**API**— это Application Programming Interface, или программный интерфейс приложения, с помощью которого одна программа может взаимодействовать с другой.

Чтобы программам общаться между собой, их API нужно построить по единому стандарту. Одним из них является **REST** — стандарт архитектуры взаимодействия приложений и сайтов, использующий протокол HTTP.

**JSON**(англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Но при этом формат независим от JS и может использоваться в любом языке программирования. JSON используется в REST API

**XML** — используется в SOAP *(всегда)* и REST-запросах *(реже)*

**SOAP** (Simple Object Access Protocol) — данные передаются в формате XML

REST архитектурный стиль взаимодействия компонентов, а SOAP – это формат обмена структурированными сообщениями.

**JavaScript** — это язык для управления веб-страницами в браузере. Он исполняется только в браузере, может красить элементы на странице в разные цвета, рисовать интерфейсы, пуляться данными, но всё — в браузере.

**CI/CD**, или «непрерывная интеграция/непрерывная доставка» либо «непрерывное развертывание», — это методика разработки программного обеспечения, реализуемая благодаря инструментам автоматизации. Регулярные и надежные обновления уменьшают циклы выпуска за счет непрерывной доставки кода.

CRUD (Create, Read, Update, Delete) - Это основные методы работы с базами данных. Операции CRUD предназначены для редактирования данных программы. Давайте рассмотрим подробнее, что означает каждая операция:

* **GET** – метод GET позволяет получить информацию из источника/базы данных.
* **POST** – метод POST позволяет вносить информацию в источник/базу данных.
* **PUT** – метод PUT позволяет обновлять существующую информацию в источнике/базе данных.
* **DELETE** – метод DELETE удалять существующую информацию из источника/базы данных.

**CRM** — аббревиатура от **Customer** **Relationship** **Management**, дословно переводится как «система управления взаимоотношениями с клиентом». CRM автоматизирует продажи, маркетинг, аналитику, базы данных или все вместе взятое

**JUnit** — фреймворк (специальная библиотека) изначально для модульного тестирования программного обеспечения на языке Java.

Apache **Maven** — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. ), являющемся подмножеством XML.

**TestNG** – это фреймворк для тестирования, написанный на Java

Atlassian **Bamboo** (commit в Bitbucket, build, deploy, release) – CI система. Bamboo имеет несколько полезных возможностей при работе с ветками. Во-первых, можно собирать не только основную ветку (master), но и все другие ветки репозитория. Страница «Triggers» позволяет настроить условия, при которых план запускается на выполнение. Например, билд может запускаться периодически в заданное время и только при условии, что в репозиторий были отправлены изменения. Можно запускать билд, по завершению коммита, используя механизм хуков (post-commit hook), или же просто раз в день, как это делают многие команды.

Atlassian Bamboo хорошо интегрирован с JIRA и другими сервисами Atlassian.

**Логирование** в KIBANA

**Bitbucket** — это сервис для хостинга систем управления версиями кода (Version Control System, VCS). Через такую систему разработчики отслеживают изменения в коде.

**База знаний и инструкций на Confluence**

Confluence была разработана в 2004 году и быстро зарекомендовала себя как одна из самых удобных систем организации совместной работы над контентом, позволяющая создавать базы знаний, делать публикации, вести обсуждения, работать с документацией, хранить файлы, доступные через Web, создать корпоративный информационный портал и многое другое.

# **Облачные бизнес-модели: Простыми словами об IaaS, PaaS и SaaS**

программное обеспечение как услуга (SaaS), платформа как услуга (PaaS) и [инфраструктура как услуга](https://1cloud.ru/blog/modeli-saas-paas-iaas) (IaaS).

На верхнем уровне располагается SaaS — по сути, это облачные приложения, доступ к которым предоставляется через веб-интерфейс. За ним следует PaaS — платформа для самостоятельной разработки и развертывания приложений.  
  
На третьем уровне расположился IaaS — серверы, хранилища, сети, вычислительная инфраструктура, которую клиент получает в пользование для запуска своих решений.

Для демонстрации этих трех типов услуг часто применяется аналогия с пиццей — своеобразная «Pizza as a Service». Когда потребитель заказывает и поедает пиццу в кафе или ресторане, то это SaaS, а если заказывает её себе на дом, то это PaaS. Если же он пошел в магазин, купил ингредиентов и приготовил блюдо самостоятельно, то, можно сказать, что это IaaS.

*1. Что знаете о тестировании нагрузки? В каком случае следует проводить такое тестирование? На каком этапе готовности продукта?*

Нагрузочное тестирование ставит своей задачей эмулировать нагрузку, создаваемую большим количеством пользователей или операций. А его цель: найти предельные значения производительности тестируемого ПО, при превышении которых его работа перестает быть стабильной.

