Geekbrains

**Исследование особенностей организации процесса ручного тестирования веб-приложений в условиях отсутствия технического задания на примере тестирования веб приложения Animal\_House (Питомник домашних животных).**

Программа:

Специализация: Программист

Голубев К.А.

г. Орехово-Зуево

2024 г.

**Введение.**

Сегодня есть множество фреймворков для тестирования, поддерживающих практически все существующие языки. Казалось бы — можно брать и автоматизировать. Но даже сейчас ручные тесты важны.

Одно из объяснений их необходимости заключается в том в том, что при ручном тестировании функционала мы можем гораздо быстрее получить информацию о состоянии продукта, который анализируем, о качестве разработки. Кроме того, при автоматизации предварительно разработанные кейсы часто приходится менять и актуализировать, а на написание автотестов требуется определённое время.

При этом в процессе разработки может прийти обратная связь от заказчика, когда он увидит в готовом продукте какую-то функцию, которую решит изменить до релиза — и, если вы уже подготовили для неё программные тесты, их придётся переписать. Обновление кейсов, автотестов и их проверка отнимут ценное время, которое можно было бы использовать на доработку продукта.

Всё это означает, что главная цель ручных тестов — предварительно убедиться в том, что заявленный функционал работоспособен, не имеет ошибок и выдаёт ожидаемые, запланированные результаты. Без них нельзя быть уверенным в том, что можно работать дальше. Особенно это актуально для функций, на реализацию которых завязана последующая разработка.

В дополнение к этому, на первых этапах разработки приложения автоматизация может оказаться довольно дорогой. Вам потребуется специалист, обладающий специфической квалификацией (и, возможно, не один). Постоянное поддержание тестов в актуальном состоянии требует затрат ресурсов вплоть до релиза фичи. А месяцы простоя, посвященные усовершенствования автотеста ударят по мотивации команды.  
  
 Если вы хотите регулярно добавлять новый функционал и успевать за действиями конкурентов, то перед тем, как создавать автотесты всегда проверяйте возможности продукта вручную. Просто, потому что ручное тестирование ускоряет ваши процессы.

**Глава 1. Основы ручного тестирования веб-приложений**

**1.1 Что такое веб-приложение, его особенности**

В настоящее время Web приложения являются наиболее интенсивно используемыми программными продуктами. Все большее число компаний используют Web приложения для предоставления своих товаров и услуг в сети Интернет. Ошибка, допущенная в таком приложении, может нанести не только серьезный удар по репутации компании, но и стоить ей немалых денег. В то же время, тестирование Web приложений является сложной и трудоемкой задачей.

В чем заключается сложность Web приложений как объектов тестирования? Web приложения включают сложные и неоднородные программные продукты, а именно программы дизайна, скрипты для обеспечения сетевых взаимодействий, серверы баз данных и так далее. Особенности тестирования их заключаются в их очень частой модификации, в интенсивных нагрузках со стороны пользователей. К особенностям Web приложений как объектов тестирования следует еще отнести их динамический характер, так как многие объекты Web страниц формируются в процессе функционирования таких приложений. Все эти особенности обуславливают повышенные требования к системам их тестирования и, в частности, регрессионного тестирования.

Веб-приложения с нами надолго: это удобно и для заказчика, и для разработчика, и для пользователя. А в России, с исчезновением многих приложений из мобильных сторов, создание веб-приложений становится всё более популярным выходом из положения.

Веб-приложение – это важное проявление взаимодействия клиента с бизнесом, и просто решения задач клиента недостаточно для того, чтобы претендовать на высокое качество. Клиент выбирает компанию, основываясь на своём пользовательском опыте – удобно ли ему было решить свою задачу? Столкнулся ли он с «изнанкой» веб-приложения в виде некорректно обработанного запроса или отсутствия обработанного сообщения об ошибке? Всё это составляет портрет компании с точки зрения клиента, а в современных реалиях – веб-приложение может быть единственной точкой касания между клиентом и поставщиком товаров или услуг. Поэтому такие сервисы нуждаются в тщательном тестировании и дополнительной «полировке» — в конечном счете, пользователю должно быть комфортно работать как с мобильными приложениями, так и с сайтами и веб-версиями сервисов.

## **Структура веб-приложений**

Веб-приложение, или веб-сайт, представляет собой пакет, загруженный на клиентской стороне и содержащий множество слоев взаимозависимых модулей, построенных на основе веб-технологий:

* *HTML* – язык разметки гипертекста, представляющий основанную на тегах систему разметки документов, определяющую структуру и отдельные компоненты того, что в итоге компилируется в объектную модель документа, или DOM.
* *CSS* – каскадная таблица стилей, являющаяся фреймворком для стилизации приложения, с помощью которого идентифицируют и оформляют различные части DOM видимой области страницы. CSS предоставляет возможности вроде выбора элементов по их ID, классу и отношению к другим элементам DOM.
* *JavaScript* – высокоуровневый интерпретируемый скриптовый язык, с помощью которого прописывается и выполняется все поведение приложения.

Функциональность этих основных технологий дополнительно расширяется слоями других технических решений, диапазон которых поистине огромен. Сюда относятся различные JS-фреймворки вроде Angular, React и Vue, CSS-инструменты предварительной/пост-обработки вроде Less и Sass, а также всевозможные механизмы HTML-шаблонизации. Помимо слоя фронтенда в большинстве приложений также присутствует серверная сторона, или бэкенд, с API, построенным на микросервисах и базах данных. Серверная часть содержит все данные и бизнес-логику, абстрагируя всю эту информацию в соответствующие контракты, к которым фронтенд может обращаться через HTTP-методы, используя нужную форму запроса и учетные данные.  
 В зависимости от применяемых инструментов и внутренней сути сайта ему потребуется подходящая стратегия хостинга и инфраструктура. Сайты могут размещаться на различных системах, которые в общем можно разделить на два вида:

* *Статические веб-хосты*: используются для статических сайтов и представляют собой платформы доставки на базе хранилищ файлов, предоставляющие домены, электронные адреса, DNS и прочие возможности вроде SSL, шифрования и интеграции сторонних сервисов. В качестве примеров можно привести Godaddy, Hostinger и прочие.
* *Динамические веб-хосты*: служат для размещения динамических веб-приложений. К наиболее популярным относятся облачные платформы вроде AWS, Google Cloud, Azure, Salesforce и IBM cloud, которые предоставляют в качестве сервисов различные вычислительные возможности, включая виртуальные машины, базы данных, масштабирование ресурсов по требованию и т.д. Эти платформы являются стандартом для развертывания веб-приложений в бизнес-целях ввиду их высокой производительности и безопасности. Кроме того, они предлагают к использованию передовые технологии ИИ и машинного обучения.

