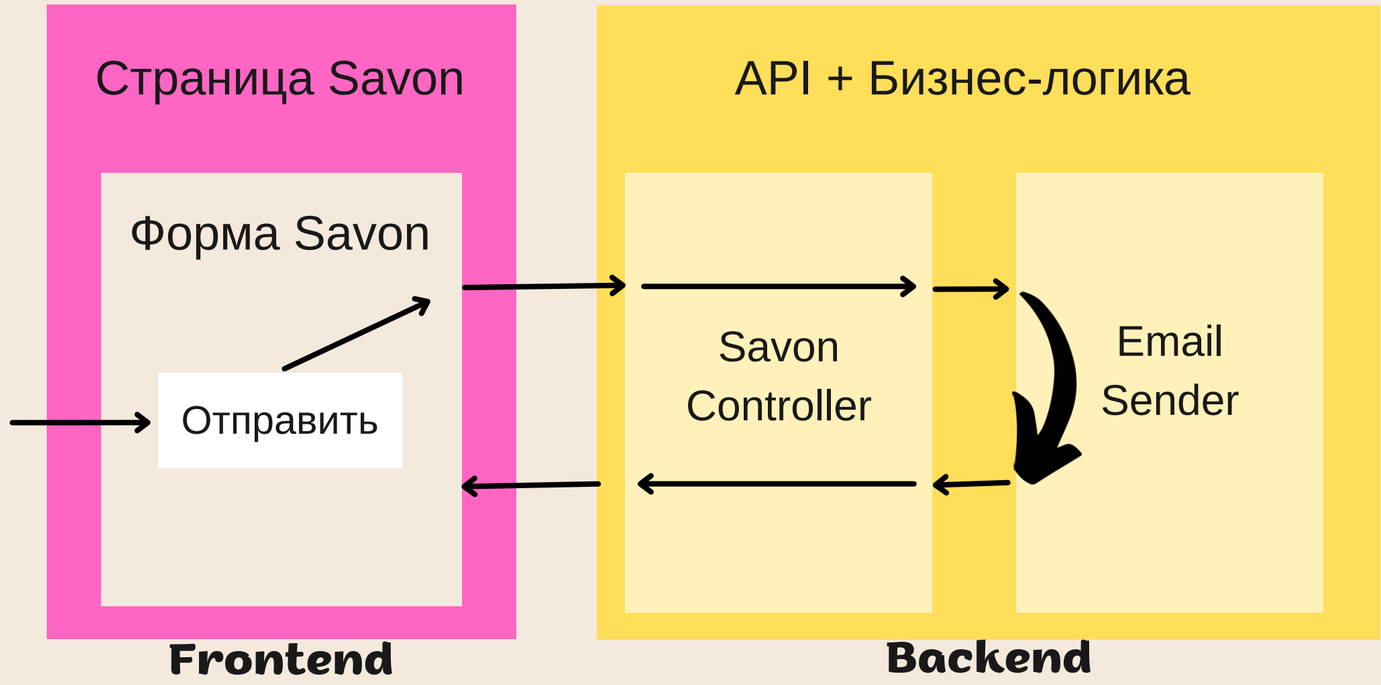
1. Цели тестирования (Для чего мы проводим тестирование?)
2. Объект тестирования (Что мы тестируем? Модуль / компонент / под-систему / систему?)
3. Объект тестирования (Что нам необходимо, чтоб провести тестирование? Спецификации, требования, ТЗ)
4. Дефекты, которые мы планируем найти
5. Кто чем занимается и кто за что отвечает?
6. Окружение (Где проводится тестирование, локально или на сервере,который выпускает продукт?)

Создадим страницу "Регистрация" на сайте интернет-магазина натуральной косметики Savon. После отправки формы отдел поддержки должен получить Email, содержащий введенные данные и контактную информацию клиента.

Первым делом разработчики прорабатывают дизайн системы.