Для продуктов, ориентированных на небольшие компании или объемы данных, нагрузочное тестирование действительно может быть не так критично. Но когда компания выходит на рынок корпоративных клиентов, где нагрузки на продукт могут быть в десятки раз выше, а финансовые риски измеряться сотнями миллионов долларов, тут уже лучше бы освоить эту область знаний.

Когда мы добавляем что-то новое в эту систему, это может так повлиять на общую архитектуру, что весь продукт перестанет работать под нагрузкой. Поэтому новые фичи надо проверять заранее и отдельно. Как только появилась более-менее стабильная версия, уже в этот момент её следует отдавать для нагрузочных тестов. Потому что если архитектурная проблема всплывёт в самом конце, на её исправление и новые проверки придётся потратить несоизмеримо больше сил и ресурсов, чем если бы её заметили на начальных этапах.

**Тест-дизайн:**

1. Классы эквивалентности
2. Граничные значения
3. Попарное тестирование
4. Таблица решений
5. Таблицы и диаграммы состояний и переходов
6. Domain-тестирование (когда одно значение зависит от второго, этот метод облегчает одновременное тестирование нескольких переменных)
7. Тестирование вариантов использования. (*это сценарий, который описывает использование системы действующим  
   лицом для достижения определенной цели. "Действующим лицом" является пользователь. Сценарий представляет собой последовательность шагов, которые описывают взаимодействия между действующим лицом и системой*
8. Тестирование потока данных. *(это одна из двух техник тестирования белого ящика. С помощью данной техники тестирования определяются пути выполнения кода программного модуля, после чего создаются и исполняются тест-кейсы для покрытия этих путей.)*

*2. Что такое таблица решений/decision table и как её можно использовать?*

**Decision Table (таблица решений)** — техника, помогающая наглядно изобразить комбинаторику условий из ТЗ. В тестировании таблица решений используется для того, чтобы на основе требований составить тест-кейсы. И ничего не забыть при сложных комбинациях входных условий! Ведь каждая строка или столбец таблицы → готовый тест-кейс.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Правило 1** | **Правило 2** | **Правило 3** | **Правило 4** |
| **Условия** |  |  |  |  |
| Стаж 5 лет | Нет | Нет | Да | Да |
| Был в авариях? | Да | Нет | Да | Нет |
|  |  |  |  |  |
| **Результат** |  |  |  |  |
| Страховка | 200 руб | 100 руб | 50 руб | 10 руб |

*3. Как организовать сквозное тестирование (e2e)?*

**Сквозное тестирование** (End-to-End) — это процесс проверки программной системы вместе с ее подсистемами. Самая большая трудность при этом типе тестирования состоит в том, что необходимо располагать достаточным количеством информации о всей системе, а также о взаимосвязанных подсистемах.

Фреймворк сквозного тестирования включает в себя три части:

1. Создание пользовательских функций
2. Создание условий
3. Создание тест-кейсов

В мире сквозного тестирования лидером отрасли является testRigor. Он помогает создавать тесты без кода для веб-интерфейса, нативных и гибридных мобильных приложений, мобильных браузеров и API. С его помощью можно тестировать электронную почту и SMS, загруженные файлы .XLS, .DOC, .PDF и т. д.

*4. Приведите примеры подходов для тестирования локализации*

Техника псевдо-локализации ([Pseudo-localization approach](https://lingoport.com/what-is-pseudo-localization/)). В рамках этого подхода вам нужно подготовить псевдо-локаль (сделать это может и сам тестировщик, не имеющий знаний нужного языка), используя любой другой язык и добавить в файлы необходимые вам данные, например спец. символы определенного языка. Также можно добавить большее количество символов в строки, чтобы проверить будет ли обрезаться текст (truncate), ведь текст может стать длиннее после перевода на новую локаль и не вместиться в отведённое для него пространство. Ещё такой подход поможет вам найти проблему с объединением строк (concatenation) и проблемы, связанные с отображением шрифта, если таковые имеются. Для того, чтобы такие проблемы можно было проще найти, то в начало и конец строки можно добавить какой-нибудь символ.

*5. Что такое A/B тестирование?*

A/B тестирование — это мощный маркетинговый инструмент для повышения эффективности работы вашего интернет-ресурса.

Идея A/B тестирования очень проста. Пользователи ресурса случайным образом делятся на сегменты. Один из сегментов остается без изменений — это контрольный сегмент “A”, на основе данных по этому сегменту мы будем оценивать эффект от вносимых изменений. Пользователям из сегмента “B” показываем измененную версию ресурса.  
  
Чтобы получить *статистически значимый* результат, очень важно исключить влияние сегментов друг на друга, т.е. пользователь должен быть отнесен строго к одному сегменту. Это можно сделать, например, записав метку сегмента в cookies браузера.  
  
Для снижения влияния *внешних факторов*, таких как рекламные кампании, день недели, погода или сезонность, замеры в сегментах важно делать параллельно, т.е. в один и тот же период времени.