Веб-приложение, даже при небольшой начальной конфигурации по мере добавления в него страниц, контента и функциональности будет постепенно разрастаться. И однажды, когда будет достигнут определенный порог сложности, значительно затруднится управление приложением, а также отслеживание потребления и выделения ресурсов.  
 В зависимости от типа создаваемого приложения разработчик может использовать для него различную структуру:

* *Одностраничное приложение*. Состоит из одной модели документа, в рамках которой реализуется вся функциональность. Такие проекты задействуют большой объем функциональной логики, упаковываемой и отправляемой на компьютер клиента с соответствующими оптимизациями безопасности и производительности. В качестве известных примеров можно назвать Gmail, Facebook, GitHub и пр.
* *Многостраничное приложение*. Наиболее частый выбор, при котором приложение разделяется на множество страниц, доступных по различным путям URL. Создаются такие приложения с помощью серверных фреймворков и механизмов шаблонизации. При этом они имеют характерное преимущество в безопасности перед одностраничными решениями.
* *Прогрессивное веб-приложение*. Это современный способ применения [WebView](https://developer.android.com/reference/android/webkit/WebView" \t "_blank) на мобильных устройствах для выполнения веб-приложений в качестве нативных, используя service worker, манифесты и оболочку.

Хотя какой бы подход вы не выбрали, для обеспечения эффективности рабочего потока DevOps в каждом случае важно будет реализовать план тестирования как на уровне разработки, так и на уровне эксплуатации приложения.

**1.2 Что такое тестирование, кто такой тестировщик. Зачем нужно тестирование и когда оно начинается**

Тестирование программного обеспечения -это метод проверки соответствия фактического программного продукта ожидаемым требованиям, который также необходим, чтобы убедиться, что продукт не содержит дефектов. Подразумевает выполнение предварительно определенных алгоритмов с использованием ручных или автоматизированных инструментов для оценки одного или нескольких интересующих свойств. Целью тестирования ПО является выявление ошибок, пробелов или отсутствующих требований, заданных на этапе проектирования продукта.

Некоторые предпочитают называть тестирование программного обеспечения:  
**«Белый ящик»** – когда у нас есть доступ к коду, и мы его тестируем, читаем сам код (статическое тестирование), запускаем в дебаге, пишем автотесты;  
**«Черный ящик»** – когда мы не знаем, как система устроена внутри, нет доступа к коду или мы не умеем его читать, и поэтому ориентируемся только на внешнее поведение или ТЗ;  
**«Серый ящик»** – когда мы смотрим в код и понимаем, как он устроен, а потом открываем само приложение и проверяем, как этот код отображается уже в нем, но ориентируемся уже больше на ТЗ (это компиляция двух вышеприведенных определений).

Проще говоря, тестирование подразумевает проверку приложения на соответствие требованиям и стандартам качества.

**Преимущества тестирования программного обеспечения**  
 Специалисты выделяют следующие преимущества использования тестирования программного обеспечения:

* *Рентабельность*  
  Одно из ключевых преимуществ тестирования. В случае, если ошибки были обнаружены на раннем этапе тестирования программного обеспечения, их исправление обходится дешевле. Своевременное тестирование любого ИТ-проекта поможет вам сэкономить деньги в долгосрочной перспективе.
* *Безопасность*  
  Люди ищут проверенные продукты. Тестирование избавиться вас от рисков утечки информации.
* *Качество продукта*  
  Главная цель любого продукта – принести своим клиентам пользу. Тестирование гарантирует, что продукт будет выполнен в соответствии с бизнес-требованиями и выполнять свои функции.
* *Удовлетворенность клиентов*  
  Если пользователь не является ярым фанатом определенного бренда, при выборе из нескольких схожих продуктов, пользователь выбирает тот, который удобнее. UI / UX Testing обеспечивает лучший пользовательский опыт.

Перед тем как человек начнет пользоваться новой версией компьютерной программы, сайта или мобильного приложения, продукт должен быть проверен инженерами-тестировщиками. Они отыскивают слабые места в коде, из-за которых программа начинает работать неправильно. Для этого тестировщики создают различные ситуации, при которых возможно возникновение ошибок.

Ручной метод является наиболее сложным, так как специалисту необходимо настраивать среду и проводить тесты. Плюс ко всему, нельзя забывать о человеческом факторе: тестировщик может ошибиться или пропустить ту или иную стадию тестового скрипта.

Тестировщики, инженеры по тестированию, или QA-инженеры, — это специалисты, которые проверяют, как работает программа или приложение. Они ищут ошибки, смотрят, чтобы программа делала именно то, что задумано, защищают продукт от хакеров, проверяют, как приложение работает на разных устройствах и в разных операционных системах.

Тестировщик должен скрупулёзно проверить, что будет, если совершить какое-то действие, и соответствует ли результат ожидаемому.

Но ошибки коварны — в лабораторных условиях они могут себя и не проявить, зато пользователь продукта наткнётся на них после релиза. Поэтому QA-инженеры придумывают разные стандартные и нестандартные ситуации, чтобы проверить, как поведёт себя программа. Это творческая профессия, для поиска скрытых багов приходится проявлять смекалку и аналитические способности.

Нужно предусмотреть все возможные и невозможные варианты. Что будет, если пользователь введёт цифры вместо букв? А что, если у него компьютер глючный? А вдруг сразу много пользователей будут работать и перегрузят систему?

Работа тестировщика — это бесконечное приближение к совершенству. Невозможно проверить всё, поэтому тестировщик должен работать так, чтобы совершить минимум действий, но найти максимум ошибок.

Всех тестировщиков делят на мануальщиков и автотестеров. Первые ищут ошибки в коде вручную, вторые пишут программы, которые делают это автоматически. Но это очень условное деление. Вы не найдёте автоматизатора, который не умел бы тестировать программу вручную. А многие мануальные тестировщики не любят рутину, поэтому осваивают языки программирования и пишут автоматические тесты. Опытные специалисты могут выполнять работу и мануальщика, и автотестера.

При этом все тестировщики используют разные инструменты для автоматизации, управления тестовым процессом и обеспечения качества продукта.

**Ручное тестирование** – это вид тестирования программного обеспечения, при котором тестовый пример выполняется вручную тестировщиком без помощи каких-либо [автоматизированных инструментов](https://www.zaptest.com/what-is-test-automation-a-no-jargon-simple-guide).

Компании используют ручное тестирование как метод выявления ошибок или проблем в своем программном обеспечении, так же соответствовало ожиданиям пользователей и работало правильно. Оно позволяет тестировщикам оценивать пользовательский опыт, выявлять визуальные и функциональные проблемы и обеспечивать ожидаемое поведение систем в различных схемах, которые могут не охватываться управляемыми тестами.

Ручное тестирование часто используют на ранних стадиях разработки и исследовательского тестирования, где решающее значение имеют человеческая интуиция и способность к адаптации.