1. Пользователь заполняет форму Savon и нажимает «Отправить»;
2. Frontend проверяет введенные значения;
3. Frontend отправляет данные на Backend;
4. Savon Controller проверяет данные и формирует запрос на отправку письма;
5. Savon Controller передает данные для отправки письма в Email Sender;
6. Email Sender получает данные, проверяет их, формирует письмо и отправляет его;
7. Email Sender отвечает Savon Controller, что письмо отправлено;
8. Savon Controller формирует ответ для Frontend;
9. Backend отправляет данные об успешной отправке письма на Frontend;
10. Frontend получает данные, понимает, что отправка была успешной и показывает пользователю сообщение.



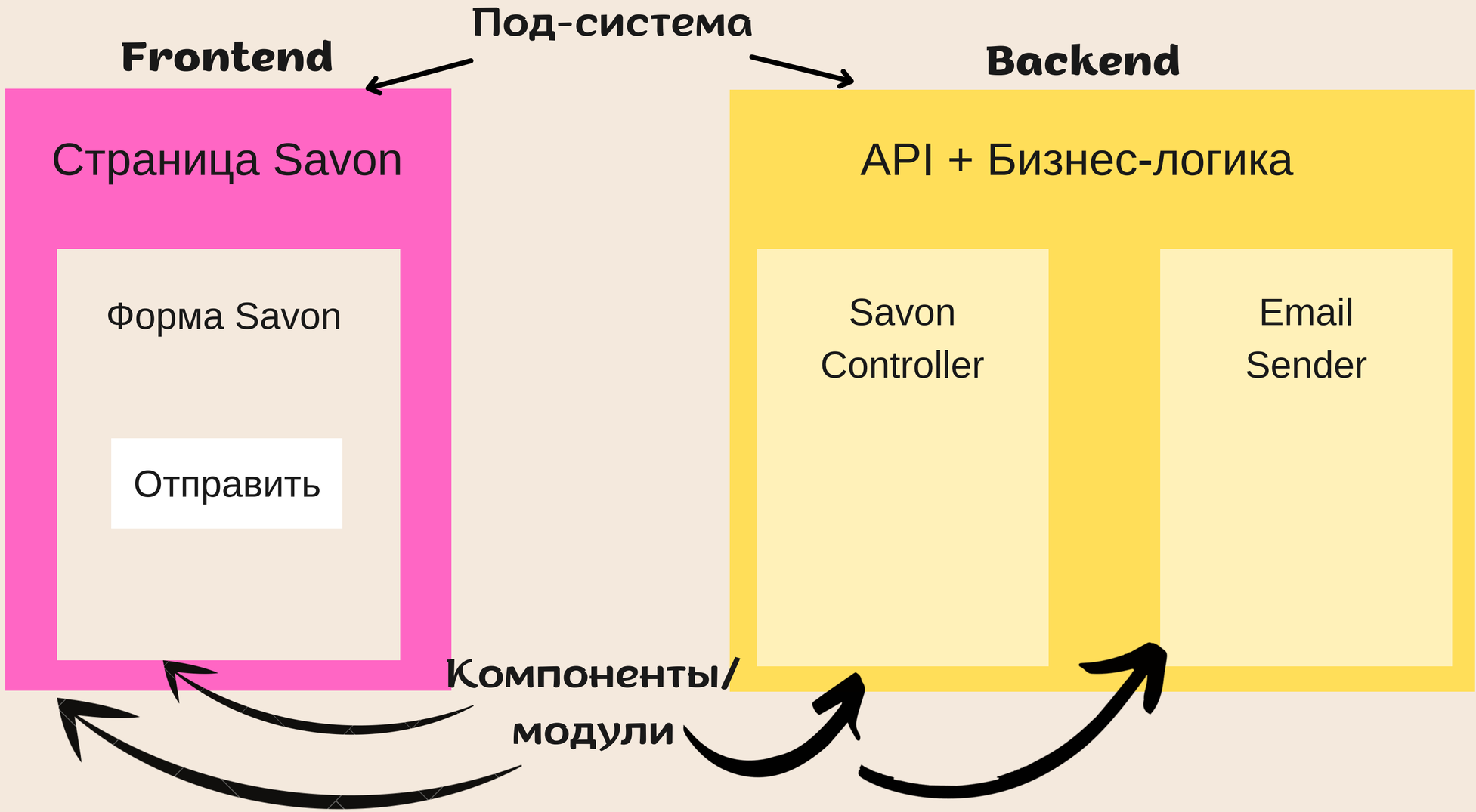
Модульное / Компонентное / Unit тестирование