*6. Когда нужно использовать технику Pairwise?*

Техника попарного тестирования очень помогает при разработке тестов для приложений, включающих множество параметров. Тесты разрабатываются таким образом, что для каждой пары входных параметров существуют все возможные комбинации этих параметров. Тестовые наборы (тест-сьюты, Test suite) охватывают все комбинации.

*7. Что такое fuzz-тестирование?*

**фазз-тестирование** - это автоматизированный метод [тестирования программного](https://translated.turbopages.org/proxy_u/en-ru.ru.17b0a6c6-634862ec-9831edb8-74722d776562/https/en.wikipedia.org/wiki/Software_testing) обеспечения, который включает предоставление неверных, неожиданных или [случайных данных](https://translated.turbopages.org/proxy_u/en-ru.ru.17b0a6c6-634862ec-9831edb8-74722d776562/https/en.wikipedia.org/wiki/Random_data) в качестве входных данных для [компьютерной программы](https://translated.turbopages.org/proxy_u/en-ru.ru.17b0a6c6-634862ec-9831edb8-74722d776562/https/en.wikipedia.org/wiki/Computer_program).

Этот анализ является техникой тестирования программного обеспечения, используемой разработчиками и тестеровщиками на проникновение для проверки своих приложений на неожиданные, недопустимые и случайные наборы вводимых данных.

Для целых чисел: ноль, возможно, отрицательные или очень большие числа

Для символов: экранированные, интерпретируемые символы / инструкции (например: для SQL-запросов, кавычки / команды и т.д.

Для двоичных чисел: случайные единицы

*8. Назовите последовательность выполнения CI/CD процесса на проекте*

**Написание кода.** Каждый из разработчиков пишет код своего модуля, проводит ручное тестирование, а затем соединяет результат работы с текущей версией проекта в основной ветке. Для контроля версий используется система Git, либо аналогичные решения. Когда участники команды опубликуют код своих модулей в основной ветке, начнется следующий этап.

**Сборка.** Система контроля версий запускает автоматическую сборку и тестирование проекта. Триггеры для начала сборки настраиваются командой индивидуально — фиксация изменений в основной ветке проекта, сборка по расписанию, по запросу и т.д. Для автоматизации сборки используется Jenkins, либо аналогичный продукт.

**Ручное тестирование.** Когда CI система успешно проверила работоспособность тестовой версии, то код отправляется тестировщикам для ручного обследования. При этом тестовая сборка получает номер кандидата для дальнейшего релиза продукта (например, v.1.0.0-1).

**Релиз.** По итогам ручного тестирования сборка получает исправления, а итоговый номер версии кандидата повышается (например, после первого исправления версия v.1.0.0-1 становится v.1.0.0-2). После этого выпускается версия кода для клиента (например, v.1.0.0) и начинается следующий этап цикла.

**Развертывание.** На этом этапе рабочая версия продукта для клиентов автоматически публикуется на production серверах разработчика. После этого клиент может взаимодействовать с программой и ознакомиться с ее функционалом как непосредственно через готовый интерфейс, так и через облачные сервисы.

**Поддержка и мониторинг.** Конечные пользователи начинают работать с продуктом. Команда разработки поддерживает его и анализирует пользовательский опыт.

**Планирование.** На основе пользовательского опыта формируются запросы на новый функционал для продукта, готовится план доработок. После этого цикл замыкается и переходит в начальную стадию — написание кода. Далее начинается новая итерация CI/CD разработки.

*9. Есть ли разница между bug leakage и bug release?*

**Bug release** — это баги, которые были выпущены в релиз и команда разработки о них знает.  
**Bug leakage** — это, неизвестные команде разработки, баги, которые находят пользователи после релиза.*11. Какие модели risk-based testing вы знаете?*

Тестирование на основе рисков (risk-based testing) — это метод тестирования программного обеспечения, который базируется на вероятности рисков. Вероятность рисков определяется путем анализа, в котором учитываются сложность программы, критичность функции для бизнеса, частота ее использования и количество возможных дефектов.

***Снифферы трафика*** - Снифферы - это инструменты, позволяющие перехватывать, анализировать и модернизировать все запросы, которые через них проходят. Они полезны, когда из потока нужно извлечь какие-либо сведения или создать нужный ответ сервера. Так можно проводить модульное тестирование продукта, в котором есть и бэк, и фронт, и разные команды со своей версионностью.

**Test-driven development – (TDD) разработка через тестирование**.  
  
Фактически, это модульное тестирование. Суть в следующем:

1. Прежде чем писать код реализации некоей возможности, пишут тест, который позволяет проверить, работает ли этот будущий код реализации, или нет. Прежде чем переходить к следующему шагу, тест запускают и убеждаются в том, что он выдаёт ошибку. Благодаря этому можно быть уверенным в том, что тест не выдаёт ложноположительные результаты, это — своего рода тестирование самих тестов.
2. Создают реализацию возможности и добиваются того, чтобы она успешно прошла тестирование.
3. Выполняют, если это нужно, рефакторинг кода. Рефакторинг, при наличии теста, который способен указать разработчику на правильность или неправильность работы системы, вселяет в разработчика уверенность в его действиях.

