# **1. Жизненный цикл разработки ПО**

# **2. Методологии**

- Вотерфол (когда начинается этап тестирования), минусы

- Итеративные методологии разработки

# **3. Виды тестирования**

⁃ Функциональное (проверяем работоспособность, также тестирование безопасности)

⁃ Нефункциональное (проверяем удобство и соответствие)

⁃ Связанное с изменениями (изменения в коде, не вызвали ли новые баги)

# **4. Уровни функционального**

⁃ модульное (юнит) (отдельные классы, функции)

⁃ Интеграционное (тест взаимодействия между многими модулями и другими системами)

⁃ Системное (удовлетворение функциональным и нефункциональным требованиям)

⁃ Приемочное (удовлетворение требований заказчика)

# **5. Виды нефункциональных**

⁃ Нагрузочное (имитация большого количества запросов)

⁃ Стрессовое (имитация аварийной нагрузки)

⁃ Стабильности и надежности (проверяет работу при средней нагрузке на длительном промежутке времени)

⁃ Тестирование на отказ и восстановление (проверка на восстановление после ошибки, вируса или взлома, разрыва сети)

⁃ UI/UX (соответствие макетам/удобство пользователя) (напр: адаптивность, кроссбраузерность, кроссплатформенность, локализация)

# **6. Связанное с изменениями**

⁃ Дымовое(вширь) (проверка основных и критические важных функций)

⁃ Санитарное(вглубь) (проверка узкой функции по требованиям)

⁃ Регрессионное (проверка старого функционала, после слияния с новым и другими изменениями)   
⁃ Тестирование сборки

# **7. Среды разработки**

**DEV-среда**

Место, где осуществляется разработка чего-либо. Там всегда самая свежая версия кода и продукта. Все новые идеи реализуются в первую очередь там и из разряда «давайте-ка сделаем…» переходят в конкретные строки кода.

**DEMO-среда**

Здесь хранится промежуточный результат, который можно «потрогать руками» и посмотреть так ли все работает, как должно. Если имеется заказчик, то он «трогает» и высказывает свое мнение насчет изменений. В это время на dev-сервере может уже по сто раз все изменится и поломаться.

**TEST-среда**

Продукт проверяется на стабильность путем моделирования нештатных ситуаций или использования нестандартных данных. По сути, это испытательный полигон, на котором можно делать все, вплоть до полного уничтожения всего и вся (на самом деле, нет).

**STAGE-среда**

Ее еще называют предпродакшн. Здесь используются данные из последнего бэкапа системы на prod-сервере, чтобы максимально проверить работоспособность и стабильность приложения или сервиса. Эта среда максимально приближена к тому, что видят перед собой конечные пользователи.

**PROD-среда**

Продакшн, если раскрыть слово prod. Это то с чем взаимодействуют пользователи. Если вы заходите на сайт, в приложение или игру, то взаимодействуете с тем, что опубликовано на prod-сервере. В этот же момент новый функционал во всю может разрабатываться и тестироваться на предыдущих средах.

# **8. Методики тестирования**

⁃ Белая (знаем код и логику, имеем доступ к внутренностям, для автотестов и самотестов кодера)

⁃ Серая (частично знаем)

⁃ Чёрная (не знаем кода и логики)

# **9. Тестовая документация**

⁃ Тест-план (документ об объекте теста, стратегии и сроков, состоит из: что тестируем, как, когда, критерии начала и конца тестов)

⁃ Чек-лист (список пунктов, которые нужно проверить)

⁃ Тест-кейс (совокупность шагов и условий для теста функции или ее части, состоит из: предварительных условий, шагов, ожидаемого результата, бывает позитивный(при правильных действиях получим правильный результат) и негативный(при некорректных действиях будет ошибка) (имеют приоритет)

⁃ Баг-репорт (отчёт о найденных ошибках, содержит инфу о ситуации, причины и результата ошибки, состоит из: заголовка, описания, название проекта, статус ЖЦ, окружение, серьёзность, приоритет, автор, исполнитель, допфайлы)

⁃ Отчет о тестировании

# **9. Серьёзность (влияние дефекта на работоспособность)**

⁃ Блокирующий (баг нельзя обойти)

⁃ Критический (баг серьёзный, но можно обойти)

⁃ Значительный (баг сильно мешает)

⁃ Незначительный (баг относится к пользовательскому интерфейсу)

⁃ Тривиальный

# **10. Приоритет (ставит тимлид)**

⁃ Высокий

⁃ Средний

⁃ Низкий

## **Пример дефекта с высоким приоритетом и низкой серьезностью:** 1) Ошибка в названии компании, логотипе, что вредит репутации компании 2) Не работает ссылка на коммерческого-партнера

## **Пример дефекта с низким приоритетом и высокой серьезностью:** Ошибка в функционале, который должен быть выпущен позже, но тестируется уже сейчас

# **11. Техники тест-дизайн (помогают оптимизировать создание тест-кейсов)**

*⁃ Эквивалентное разделение* (много числовых значений, которые делим на диапазоны по принципу эквивалентности по одинаковому признаку)

*⁃ Анализ граничных значений* (предположение, что ошибки возникают на границе значений)

*⁃ Предугадывание ошибки/исследовательское тестирование* (предположение на опыте для случаев, в которых часто возникают ошибки)

⁃ Исчерпывающее тестирование (использование крайне редких случаев, упаковываемся в огромное количество проверок)

⁃ Матрица соответствия требований (двумерная таблица требований и сценариев для их проверки)