Модульное / Компонентное / Unit тестирование рассматривает компоненты / модули, которые должны быть проверены в изоляции, как самостоятельные, независимые блоки.

Цель: проверка правильности реализации функциональных / нефункциональных требований в модуле, раннее обнаружение ошибок.
Объект: модуль / компонент / unit.
Базис: дизайн системы, код, спецификация компонента.
Типичные ошибки: ошибка в реализации требований, в коде
Ответственный: разработчик (редко тестировщик).

На этом уровне тестирования создаются модульные тесты (unit тесты), которые проверяют правильность работы модуля в тестовых условиях. Эти проверки всегда автоматизированы и выполняются очень быстро.
Unit тесты, кроме поиска ошибок, также помогают оценивать качество кода, сокращать время и затраты на тестирование.

Выделим следущие подсистемы, компоненты и модули



1. Страница Savon отвечает за показ формы Savon.
2. Форма Savon отвечает за проверку и отправку данных на Backend
3. Savon Controller является частью API и отвечает за обработку запросов с формы Savon
4. Email Sender отвечает за отправку Email

Для примера, рассмотрим модуль «страница Savon».

Требования:

1. Открывается в браузере по указанному URL;
2. Корректно отображается форма Savon;
3. Корректно отображается текст.
4. ...

Т/ребования к модулю «Savon Controller»:

1. Принимать данные по указанному URL (API endpoint);
2. Проверять полученные данные;
3. Правильно формировать данные для компонента Email Sender (без фактической отправки)
4. Возвращать правильные HTTP ответы в случае успешной отправки Email и в случае возникновения ошибки.
5. ...

Все описанные выше требования должны проверяться Unit тестами.

Цель: проверка правильности реализации взаимодействия между компонентами / модулями / частями системы.

Объект: БД, под-системы, API, микросервисы, интерфейсы.

Базис: дизайн системы, архитектура системы, описание связей компонентов.