**Kanban**

Основная **разница** между **Scrum** и **Канбан** — в длине итераций. В **Scrum** итерации — 2 недели, в **Kanban** задачи программисту можно «подсовывать» хоть каждый день. **Kanban** дает больше гибкости, если под гибкостью понимать частоту смены приоритетов.

**Что такое AJAX?**

AJAX — это аббревиатура, которая означает Asynchronous Javascript and XML. На самом деле, AJAX не является новой технологией, так как и Javascript, и XML существуют уже довольно продолжительное время, а AJAX — это синтез обозначенных технологий. AJAX чаще всего ассоцириуется с термином Web 2.0 и преподносится как новейшее Web-приложение.  
  
  
При использовании AJAX нет необходимости обновлять каждый раз всю страницу, так как обновляется только ее конкретная часть. Это намного удобнее, так как не приходится долго ждать, и экономичнее, так как не все обладают безлимитным интернетом. Правда в этом случае, разработчику необходимо следить, чтобы пользователь был в курсе того, что происходит на странице. Это можно реализовать с использованием индикаторов загрузки, текстовых сообщений о том, что идёт обмен данными с сервером. Необходимо также понимать, что не все браузеры поддерживают AJAX (старые версии браузеров и текстовые браузеры). Плюс Javascript может быть отключен пользователем. Поэтому, не следует злоупотреблять использованием технологии и прибегать к альтернативным методам представления информации на Web-сайте.  
  
  
Обобщим достоинства AJAX:

* Возможность создания удобного Web-интерфейса
* Активное взаимодействие с пользователем
* Частичная перезагрузка страницы, вместо полной
* Удобство использования

AJAX использует два метода работы с веб-страницей: изменение Web-страницы не перезагружая её, и динамическое обращение к серверу.

В основе технологии лежит использование нестандартного объекта XMLHttpRequest, необходимого для взаимодействия со скриптами на стороне сервера.

Шифрование ключей.

QAuth1, QAuth2, APIkey

Значение HEADERS

HTTP:// :8080 – порт для Web-приложений

HTTPS:// :443 – порт для Web-приложений

Пара IP-адрес + порт называется СОКЕТОМ

SSL/TLS – сертификат защиты для передачи данных по HTTPS. Шифрует данные, проверяет целостность сообщений, аутентификацию.

Как работает SSL/TLS:

* Клиент сообщает серверу по какой версии SSL/TLS он будет общаться.
* Сервер обрабатывает эту информацию и возвращает ответ клиенту в котором содержится сертификат, по которому они договорились работать и открытый ключ шифрования.
* Клиент обращается в центр сертификации и проверяет валидность сертификата
* Если сертификат подтвержден, клиент генерирует и шифрует сеансовый ключ, при помощи открытого ключа, полученного от сервера.
* Клиент передает серверу шифрованный сеансовый ключ.
* Сервер расшифровывает сеансовый ключ, полученный от клиента, при помощи закрытого ключа шифрования, который есть только у него.
* Теперь сеансовый ключ есть как у клиента, так и у сервера. При помощи этого ключа происходит общение, в соответствии с методами версии сертификата.

**Что такое Docker?**

По сути, Docker очень похожи на виртуальную машину (VM), которая устанавливается поверх host OS. Но, VM очень тяжеловесны в плане ресурсов. На практике, приходится разворачивать проект на сервере, настраивая совместимости с front-end и back-end, DB. Это зачастую сложно. VM упрощают этот процесс, т.к. они могут быть настроены под нужды каждой части проекта.