**Когда необходимо проводить ручное тестирование?**

Существует несколько этапов, на которых разработчики используют ручное тестирование, первый из них – на этапе разработки базовой функциональности.

Когда основная функциональность программного обеспечения находится в разработке, разработчики тестируют работу каждой части программы вручную, так как это быстрее, чем создавать тестовые примеры для достаточно простых частей кода.

Ручное тестирование также преобладает на последних этапах разработки, когда в программе создан пользовательский интерфейс. [Тестирование пользовательского интерфейса](https://www.zaptest.com/what-is-ui-software-testing-deep-dive-into-the-types-process-tools-implementation) включает в себя наблюдение за тем, как реальный пользователь реагирует на то, как разработаны меню и как работает система.

Поскольку в этом случае используется много качественных данных и личного мнения, а не чистых количественных показателей, ручное тестирование является идеальным вариантом для получения более глубокого представления о продукте.

### **Когда вам не нужно проводить ручное тестирование**

### Есть несколько случаев, когда использование ручного тестирования потребует гораздо больше времени и усилий, чем необходимо, и первый из них – это тестирование баз данных.

Базы данных обрабатывают огромные объемы данных, и их ручной ввод занял бы много времени и был бы неэффективным для организации. В таких случаях идеальным вариантом является использование автоматизированных систем, поскольку они могут обрабатывать большие пакеты данных за ограниченное время.

Ручное тестирование также менее полезно в таких областях, как [нагрузочные тесты](https://www.zaptest.com/what-is-load-testing-deep-dive-into-the-types-practices-tools-challenges-more), когда разработчик проводит тестирование, чтобы увидеть, как его программное обеспечение справляется со значительной нагрузкой пользователей.

Это часто касается онлайн-приложений и программ с серверами, которые требуют тщательной оценки. Проведение ручного тестирования требует одновременного доступа к приложению большого количества людей, что может привести к значительным трудозатратам на услуги, которые могут быть выполнены автоматизированной системой тестирования программного обеспечения с гораздо меньшими затратами.

**Преимущества использования ручных тестов:**

### **1. *Базовая функциональность***

Одна из самых ранних частей процесса тестирования программного обеспечения рассматривает базовую функциональность части программного обеспечения.

На этом этапе разработчик или тестировщик просматривает один из функциональных модулей кода и оценивает, работает ли он так, как ожидалось. Из-за небольшого масштаба этих модулей стоит сосредоточиться на ручном тестировании, поскольку автоматизация займет слишком много времени.

В качестве примера можно привести программное обеспечение для работы с базами данных: тестировщики вводят часть данных в функцию и уже знают ожидаемый результат.

Если эти два значения совпадают, тест успешно завершен. Тестирование на этом этапе процесса закладывает прочный фундамент для остальной работы компании.

### **2. *Дизайн пользовательского интерфейса***

Пользовательский интерфейс относится к пользовательскому интерфейсу программного обеспечения, или к меню, кнопкам и интерактивным возможностям, доступным пользователю.

Тестирование пользовательского интерфейса сосредоточено на том, как работает пользовательский интерфейс и удобен ли он для пользователя, в том числе может ли пользователь взаимодействовать со всеми функциями и эстетично ли меню.

Ручное тестирование на этом этапе необходимо, поскольку качественная информация, например, о том, хорошо ли выглядят интерфейсы, не является тем, в чем преуспела автоматизированная программа.

### **3. *Тестирование на проникновение***

Тестирование на проникновение – это тестирование программного пакета на предмет того, насколько легко внешняя сторона может получить доступ к программному обеспечению незаконными средствами.

Автоматизация программного обеспечения сосредоточена на выполнении нескольких конкретных шагов и завершении процессов, которые уже являются частью приложения, а не на исследовании новых областей, что является обязательным условием для тестирования безопасности. Например, компания может нанять этичного хакера для оценки своего программного обеспечения и поиска любых возможностей для доступа злоумышленников к данным пользователей.

### **4. *Исследовательское тестирование***

Эксплораторное тестирование относится к тестированию, которое необходимо провести только один или два раза, получив такое название, поскольку оно является частью “исследования” программного обеспечения на предмет наличия неожиданных функций или ошибок.

Ручное тестирование лучше подходит для этого случая, поскольку написание кода для тестового случая занимает время, а если кто-то вручную зайдет в программное обеспечение и изучит его, то это займет меньше времени.

Примером может служить ситуация, когда разработчик хочет проверить, правильно ли интегрирована определенная функция, при этом один тест проверяет, что данные правильно перемещаются по программе.

**1.3 Этапы ручного тестирования веб-приложений**

Если смотреть на процесс разработки в целом и на тестирование как на одну из его частей, то при грамотном планировании вы всегда будете понимать, что и когда будет готово. Это позволит лучше планировать время тех или иных действий — поскольку одни события логично следуют за другими и у вас есть возможность выстраивать их в цепочки на основании своих ожиданий.

**Тестировщик появляется в процессе создания приложения уже на ранних этапах.** Вот у клиента появляется некая идея, бизнес-аналитики собирают из этого требования — а тестировщики уже в этот момент приступают к работе, проверяя эти требования.

**Как это происходит?** Они задают вопросы по предполагаемому функционалу. Пытаются представить, как будет выглядеть приложение, когда оно будет реализовано. Если речь идёт о новой фиче в уже существующем продукте — разбираются, как она будет взаимодействовать с существующим функционалом. После того, как разработчики провели оценку трудозатратности идей клиента и определили сколько потребуется времени на их реализацию.

**После этого начинается этап проектирования.** Здесь появляется необходимость понять, как будут осуществляться переходы между экранами, валидироваться те или иные поля, как приложение или его отдельная функция вообще будет взаимодействовать с конечным пользователем. В случае с фичами важно разобраться и с тем, как они будут входить в архитектуру существующего приложения.

**На этапе дизайна**, когда создаётся карта переходов между экранами, тестировщик уточняет поведение отдельных элементов, из которых они состоят и то, какими визуальными эффектами будут сопровождаться те или иные действия пользователя. Кроме этого тестировщик валидирует макеты на полноту, подтверждая, что они отображают всё, что нужно для реализации задуманного функционала. Нелишней будет и самостоятельная проверка макетов на соответствие гайдлайнам.

**С началом разработки QA-специалисты сразу же приступают к написанию тест-кейсов.**На разных этапах разработки они могут обладать различным уровнем детализации, но к её окончанию желательно обеспечить максимальное покрытие кейсами всего продукта, чтобы, получив сборку, можно было приступить к тестированию немедленно.