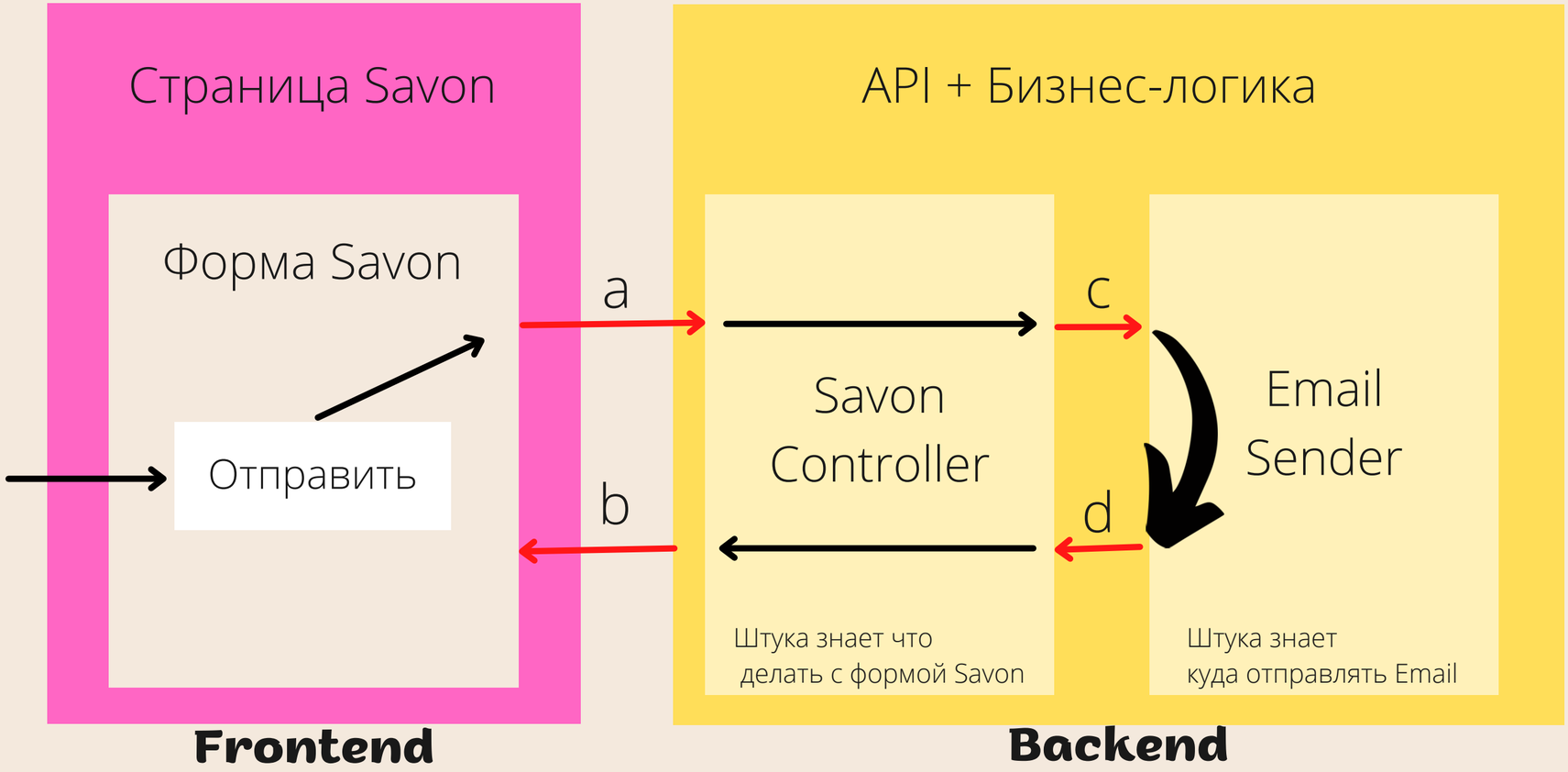
Типичные ошибки: отсутствие / неправильные связи между элементами системы, неправильные передаваемые данные, отсутствие обработки ошибок, отказы и падения при обращениях к API.

Ответственный: разработчик и тестировщик.

Интеграционное тестирование

Системное интеграционное тестирование(a, b)

Компонентное интеграционное тестирование(c, d)



Они описывают связь между компонентами Savon Controller и Email Sender внутри под-системы Backend.
Savon Controller обращается к Email Sender с запросом для отправки Email сообщения, Email Sender отправляет письмо и отвечает Savon Controller что все прошло удачно . Если при отправке произошла ошибка, в ответе вернется информация об ошибке.
В нашем случае интеграционные тесты проверят, что описанный выше процесс работает и что модуль Savon Controller инициирует отправку Email сообщения, а не SMS.
Тестирование интерфейсов (частично) и тестирование API являются примерами интеграционного компонентного тестирования.
В случае с тестированием API мы «имитируем» запрос от клиента — и анализируем ответ сервера — , таким образом проверяя интеграцию всех задействованных модулей для конкретного API Endpoint внутри Backend.

Системное интеграционное тестирование.
Обрати внимание на стрелки a и b.
Они описывают связь между двумя под-системами: Frontend, который формирует и отправляет запрос со страницы Savon с данными формы, и Backend, который обрабатывает и отвечает на запрос.
Тестирование на этом уровне показывает, что интеграция под-систем реализована в соответствии с заявленными требованиями.
В нашем случае для проверки правильности достаточно написать 1 тест: отправить форму Savon с ожидаемым результатом в виде показанного сообщения об успешной отправке — и полученного Email сообщения с данными, оставленными с формы Savon.