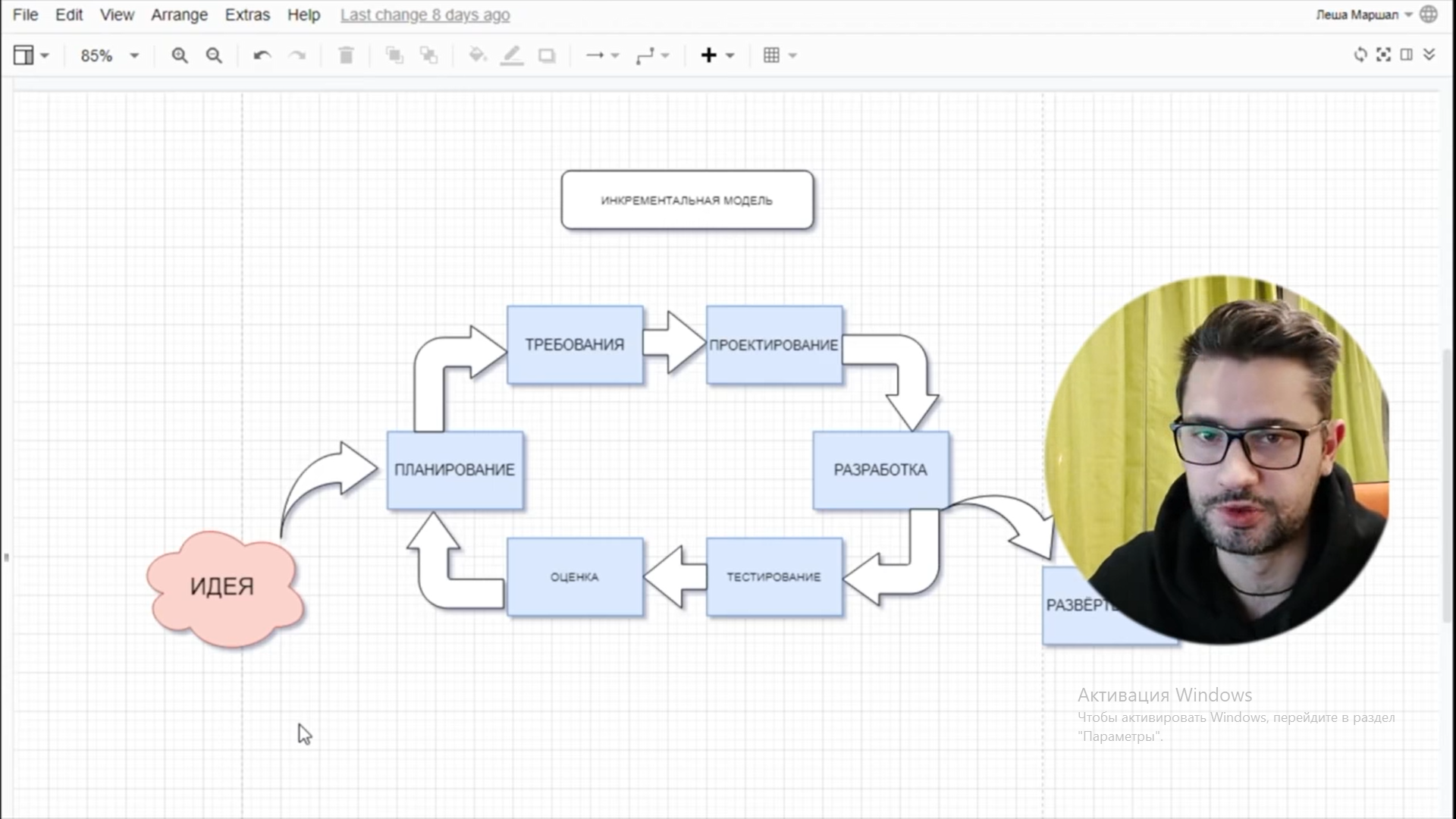
Контейнеры – тоже устанавливаются на серверную OS, но каждый контейнер, в отличии от VM использует ядро этой серверной OS, оставаясь при этом изолированным (независимым) от других контейнеров. В результате, контейнеризация менее ресурсоемка, чем виртуализация.

Docker – это программа содержащая YML файл, в котором единожды прописаны все необходимые настройки. Поэтому, к примеру достаточно чтобы на серверной OS (Ubuntu к примеру) был установлен Docker с файлом YML, чтобы загрузить нужный контейнер и развернуть в нем нужную часть проекта. (Запускается докер командой “docker run -imageName”)

**ТЕОРИЯ ТЕСТИРОВАНИЯ**

**ВОДОПАДНАЯ МОДЕЛЬ**

**V- МОДЕЛЬ**

**ИНТКРЕМЕНТАЛЬНАЯ / ИТЕРАТИВНАЯ МОДЕЛЬ**

**ОБЪЕКТЫ ТЕСТИРОВАНИЯ**

* Документация (требования, спецификация)
  + - Требования прямые и косвенные
    - Функциональные (бизнес-требования)
    - не функциональные (производительность, интерфейс работы – протокол передачи, платформа)
* Код (модульные тесты)
* Софт
* Хард
* Прототипы приложения