**Первый шаг непосредственно тестирования — смоук-тест:** оценка на то, что приложение или его новая часть вообще готовы к проверке. Смоук-тест — это проверка того, запускается ли приложение или конкретная функция в принципе. Смоук-тест — быстрый способ убедиться, можем ли мы вообще начать функционально тестирование. Термин пришел от создателей плат и микросхем, которые для начала должны были убедиться не сгорит ли новая схема — отсюда и название: задымилась/не задымилась.  
Это быстрый способ убедиться в том, что нам действительно сдали задачу и её можно принимать в работу, а не отправлять обратно программистам.

На примере формы авторизации смоук — это оценка того, можно ли залогиниться вообще, без уточнения того, валидны ли данные, вводимые в поля, работают ли дополнительные возможности вроде напоминания пароля и прочего. Если мы смогли авторизоваться в принципе — можно переходить к дальнейшей, углублённой проверке этого модуля: брать негативные кейсы, пограничные значения, оценивать соответствие установленным правилам валидации.

**Следующий этап — проведение регресс-тестов.**В ручном или автоматическом режиме проводится основной заранее запланированный массив тестов. Регрессионное тестирование хорошо тем, что оно позволяет найти ошибки даже в тех местах, где раньше всё было в порядке. Это происходит благодаря тому, что регресс — это оценка функционала на стандартный набор кейсов при внедрении каждого нового модуля и каждого изменения приложения. Ведь, когда разработчики добавляют новый функционал может быть повреждена текущая версия или новая фича может вступать в конфликт с уже существующими.

Например, добавление новых экранов, а значит и изменение навигации может нарушить функционирование меню или, как минимум — его отображение. С другой стороны, неприятные сюрпризы может принести и глобальный рефакторинг кода приложения — после него тоже необходимо проводить регресс-тесты. Проблемы могут вызвать и обновление используемой приложением библиотеки и изменение среды в которой оно работает. Чем чаще вы обновляете приложение — тем более важную роль играют регрессионные проверки. Не стоит ограничиваться однократной проверкой, проводимой, когда фича уже внедрена — проверяйте все изменения. Здесь вам поможет автоматизация регрессионного тестирования — просто потому, что ручное регрессионное тестирование новой фичи, создававшейся всего неделю может занять две, а то и больше и отдел тестирования просто завязнет в этом.

**Завершается всё тестированием финальной сборки** — release candidate. В него входят проверка бета-версии внутренними тестировщиками, бизнес-тестирование — оценка получившегося продукта самим клиентом и получение от него обратной связи, а также предложение определённой группе пользователей познакомиться с предрелизной альфа-версией приложения или его новых возможностей. После этого приложение готово к тому, чтобы его выкатили в продакшн.

**Но на этом работа QA-специалиста не заканчивается** — ему предстоит тестировать обновления приложения и их совместимость с более ранними версиями, составлять кейсы для проверки нововведений и, в случае необходимости, автоматизировать прохождение этих сценариев.

**Параллельно с этим тестировщики участвуют в дальнейшем анализе статистики**, собираемой аналитиками, мониторинге поведения приложения и того, как взаимодействуют с ним пользователи. Это позволяет не только увидеть живое использование результатов их работы, но и, порой, открыть для себя новые сценарии и неизвестные баги, вызывающие падения приложения.

Главное о ручном тестировании

1. Тестирование ПО — это проверка того, как работает программа, нет ли ошибок, вылетов, удобна ли она в использовании.
2. Поиском ошибок и проверкой функциональности программы занимаются тестировщики. QA-инженеры отвечают за качество продукта на всех стадиях его создания.
3. Для ручного тестирования нужно немного навыков — умение работать с баг-трекинговыми системами, навык создания тест-кейсов, аналитические способности. Тестировщикам и QA-инженерам еще может понадобиться знание SQL и Python.
4. Ручное тестирование имитирует действия пользователей. Тестировщики проверяют программу и выполняют реальные сценарии, чтобы найти ошибки и проблемы. Сценарии описаны в тест-кейсах.
5. Тест-кейс — это пошаговый план проверки работы программы. В него входят условия выполнения, список действий, которые нужно выполнить, и ожидаемый результат. В те же тест-кейсы тестировщики записывают фактический результат — итоги проверки.
6. Юзкейс описывает, как пользователь взаимодействует с программой для достижения цели. Его пишут еще на этапе проектирования приложения. Он не описывает ту программу, которую написали де-факто, он описывает то, что планировалось получить.
7. Ручное тестирование бывает разным: тестирование черного, белого и серого ящика, тестирование дыма и кросс-браузерное тестирование, бета-тестирование, приемочное и исследовательское тестирование, юзабилити-тестирование. Разные виды тестирования раскрывают разные потенциальные проблемы и уязвимости приложения.
8. Ручное тестирование имеет плюсы и минусы. Основной минус — тестировщики подвержены человеческому фактору: они устают, ошибаются, им может не хватать квалификации. Некоторые проблемы решаются повышением квалификации специалиста, другие — автоматизацией тестирования.

**1.4 Источники требований: что такое техническое задание и как проводить ручное тестирование, если техническое задание отсутствует**

Все удачные IT-продукты были когда-то просто идеями. И очень часто (почти всегда) идеи принадлежат людям, далеким от разработки программного и аппаратного обеспечения.

Они хотят что-то улучшить в своем бизнесе, в том направлении деятельности, которое знают досконально. Идеи, мысли, мечты – это прекрасно, но важно, чтобы специалист верно их понял и воплотил в работающее устройство или программу.

Правильно довести идею до разработки поможет техническое задание.

Техническое задание на разработку прибора или программного обеспечения – это документ, определяющий требования к IT-продукту, включая его назначение, функции, поведение, используемые компоненты, технологии, инструменты разработки, а также порядок выполнения работ. ТЗ служит руководством для бизнес- и технических групп, занимающихся созданием IT-решения.

Техническое задание в равной степени нужно и заказчику, и разработчику. Спецификация является трудом специалистов из разных областей и используется клиентом и исполнителем на протяжении всего периода разработки и после окончания проекта.

Техническое задание на проектирование устройства или написание ПО позволяет получить предварительную оценку стоимости разработки продукта

Стоимость сложного устройства или приложения невозможно оценить навскидку. Необходимо учесть множество моментов – затраты труда специалистов, стоимость компонентов и логистики, работы, связанные с сертификацией и т.д. Грамотно составленный документ позволяет и исполнителю, и заказчику видеть и оценивать как процесс разработки целиком, так и отдельные его ступени. Таким образом, заказчик получит представление о предварительной стоимости каждого этапа работ. Более точные данные будут даны в смете проекта.

В ТЗ очерчиваются примерные сроки исполнения заказа. У клиента и аутсорсинговой компании не будет разногласий по поводу тайминга, если с самого начала в документе обозначены временные отрезки для каждого этапа проекта.

Сроки выполнения работ по проектированию электроники и созданию программного обеспечения могут сдвигаться по разным причинам. Некоторые из них – например, время ожидания компонентов и сроки доставки – можно предусмотреть уже на этапе написания ТЗ.