Системное тестирование (System Testing)

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

Цель: проверка работы системы в целом.

Объект: система, конфигурации системы, рабочее окружение.

Базис: системные требования, бизнес требования, сценарии использования, User Stories, системные руководства, инструкции.

Типичные ошибки: невозможность выполнять задачи, для которых была создана система, неправильная передача данных внутри системы, неработоспособность системы в среде эксплуатации, нефункциональные сбои (уязвимости, зависания, выключения).

Ответственный: тестировщик.

Кроссбраузерное
Юзабилити

Система

Нагрузочное
Безопасности

Поддержки
Восстановления
Надежности
Сохранности

Frontend

Страница Savon

Форма Savon

Отправить

Backend

API + Бизнес-логика

Savon
Controller

Email
Sender

Помимо проверки отправки формы Savon, получения Email сообщения на почту суппорта и показа Success сообщения, в ходе системного тестирования мы должны ответить на вопросы:

- Работает ли форма Savon во всех поддерживаемых браузерах?
- Удобно ли ей пользоваться? Все ли понятно? Насколько осмысленны сообщения об ошибках?
- Что произойдет, если кто-то отправит 1,000 запросов Savon в секунду? (DDOS)
- Какое максимальное количество запросов можно отправить, чтобы сайт работал без сбоев? (Load testing)
- Насколько читабельным является Email, который получит поддержка? (весь текст в одну строку или письмо оформлено красиво)
- Не попадает ли письмо в Spam?
- Сохраняются ли данные клиента, который отправляет форму? Если «да» — насколько безопасно место хранения? Существует ли способ получения списка отправленных Email-ов?
- Знает ли суппорт о почтовом ящике, куда попадет письмо? Знает ли он, как реагировать на такие письма?
- и много, много других .

На этом уровне тестирования создаются end-to-end тесты, имитирующие бизнес процессы, Use Cases и Use Stories от начала до конца.
Эти тесты все чаще автоматизируется и именно этот вид автоматизации сейчас очень востребован (JAVA, Python, JavaScript, C#, Selenium и т.п. — все здесь).
E2e тесты очень медленные (обычно 5-10 тестов в минуту) и коварные, с их автоматизацией нужно быть очень осторожным .
Системное тестирование — одна из самых творческих и объемных областей тестирования. Кроме end-to-end (e2e) тестирования, к этому уровню относятся все виды нефункционального тестирования.

Приемочное тестирование

Приемочное тестирование фокусируется на готовности всей системы в целом.
Существуют несколько форм приемочного тестирования:

Пользовательское приемочное тестирование (User Acceptance testing, UAT) — проверяет пригодность системы к эксплуатации конечными пользователями.

Контрактное приемочное тестирование — проводится в соответствии с критериями, указанными в контракте приемки специального ПО.

Альфа-тестирование (alpha testing) и бета-тестирование (beta-testing) — используются для получения обратной связи от потенциальных или существующих клиентов.

Альфа-тестирование проводится “внутри” компании, без участия разработчиков / тестировщиков продукта.
Бета-тестирование проводится реальными пользователями системы.