**ТЕСТ-ПЛАН**

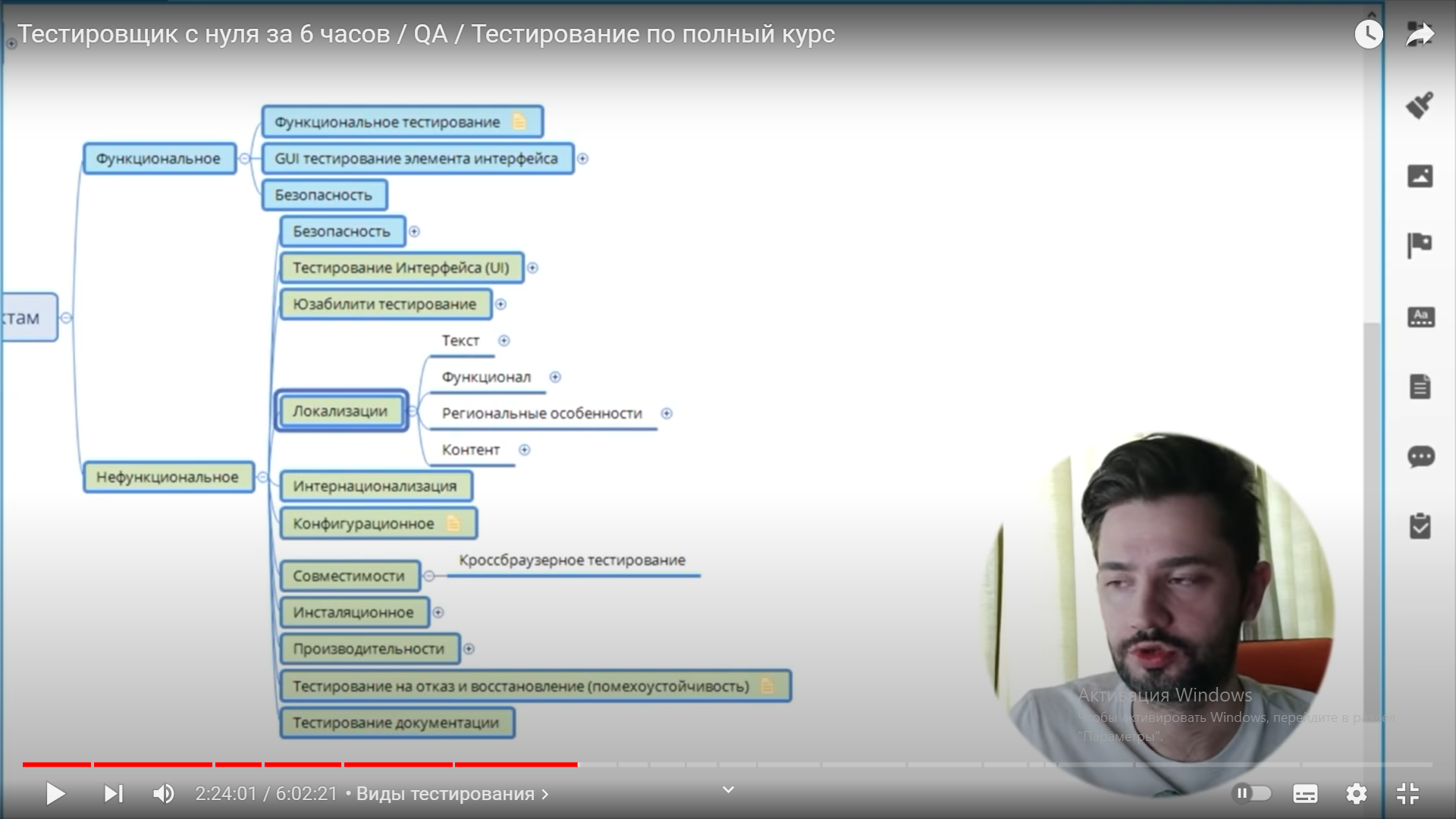
**Тест-план**. Туда пишем:

1. ЧТО надо протестировать
2. Что мы БУДЕМ действительно тестировать
3. КАК мы будем это делать
4. КОГДА мы будем это делать
5. Критерии сдачи
6. Критерии приемки

Более подробно:

1. Перечень работ (какие будут выполнены и какие НЕ будут)
2. Критерии качества;
3. Оценка рисков (ваши навыки, вложиться в срок, возможная замена QA, оборудование);
4. Документация (тест-кейсы и их шаблон, чек-листы, нужен ли отчет, ревью тест-кейсов внутри команды);
5. Тестовая стратегия (методы тестирования – black/white/grey, уровни тестирования – UNIT/Интеграционный/UI, виды тестирования – regression etc.)

**ВИДЫ ТЕСТИРОВАНИЯ**



Что Я использовал из видов:

- Тестирование интерфейса (UI)

- Юзабилити тестирование

- Тестирование Локализации (перевод на поддерживаемые языки, разные валюты)

- Совместимости (кросс-браузерное, кросс-платформенное)

- Тестирование Документации

- Безопасности (в ключе шифрования паролей в базе)

- Функциональное

- Регрессионное

- Smoke

- Интеграционное (как новые модули программы взаимодействуют со сторонней системой)

**JAVA**

Типы данных:

* **boolean**: true/false
* **byte**: хранит целое число от -128 до 127 и занимает 1 байт
* **short**: хранит целое число от -32768 до 32767 и занимает 2 байта
* **int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта
* **long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт
* **double**: хранит число с плавающей точкой от ±4.9\*10-324 до ±1.7976931348623157\*10308 и занимает 8 байт
* **float**: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта
* **char**: хранит одиночный символ в кодировке UTF-16 и занимает 2 байта