Качественно написанное техническое задание на разработку прибора или ПО может свидетельствовать о компетенции и опыте специалистов. Вдумчивый подход разработчиков к подготовке проекта, понятная и исчерпывающая информация в ТЗ говорят об общем уровне сервиса компании.

В ТЗ описывается сам продукт, его назначение и функциональность, а также этапы разработки, основные элементы электроники и инструменты для создания ПО. Таким образом, и заказчик, и разработчик имеют полное представление о проектируемом IT-решении, что служит страховкой от разногласий, недопониманий, внесений незапланированных изменений в концепцию продукта.

Работа над проектом идет быстрее и проще, когда команда разработчиков опирается на ТЗ. Нет необходимости согласовывать каждый шаг, теряя время.

Любой, даже совсем небольшой типовой проект требует оформления спецификации – документа, где будут зафиксированы требования к разрабатываемому решению, порядок работ, используемые компоненты и т.д. Это не будет ТЗ в классическом виде, но совсем без спецификации не обойтись.

Клиент может предоставить документ, в котором в произвольной форме изложены его идеи, пожелания, видение продукта. Компетентность клиента в вопросах проектирования и программирования будет большим плюсом, но главное – четко и понятно сформулировать свои пожелания к продукту. На основе такого объяснения компания-разработчик создаст полноценное качественное ТЗ, которое будет служить ориентиром в последующей разработке.

**Что должно быть в ТЗ для разработки IT-продукта:**

### **Информация о разрабатываемом IT-продукте**

1. Назначение и цели создания
2. Структура IT-продукта

* из чего состоит продукт
* как он взаимодействует с системой в целом

### **Глоссарий**

### Он же справочник. Как правило, бизнес пользуется специфической терминологией, которая понятна заказчику или представителю сферы. Команда разработки же может не всегда верно истолковать эту терминологию, поэтому важно сверить «понятийные аппараты» и говорить на одном языке. Также и со стороны разработчиков необходимо описывать технические термины, чтобы сотрудник Заказчика, согласовывающий ТЗ (при этом чаще всего не технический специалист) понимал, о чем идет речь

### **Требования к IT-продукту**

1. Бизнес-требования
2. Функциональные требования
3. Технические требования:

* характеристики (нагрузка, которую должен выдержать сервис, производительность и т.д.)
* требования к технологиям
* требования к серверам
* требования к скорости работы сервиса
* обязательная интеграция со сторонними сервисами
* требования к безопасности и прочее. Таких требований может быть очень много, все зависит от того, какой продукт вы разрабатываете

### **Спецификация на интерфейс**

Этот пункт присутствует в ТЗ на разработку IT-продукта при реализации по прототипам или макетам.

* Спецификация — это документ, который описывает функциональное назначение и тип каждого элемента управления/блока, техническую составляющую интерфейса. Как правило, спецификация составляется по типу: Элемент-Изображение-Описание.

### **Пользовательские сценарии (use case)**

### Use case — это ответ системы на действия пользователя. Соответственно, чем больше пользовательских сценариев и реакций системы будет описано в ТЗ, тем лучше.

Пользовательские сценарии помогают всем участникам системы, в том числе тимлиду:

* Определить появление возможных проблем во время разработки
* Четко определить и предсказать поведение системы
* Упрощает приемку задачи, дает совпасть ожиданию и реальностиДает однозначность: упрощает жизнь для разработчика, тестировщика, постановщика

Возможно ли тестировать без требований? Нет! Потому что именно они определяют, что должен представлять собой тот или иной продукт, и без них он фактически не может быть создан.

Распространенные возражения, как правило, сводятся к двум пунктам:

* У нас нет ТЗ, но проект-то есть, и тестирование ведется.
* Мы работаем по agile — функциональный продукт важнее документации, которая бы описывала его исчерпывающим образом.

Однако в любом случае необходимо понимать, кто будет пользоваться продуктом, как он должен выглядеть, из чего состоять и какими обладать функциями. Несмотря на то что эта информация не содержится в спецификациях, в ней как раз и заключены требования к ПО.

**Глава 2. Подготовка к ручному тестированию веб-приложения**

**2.1 Составление чек-листов**

Подробное описание шагов поможет не пропустить ничего, особенно таких пунктов, до которых обычный тестировщик может и не добраться, к примеру, наличие у сайта favicon.

Базовый чек-лист позволит провести [web-тестирование](https://qaservicelab.ru/web/), которое отвечает минимальному набору требований для того, чтобы его приложение было назвать качественным:

* Сайт доступен и открывается
* Доступно открытие сайта в режиме Инкогнито
* Можно нажать все кнопки на сайте
* Отсутствуют битые ссылки
* Ссылки с заголовками соответствуют заголовкам
* Все формы отправляют запросы и отображают пользователю результаты в соответствии с ожидаемыми
* Для полей форм настроена валидация
* Ввод данных соответствует валидации
* Для обязательных полей есть метка обязательного поля (звездочка)
* Отправка формы без значения необязательного поля недоступна
* Развертывание скрытых элементов (попапы, списки) не вызывают блокирование функционала и друг друга
* Проверить сквозные разделы сайта (хэдер, меню, футер). Убедиться в том, что в разных разделах сквозные элементы отображаются и функционируют соответственно ожидаемому результату
* Если есть поля для загрузки файлов – проверить загрузку, отображение, изменение и удаление файлов для всех валидных форматов
* Если есть логика выгрузки файлов – проверить выгрузку файлов, соответственно связанному функционалу
* Для фильтров – проверить выборку для каждого фильтра отдельно и их комбинаций
* Проверить, что будет при удалении cookie, находясь на сайте

**2.2 Составление тест-кейсов**

Тест-кейс — это документ с пошаговыми инструкциями для проверки работы программы.

Такой документ организовывает сам процесс тестирования: вместо того чтобы наугад выполнять какие-то действия, которые может совершить пользователь программы, у тестировщика перед глазами список действий для проверки и вся необходимая информация.

Обычно тест-кейс содержит:

* предварительные условия — например, заходить из российского сегмента интернета;
* входные данные — в данном случае это была ссылка;
* действия, которые нужно выполнить;
* ожидаемый результат;
* фактический результат.

Тест-кейсы можно разделить на несколько типов:

* позитивные тест-кейсы — проверяют, как система работает при правильных данных;
* негативные тест-кейсы — проверяют, как система реагирует на неправильные или неожиданные данные;
* деструктивные тест-кейсы — ищут ошибки при неправильном использовании системы.

Цель тест-кейса — убедиться, что программа работает правильно и соответствует требованиям. Их нужно отличать от юзкейсов.