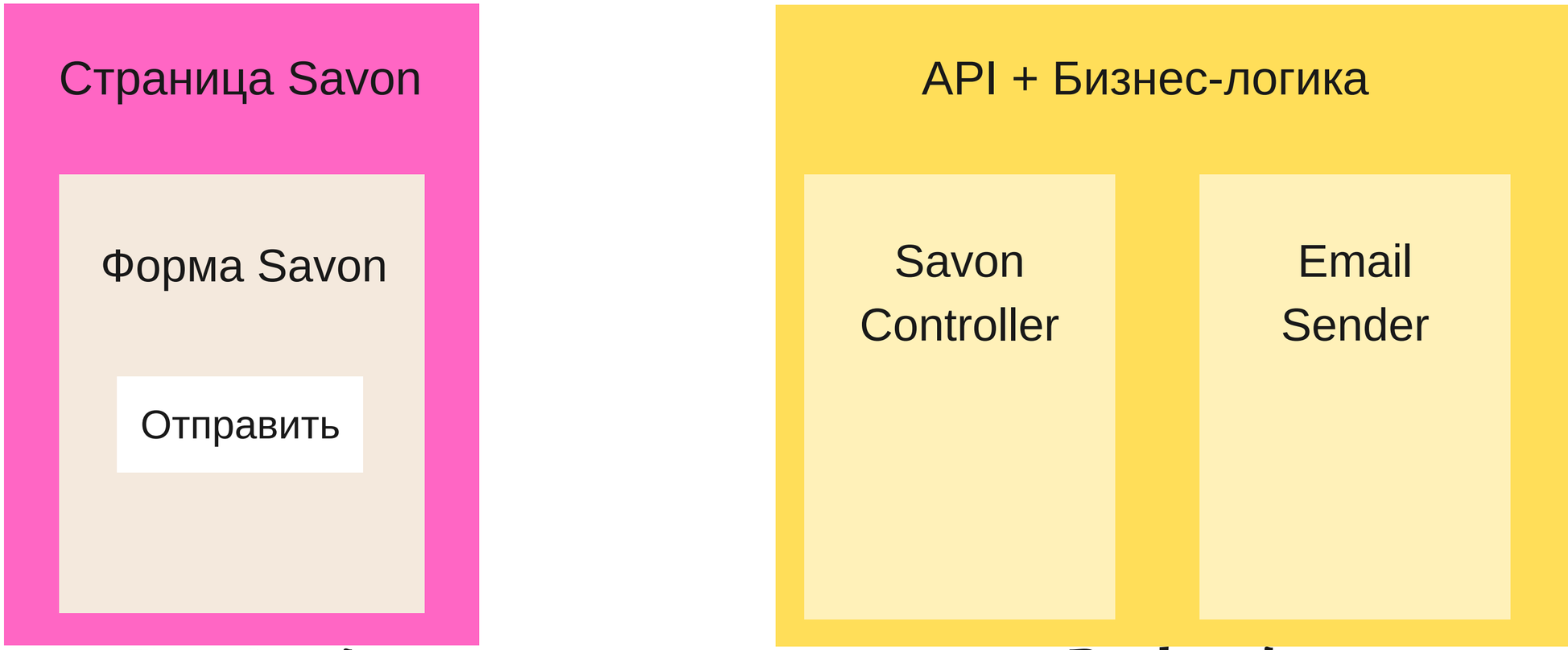
Цель: проверка готовности системы.

Объект: система, конфигурация системы, бизнес процессы, отчеты, аналитика.

Базис: системные требования, бизнес требования, сценарии использования, User Stories.

Типичные ошибки: бизнес-требования неправильно реализованы, система не соответствует требованиям контракта.