Пример округления до одного знака после запятой:

double totalBonus = regularBonus + specialBonus;  
  
BigDecimal result = new BigDecimal(totalBonus);  
result = result.setScale(1, RoundingMode.*UP*);  
System.*out*.println(result);

**ФРЭЙМВОРКИ ЛОГИРОВАНИЯ:**

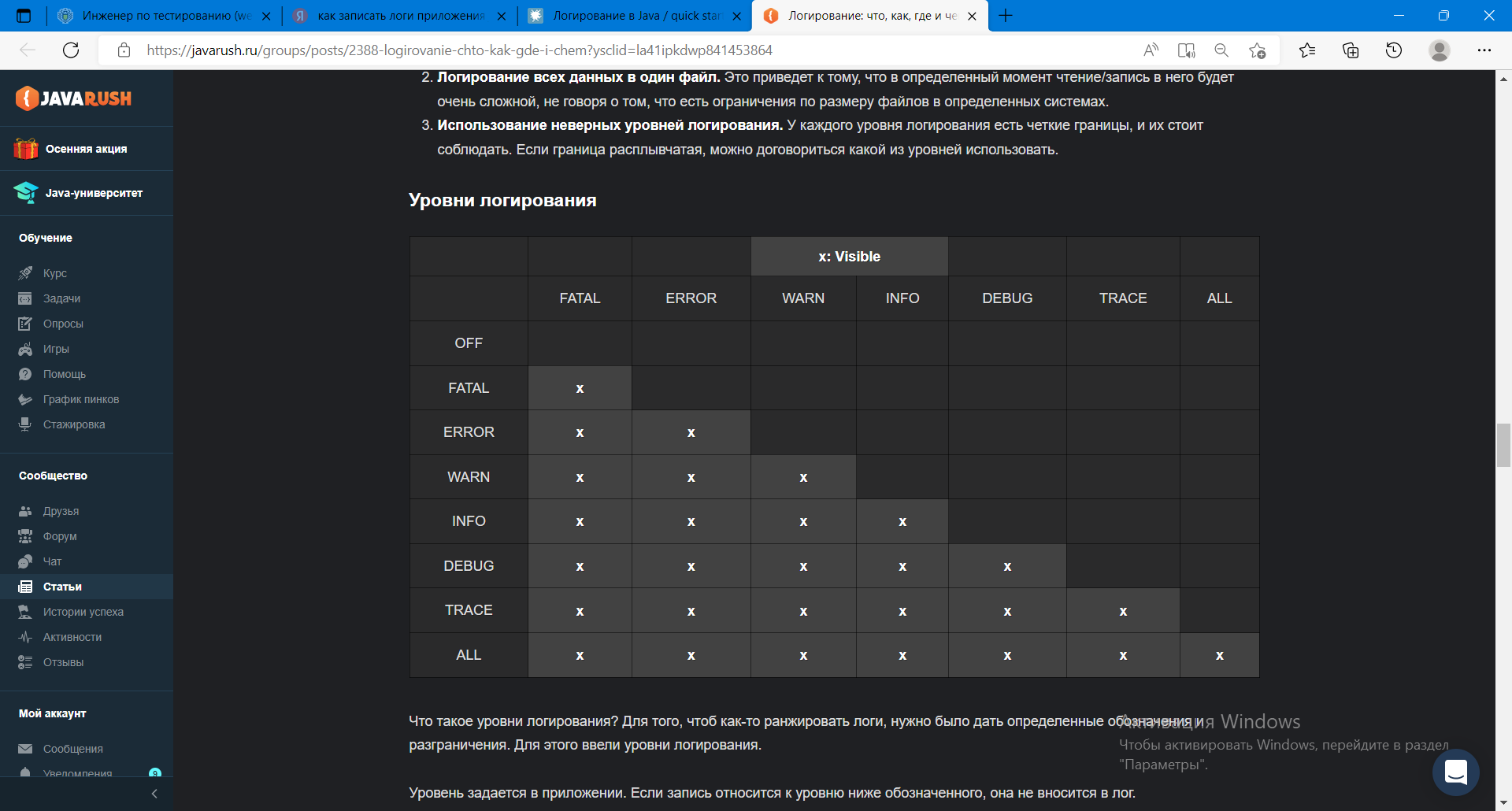
**Java.util.logging**

Уровни логгирования по возрастанию: FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE

**Log4j** имеет несколько отличное от JUL именование уровней логгирования: TRACE, DEBUG, INFO, WARN, ERROR, FATAL

JUL — java.util.logging

java.util.logging.Logger log = java.util.logging.Logger.*getLogger*(Box.class.getName());



**АВТОМАТИЗАЦИЯ RESTASSURED JAVA**

1. Создаем новый проект Maven
2. В *pom.xml* нужно добавить Dependencies:

<dependency>  
 <groupId>io.rest-assured</groupId>  
 <artifactId>rest-assured</artifactId>  
 <version>5.2.0</version>  
 <scope>test</scope>  
</dependency>

1. Для аннотаций *@Test* (и других, используемых в автоматизации) нужно добавить зависимость либо *Junit*, либо *TestNg*:

<dependency>  
 <groupId>org.testng</groupId>  
 <artifactId>testng</artifactId>  
 <version>7.6.1</version>  
 <scope>test</scope>  
</dependency>

1. В проекте, в папке *test/java* создаем *package*, в который добавляем тестовый класс.
2. В классе, после аннотации *@Test* метод, в котором будем писать Rest Api тест:

public class MarsPhotos {  
 @Test  
 public static void getPhotos() {  
 Response response =  
 RestAssured.*given*()  
 .when()  
 .contentType(ContentType.*JSON*)  
 .get("https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&page=2&api\_key=DEMO\_KEY")  
 .then()  
 .log().all()  
 .statusCode(200)  
// .extract().body().jsonPath().getString();  
 .assertThat().extract().response();  
 }  
}

1. Тест начинается с ключевого метода *given()*, после которого, через точку следуют вызовы других возможных методов, обязательными из которых являются *when()* и *then().*
2. Чтобы извлечь из запроса в консоль тело запроса пишем log().all()
3. Хорошей практикой является создание отдельного класса спецификации, в котором указываются статичные переменные типа URL, ContentType. Спецификация позволяет не прописывать всякий раз в тестах базовый адрес и тип ответа, к примеру. Достигается это путем объявления метода у которого возвращаемый тип *RequestSpecification.* Параметры задаются через метод *RequestSpecBuilder()*.  
   То же самое касается спецификации для ответа. Там используется метод с возвращаемым типом *ResponseSpecification.* Параметры задаются через метод *ResponseSpecBuilder()*.

public class Specification {  
  
 public static RequestSpecification requestSpec(String url) {  
 return new RequestSpecBuilder()  
 .setBaseUri(url)  
 .setContentType(ContentType.*JSON*)  
 .build();  
 }  
  
 public static ResponseSpecification responseSpecOK200() {  
 return new ResponseSpecBuilder()  
 .expectStatusCode(200)  
 .build();  
  
 }  
}

1. В следующем методе класса спецификации мы присваиваем возвращаемые значения этих двух методов библиотечным переменным:

public static void installSpecification(RequestSpecification request, ResponseSpecification response) {  
 RestAssured.*requestSpecification* = request;  
 RestAssured.*responseSpecification* = response;  
}

1. Теперь тестовый класс можно немного сократить:

public class MarsPhotos {  
  
 public static final String *baseUri* = "https://api.nasa.gov/mars-photos/api/v1/";  
 public static final String *basePath* = "rovers/curiosity/photos?sol=1000&page=2&api\_key=DEMO\_KEY";  
 @Test  
 public static void getPhotos() {  
 Specification.*installSpecification*(Specification.*requestSpec*(*baseUri*), Specification.*responseSpecOK200*());  
  
 List<Photo> response = *given*()  
 .when()  
 .get(*basePath*)  
 .then()  
 .log().all()  
 .extract().body().jsonPath().getList("img\_src", Photo.class);  
 }  
}

1. Мы можем извлекать JSON в поля класса. Такой класс называется POJO. Т.е. у него такие же поля (типы и названия полей) как в Json body. Pojo class легко сгенерировать специальными онлайн тулами, скопировав тело Json ответа. На скрине выше, показано, что ответ мы записываем в Pojo класс “Photo”.

Сереализация - преобразование Java-объектов в текстовый формат JSON

Десериализация – обратный процесс: преобразование данных из формата JSON в нужный тип данных в Java при ответе сервера.

**GIT BASH команды**

git init – создать локальный репозиторий

git status – проверить в какой ветке и есть ли файлы в состоянии staging

git checkout -b [новая\_ветка] - создать ветку

git checkout [репозиторий]/[ветка] - перейти в ветку

git branch -d [ветка] – удалить ветку

git add – добавить файлы для коммита (в index ветку)

git commit – закоммитить.

git remote add origin <https://github.com/EvgeniyVolk/master.git> - создать ветку в на удаленном сервере

git push <https://github.com/EvgeniyVolk/master.git> - залить изменения на удаленный сервер в ветку

Добавить удаленный репозиторий и присвоить ему короткое имя: **git remote add shortname (e.g. origin) https://github.com/EvgeniyVolk/reponame**

Удалить ветку с сервера GitHub: **$ git push origin --delete branch\_name**

Залить на сервер из локального репо: **$ git push -u origin branch\_name**

Посмотреть историю коммитов: **$ git log --oneline –decorate --all**

Чтобы посмотреть последний коммит на каждой из веток: **git branch -v**