**2.3 Разработка тест-плана**

Разработка тестового плана – важный этап подготовки к ручному тестированию любого веб-приложения. Этот план помогает структурировать работу, определить приоритеты и ресурсы, а также обеспечить систематический подход к процессу тестирования. Ниже приведены основные шаги по разработке тест-плана и примеры их применения.

Шаг 1: Определение целей тестирования

Цели тестирования определяют, что именно необходимо проверить и какой результат ожидается после завершения тестирования. Примером целей тестирования могут быть:

* Проверка корректной работы функционала приложения.
* Обеспечение безопасности данных пользователей.
* Оценка производительности приложения при высокой нагрузке.
* Убедиться в совместимости с различными браузерами и устройствами.

Пример: для веб-приложения электронной коммерции целью может быть проверка корректности обработки заказов и платежей.

Шаг 2: Идентификация рисков и приоритетов

На данном этапе определяются потенциальные риски, связанные с работой приложения, и устанавливаются приоритеты для различных аспектов тестирования. Риски могут включать:

* Потенциальные уязвимости безопасности.
* Проблемы с производительностью.
* Некорректную обработку данных.
* Неправильное отображение контента на разных устройствах.

Пример: при тестировании веб-приложения для онлайн-банкинга высокий приоритет будет отдан проверке безопасности транзакций и защиты персональных данных.

Шаг 3: Выбор стратегии тестирования

Стратегия тестирования определяет, какие виды тестирования будут использоваться и в каком порядке. Возможны следующие типы стратегий:

* Функциональное тестирование.
* Нефункциональное тестирование (производительность, безопасность, удобство использования).
* Интеграционное тестирование.
* Системное тестирование.
* Приемочное тестирование.

Например, для нового веб-приложения социальной сети стратегия может включать сначала функциональное тестирование базовых функций (регистрация, авторизация), затем интеграционное тестирование взаимодействия с другими сервисами (например, интеграция с мессенджерами), и завершится приемочным тестированием с участием реальных пользователей.

Шаг 4: Составление списка тестовых случаев

Тестовые случаи представляют собой конкретные шаги, которые необходимо выполнить для проверки определенного аспекта функционирования приложения. Список тестовых случаев должен охватывать все функциональные и нефункциональные требования.

Пример:

* Функциональный тестовый случай: проверить регистрацию нового пользователя.
  + Открыть страницу регистрации.
  + Заполнить форму регистрации.
  + Нажать кнопку "Зарегистрироваться".
  + Проверить, что пользователь успешно зарегистрирован и перенаправлен на главную страницу.
* Нефункциональный тестовый случай: Проверка скорости загрузки страницы.
  + Измерить время загрузки главной страницы приложения.
  + Сравнить полученные данные с установленными требованиями к производительности.

Шаг 5: Назначение ресурсов и сроков

Определите, кто будет выполнять тесты, сколько времени потребуется на каждый этап тестирования и какие инструменты будут использоваться. Также учитывается необходимость привлечения дополнительных специалистов (например, экспертов по безопасности или UX/UI-дизайнеров).

Для более четкого понимания, к примеру: на тестирование веб-приложения отводится две недели. За этот период два тестировщика выполнят функциональное и интеграционное тестирование, а затем проведут приемочные испытания с участием трех фокус-групп пользователей.

Шаг 6: Документирование тест-плана

Создайте документ, содержащий всю информацию о тест-плане. Он должен включать:

* Цели тестирования.
* Стратегию тестирования.
* Список тест-кейсов.
* Ресурсы и сроки.
* Критерии приемки.

Пример структуры документа:

# Тест-план для веб-приложения X

## 1. Введение

Описание проекта и его целей.

## 2. Цели тестирования

Перечислить цели тестирования.

## 3. Риски и приоритеты

Определение потенциальных рисков и установление приоритетов.

## 4. Стратегия тестирования

Описание выбранной стратегии тестирования.

## 5. Тестовые случаи

Список тестовых кейсов с описанием каждого шага.

## 6. Ресурсы и сроки

Назначение ответственных лиц и указание временных рамок.

## 7. Критерии приемки

Критерии, по которым будет оцениваться успешность тестирования.

Заключение

Разработка тест-плана – ключевой шаг в подготовке к ручному тестированию веб-приложений. Четко определенный и хорошо структурированный тест-план поможет эффективно организовать процесс тестирования, снизить риски и обеспечить высокое качество конечного продукта.

**2.4 Тест-дизайн и тест-аналитика**

Тест-дизайн играет важную роль в обеспечении качества программного обеспечения, позволяя создать набор тестов, которые максимально полно проверяют функциональность и надежность веб-приложения. Тест-аналитика же помогает оценивать эффективность проведенных тестов и выявлять области, требующие дополнительного внимания. Рассмотрим эти процессы подробнее.

Тест-дизайн заключается в разработке и реализации тестовых сценариев, которые позволят убедиться в правильности работы всех компонентов веб-приложения. Этапы тест-дизайна включают:

1. Анализ требований: перед началом проектирования тестов необходимо изучить доступные требования к проекту, будь то техническое задание, спецификация или пользовательские истории. Если ТЗ отсутствует, как в нашем случае, тестировщик опирается на собственные знания и опыт, а также на взаимодействие с командой разработки.
2. Идентификация тестируемых областей: определите, какие части приложения требуют наибольшего внимания. Это могут быть критические функции, новые или измененные модули, а также те компоненты, которые вызывают наибольшее количество ошибок.
3. Разработка тестовых случаев: Создавайте детальные тестовые случаи, включающие входные данные, ожидаемые результаты и шаги выполнения. Важно учитывать как позитивные, так и негативные сценарии, чтобы проверить все возможные пути выполнения кода.
4. Приоритизация тестовых случаев: установите приоритеты для тестовых случаев, основываясь на рисках и важности каждой функции. Высокоприоритетные тесты выполняются первыми, чтобы быстро обнаружить серьезные дефекты.
5. Автоматизация тестов: рассмотрите возможность автоматизации некоторых тестов, особенно тех, которые повторяются многократно или занимают значительное время при выполнении вручную. Автоматизация может существенно ускорить процесс тестирования и уменьшить вероятность человеческих ошибок.

Пример тестового случая

Рассмотрим пример тестового случая для веб-приложения электронной коммерции:

Название теста: Проверка добавления товара в корзину.

Шаги выполнения:

1. Перейти на страницу каталога товаров.
2. Выбрать товар и нажать кнопку "Добавить в корзину".
3. Проверить, что товар добавлен в корзину (количество товаров в корзине увеличилось на 1).
4. Проверить правильность отображения суммы заказа в корзине.

Ожидаемый результат:

* Товар успешно добавлен в корзину.
* Сумма заказа обновлена и соответствует цене выбранного товара.