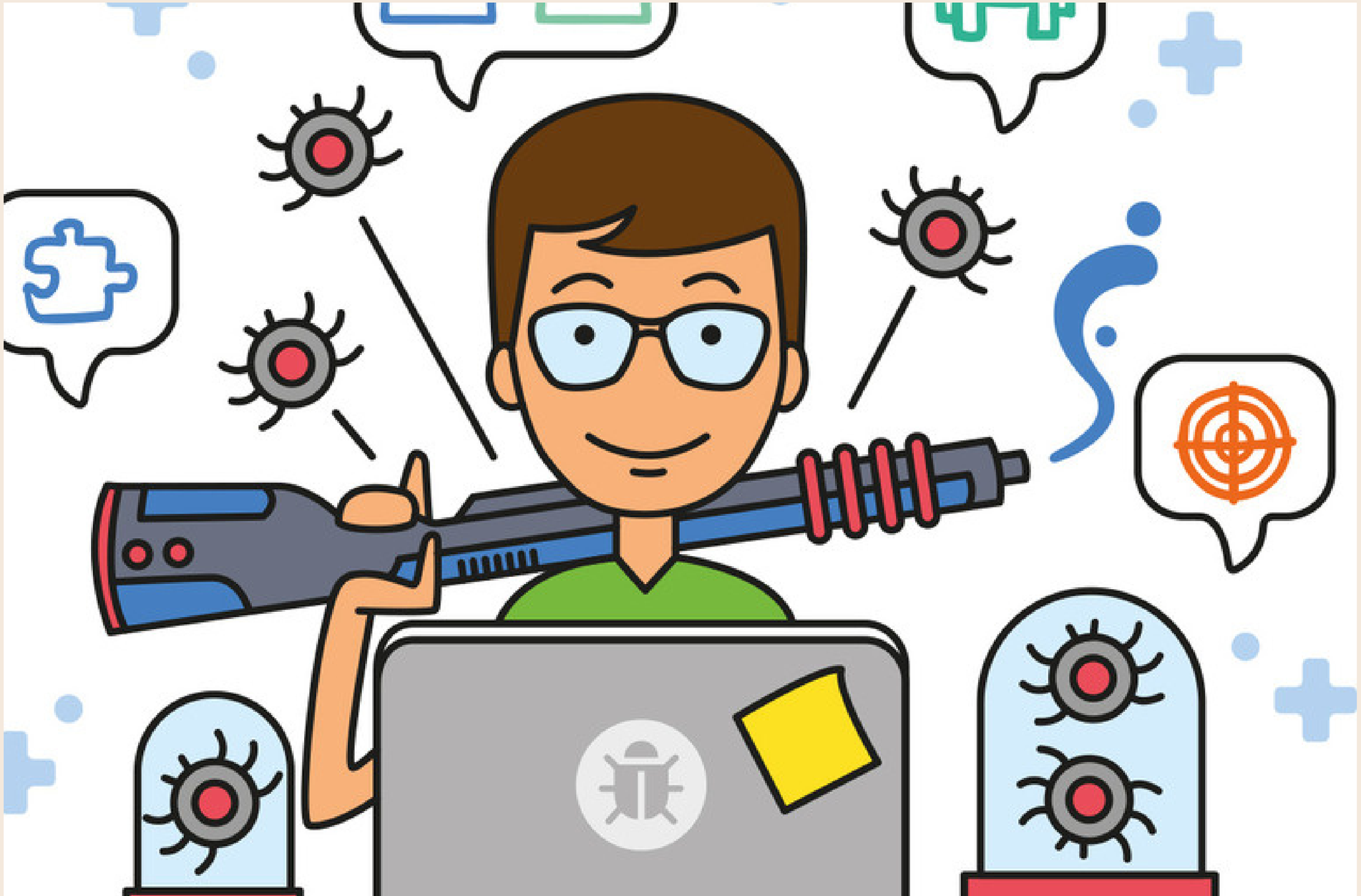
Ответственный: заказчик / клиент / бизнес-аналитик / product owner и тестировщик.



Заказчик заполняет форму, нажимает на кнопку «Отправить»;
Через 1 секунду он видит сообщение об успешной отправке формы;
В течении минуты на почту поддержки приходит письмо;
содержащее данные; отправленные с формы.

Количество тестов на приемочном уровне намного меньше, чем на других уровнях, потому что в этот момент времени вся система уже проверена. Приемочные тесты практически никогда не автоматизируются.
В Agile разработке, конкретно в Scrum, для всех User Stories обязательно прописываются Acceptance Criteria. Именно они являются основой для приемочных тестов и показывают, что команда сделала именно то, что было нужно.
После завершения приемочного тестирования задача передается клиенту.

Спасибо за внимание!



Использованная литература.

1. Уровни тестирования ПО BeQA.pro.

   https://beqa.pro/blog/testing-levels/

2. Разница между компонентным и модульным тестированием.

   https://blog.emptyq.net/a?id=00013-8299949c-c39f-4bab-b4a5-f223f030b93d

3. Тестирование ПО с нуля. Виды, типы и уровни тестирования ПО.

   https://www.youtube.com/watch?v=d52AIqssLtO