⁃ Попарное тестирование (минимум тестов для максимального покрытия)

# **12. Клиент-сервер**

⁃ Доменное имя - человеческое название для айпи

⁃ Виды клиентов (веб/мобилка/десктоп) (толстый/тонкий)

⁃ Очерёдность действий:   
— браузер обращается к днс(сопоставляет домен с айпи)   
— браузер посылает хттп запрос к серверу   
— статус ответа и пакетная отправка от сервера   
— браузер собирает пакеты воедино

⁃ Методы: get(самый частый)(3 заголовка, нет тела)(данные отправляются на сервер в урле), post(create), put(update), delete(данные отправляются на сервер в теле запроса)(все 4 запроса)

⁃ JSON - формат передачи данных (ключ:значение)

⁃ HTTP - 80,

⁃ HTTPS - надстройка, исп шифрование, - 443

⁃ REST/SOAP

REST - можно кэшировать данные, wsdl - язык описания веб-сервисов и доступа к ним, основанный на языке XML.

**Трехзвенная архитектура**

*Прокси-сервер* – промежуточный сервер, выполняющий роль посредника между пользователем и целевым сервером, позволяющий клиентам как выполнять косвенные запросы к другим сетевым службам, так и получать ответы. Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс, расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша.

*Ротирующий прокси* – тип прокси, который назначает новый айпи-адрес для каждой попытки подключения.

## **Как тестировали API**

При помощи Postman или Swagger. Копировали из документации URL и вставляли в Postman. Выбирали нужный метод. Вставляли тело запроса, если оно было нужно, и отправляли запрос. После чего, сверяли ответ с документацией.

# **13. Статус-код:**

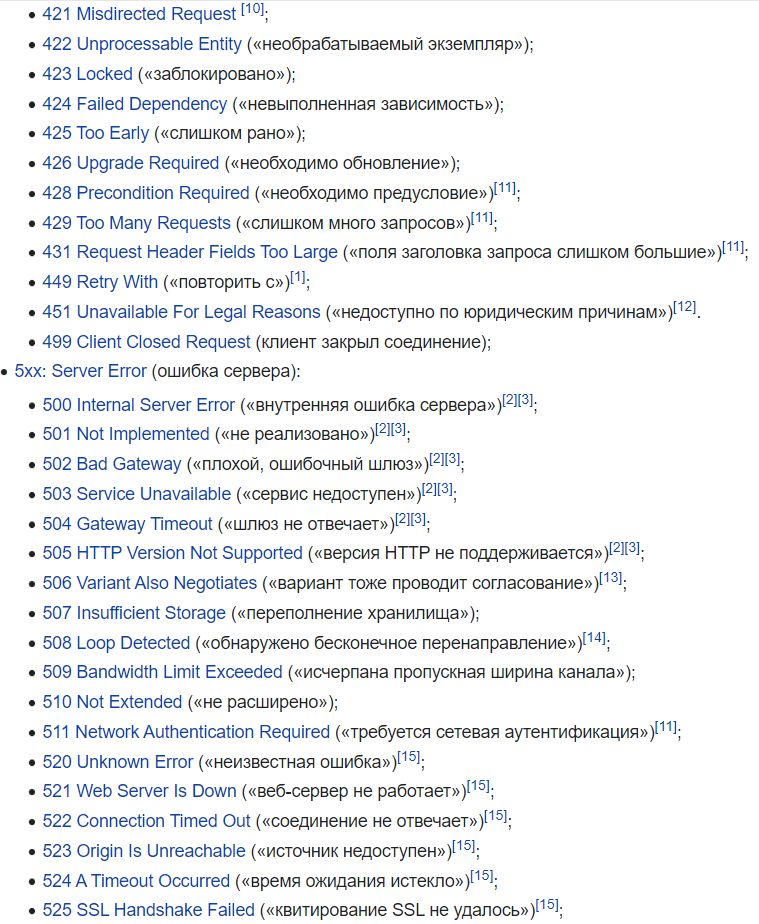
⁃ 1 - информационные

⁃ 2 - успешные

⁃ 3 - перенаправление

⁃ 4 - ошибка на стороне клиента

⁃ 5 - ошибка на стороне сервера



# **14. Заголовки http-протокола**

⁃ Основные (название метода, урл, статус код)

⁃ Запроса (информация о клиенте)

⁃ Ответа (больше и сервере и ответе)

⁃ Тело (JSON)

## **Вкладки в инструментах разработчика и для чего они нужны?**

Elements - смотрели структуру и стили страницы (цвет, размеры, отступы) Console - в этой вкладке выводились ошибки(фронтовые)  
Network - здесь отслеживали как отработали запросы  
Application - очищали кэш и куки

# **Задание на БД**

Есть база клиентов Customers, база продуктов Products, база покупок Purchase. Вывести список уникальных клиентов, который купили продукты с айди 11.

select \* from Customers where ID in (select distinct \* Customer.ID from Purchase where Product.ID = 11)

select \* from Customers join Purchase on Customers.CustomerId = Purchase.CustomerId

## **Первичный ключ или primary key**

Первичный ключ — особенное поле в SQL-таблице, которое позволяет однозначно идентифицировать каждую запись в ней.

Свойства:

* каждая запись - уникальна;
* запись в поле не должна быть пустой;
* в одной таблице может быть только один ключ.

## **Внешний ключ или foreign key**

Внешний ключ нужен для того, чтобы связать две разные SQL-таблицы между собой.