Тест-аналитика

После выполнения тестов наступает этап анализа результатов, известный как тест-анализ. Этот процесс включает:

1. Сбор данных: Собирайте информацию обо всех выполненных тестах, включая результаты, обнаруженные дефекты и затраченное время.
2. Анализ дефектов: Проанализируйте обнаруженные ошибки, определив их серьезность и влияние на работу приложения. Классифицируйте дефекты по категориям (функциональные, интерфейсные, производительность и т.д.).
3. Оценка покрытия тестов: оцените, насколько полно выполнены запланированные тесты и какие области остались не протестированными. Это поможет выявить пробелы в тестовом покрытии и скорректировать дальнейшие действия.
4. Генерация отчетов: Подготовьте отчеты о результатах тестирования, включив информацию о количестве выполненных тестов, обнаруженных дефектах, времени выполнения и общем статусе тестирования.
5. Планирование последующих действий: на основе анализа результатов определите, какие дополнительные тесты необходимы, какие дефекты требуют исправления и какие изменения следует внести в процесс тестирования.

Инструменты для тест-анализа

Существует множество инструментов, которые помогают в сборе и анализе данных тестирования. Некоторые из них:

* TestRail: Система управления тестовыми случаями и результатами, позволяющая отслеживать прогресс тестирования и генерировать отчеты.
* Jira: Популярный инструмент для управления задачами и дефектами, который интегрируется с многими инструментами тестирования.
* Allure Report: Инструмент для генерации красивых и информативных отчетов о результатах автоматизированного тестирования.

В итоге получается, что тест-дизайн и тест-аналитика также играют ключевую роль в обеспечении высокого качества веб-приложений. Грамотный дизайн тестов позволяет максимально полно проверять функциональность и надежность системы, а аналитика помогает оценить эффективность тестирования и выявить области, требующие дополнительного внимания. Использование современных инструментов и методик делает этот процесс более эффективным и управляемым.

**2.5 Баг-репорты и багтрекинговая система**

Баг-репортинг — это процесс документирования и передачи информации о найденных дефектах в программном обеспечении. Этот процесс важен для того, чтобы разработчики могли своевременно исправить ошибки и улучшить качество продукта. Хорошо составленный баг-репорт содержит всю необходимую информацию для быстрого и эффективного устранения проблемы.

Основные элементы баг-репорта:

1. Заголовок: Краткий и точный заголовок, который ясно описывает суть проблемы. Заголовок должен быть достаточно информативным, чтобы другой человек мог сразу понять, о чем идет речь, не читая весь репорт.

*Пример:* «Ошибка при создании нового класса для объекта»

1. Описание: Подробное описание проблемы, включающее контекст, в котором она возникла, и шаги для воспроизведения. Чем больше деталей предоставлено, тем легче будет разработчику понять и воспроизвести проблему.

*Пример:* при попытке создать новый объекту с использованием кнопки "Добавить новое животное" происходит ошибка. Вместо перехода к форме создания пользователь видит сообщение об ошибке "Не удалось обработать запрос". Проблема воспроизводится в Google Chrome и Microsoft EDGE.

1. Шаг 1, Шаг 2, ...: Последовательность шагов, необходимых для воспроизведения проблемы. Каждый шаг должен быть описан подробно, чтобы любой член команды смог повторить его.
2. Шаг 1: Перейдите на страницу товара.
3. Шаг 2: Выберите опцию "Быстрая покупка" и нажмите соответствующую кнопку.

Шаг 3: обратите внимание на появившееся сообщение об ошибке.

1. Ожидаемое поведение: Описание того, как система должна вести себя в нормальной ситуации. Это помогает сравнить фактическое поведение с ожидаемым и понять, где произошла ошибка.
2. После нажатия кнопки "Быстрая покупка" пользователь должен быть перенаправлен на форму оформления заказа.
3. Фактическое поведение: Фактическое поведение системы, которое отличается от ожидаемого. Это описание должно быть точным и объективным.
4. Появляется сообщение об ошибке, и переход на форму заказа не осуществляется.
5. Скриншоты и видео: Визуализация проблемы помогает быстрее понять, что произошло. Скриншоты или видеозаписи поведения системы облегчают восприятие и ускоряют диагностику.

Обязательно прилагается [ссылка на скриншот].

1. Версия ПО: Версия программного обеспечения, в которой был обнаружен дефект. Это важно для определения контекста и возможности воспроизведения ошибки, т.к. может так оказаться, что в более ранних версиях приложения ошибки может и не быть, поэтому возможно упущение данной проблемы.
2. Среда исполнения: Информация о среде, в которой была найдена ошибка (операционная система, браузер, устройство и т.п.). Это помогает сузить круг поиска причины проблемы.

*Пример:* ОС: Windows 10 Браузер: Google Chrome 90.0.4430.212 Устройство: Desktop PC (среда, в которой производилось тестирование полностью указывается со всеми его инструментами.)

1. Серьезность и приоритет: Оценка степени влияния дефекта на работоспособность системы и срочность его устранения. Обычно используется шкала от низкой до высокой.
   * Низкая: Дефект незначителен и не влияет на основной функционал.
   * Средняя: Дефект влияет на второстепенный функционал, но не блокирует работу системы.
   * Высокая: Дефект серьезно влияет на работу системы или блокирует основной функционал.

*Пример:*

Серьезность: Высокая (блокирует основную функцию)

Приоритет: Высокий (необходимо устранить до следующего релиза)

1. Дополнительная информация: Любая другая полезная информация, которая может помочь в диагностике и устранении проблемы. Это могут быть лог-файлы, конфигурационные файлы, консольные сообщения и т.д.

Пример баг-репорта

\*\*Заголовок:\*\* Невозможно добавить новый объект при выборе опции "Добавить новое животное".

\*\*Описание:\*\* При попытке добавить новый объект в базу данных с использованием кнопки "Добавить новое животное" происходит ошибка. Вместо перехода к форме создания пользователь видит сообщение об ошибке "Не удалось обработать запрос". Проблема воспроизводится в Google Chrome и Mozilla Firefox.

\*\*Шаг 1:\*\* Перейдите на страницу создания объекта.

\*\*Шаг 2:\*\* Выберите опцию "добавить новое животное" и нажмите соответствующую кнопку.

\*\*Шаг 3:\*\* Обратите внимание на появившееся сообщение об ошибке.

\*\*Ожидаемое поведение:\*\* После нажатия кнопки "добавить новое животное" пользователь должен быть перенаправлен на форму создания нового объекта класса Animal.

\*\*Фактическое поведение: \*\* Появляется сообщение об ошибке, и переход на форму создания нового класса не осуществляется.

\*\*Скриншот: \*\* [ссылка на скриншот]

\*\*Версия ПО: \*\* v1.2.3

\*\*Среда исполнения: \*\*

- ОС: Windows 10

- Браузер: Google Chrome Версия 130.0.6723.59 (Официальная сборка)

- Устройство: Desktop PC

\*\*Серьезность: \*\* Высокая (блокирует основную функцию)

\*\*Приоритет: \*\* Высокий (необходимо устранить до следующего релиза)

\*\*Дополнительная информация: \*\* Лог-файл прикреплен к задаче.

Бугтрекинг-система

Бугтрекинговая система — это программное обеспечение, предназначенное для управления жизненным циклом дефектов, начиная от их обнаружения и заканчивая устранением. Она предоставляет централизованное хранилище для всех зарегистрированных дефектов и позволяет отслеживать их текущее состояние.

Основные функции багтрекинговой системы:

1. Хранение баг-репортов: Все зарегистрированные дефекты хранятся в одном месте, что упрощает их поиск и доступ к ним.
2. Управление статусами: Возможность отслеживать текущий статус каждого бага (новый, в работе, закрыт и т.д.), что помогает контролировать процесс устранения дефектов.
3. Коммуникация: Средства для общения между тестировщиками, разработчиками и менеджерами проектов. Комментарии и обсуждения внутри системы позволяют уточнять детали и координировать работу.
4. Фильтрация и поиск: Поиск и фильтрация багов по различным параметрам (приоритет, версия, ответственный и др.) для удобного доступа к нужной информации.
5. Отчеты и метрики: Генерация отчетов и сбор статистики для анализа эффективности процесса тестирования. Это помогает принимать обоснованные решения и улучшать рабочие процессы.

Популярные бактрекинг-системы

1. Jira: Один из самых популярных инструментов для управления проектами и багами. Поддерживает гибкую настройку рабочих процессов и интеграцию с множеством сторонних сервисов.
2. Trello: Простой и интуитивно понятный инструмент для управления задачами и баг-листами. Подходит для небольших команд и проектов.
3. Redmine: Бесплатная и открытая платформа для управления проектами, поддерживающая багтрекинг и множество плагинов для расширения функциональности.
4. Bugzilla: Открытый и бесплатный багтрекер, широко используемый в сообществе open-source проектов.

Процесс работы с багтреком

1. Создание бага: Тестировщик создает новый баг-репорт в системе, заполняя все необходимые поля. Это первый шаг в процессе управления дефектами.
2. Назначение задачи: Менеджер проекта или руководитель команды назначает задачу на разработку исправлений конкретному разработчику. Это обеспечивает ответственность и контроль за процессом устранения дефектов.
3. Исправление бага: Разработчик работает над исправлением дефекта, периодически обновляя статус бага в системе. Это позволяет другим членам команды следить за ходом работ и вовремя реагировать на изменения.
4. Проверка исправления: после внесения изменений тестировщик повторно проверяет функционал, чтобы убедиться, что проблема устранена. Если дефект действительно устранен, баг закрывается, и статус меняется на "закрыт".
5. Закрытие бага: если дефект действительно устранен, баг закрывается, и статус меняется на "закрыт". Это завершает цикл управления дефектом и позволяет перейти к следующему этапу работы.

Таким образом Баг-репорты и багтрекинговая система играют ключевую роль в процессе тестирования веб-приложений. Они обеспечивают эффективное управление дефектами, ускоряя их устранение и повышая общее качество продукта. Правильная организация этого процесса способствует успешной разработке и поддержке приложений, удовлетворяющих потребности пользователей.

**Глава 3. Проведение тестирования и разработка предложений по улучшению тестирования веб приложения AnimalHouse.**

## **Важность тестирования веб-приложений**

Тестирование веб-приложений не только поможет избежать негативного опыта использования вашего продукта конечными пользователями. Такая работа повысит уровень удобства использования приложения, уровень безопасности важных данных, их сохранности и конфиденциальности, существенно снизит время отклика и повысит стабильность работы приложения при нестабильном интернет-соединении. Качественно проведенные проверки, и меры, принятые в соответствии с результатами тестирования, позволяют сделать так, чтобы ожидания пользователей, сформулированные в требованиях к веб-приложению, оказались максимально оправданы. Так или иначе, тестирование помогает сокращать траты на выпуск, доработку и дальнейшее обслуживание веб-приложения – проблемы, выявленные на этапе анализа требований, требуют гораздо меньшего вмешательства со стороны разработки и времени на реализацию правок. Это сэкономит заказчику деньги, разработчикам — время, а пользователям — нервы.

**3.1 Тестирование функциональности**

**Функциональное тестирование** обеспечивает уверенность в правильности работы всех функций ПО, необходимых приложению и прописанных в документации. Используется для доказательства того, что функции приложения имеют ожидаемый результат, который соответствует документации. Традиционно [тестирование функциональности](https://qaservicelab.ru/functional-testing/) веб-приложения подразумевает написание тестовых сценариев для всех функций приложения. Данные сценарии должны описывать шаги проведения проверок и критерии прохождения. Рекомендуется охватить наиболее широкий спектр вариантов сценариев и пользовательских данных.

Такие сценарии, помимо проверки основной функции, также включают в себя некоторый набор стандартных процедур. Например, тестирование гиперссылок, чтобы определить, все ли ссылки работают правильно, нет ли неработающих URL. Здесь важно не забыть про внутренние и внешние ссылки, якорные ссылки, ссылки для почтовых адресов и телефонов и т. д. Обязательно протестируйте формы и поля ввода, удостоверьтесь, что значения по умолчанию заполняются согласно документации, выводятся ли сообщения об ошибке в случае ввода неверных данных, или в случае отсутствия значений для обязательных полей.

**3.2 Тестирование юзабилити**

**Юзабилити-тестирование**, или [тестирование удобства использования](https://qaservicelab.ru/ui-ux-tesintg/), дает возможность оценить простоту использования программного продукта, улучшить взаимодействие с пользователем. Если у команды была фокус-группа — прекрасно, вам обязательно нужно узнать, соответствует ли приложение их ожиданиям. Проведите подробное тестирование интерфейса веб-приложения: проверьте логичность интерфейса, компоновку элементов, навигацию, локализацию проекта, визуальную привлекательность. Если важны культурные особенности для разнообразных групп пользователей — необходимо включить в список и их.

У пользователя должна быть возможность выполнять свои задачи быстрее — без лишних действий и шагов, без указания лишних данных. Также важно, чтобы действия были пользователю привычны. Т.е. если у приложения есть аналоги, и пользователь переходит на ваше ПО с этих программ, то важно сохранить некую преемственность, чтобы пользователю был понятен и привычен интерфейс приложения, его логика.

**3.3 Тестирование интерфейса**

Тестирование пользовательского интерфейса (UI testing) является одним из типов тестирования ПО, направленным на проверку корректности работы и удобства использования пользовательского интерфейса приложения. Важность данного вида тестирования обусловлена тем, что качество пользовательского интерфейса напрямую влияет на удовлетворенность пользователей и их готовность продолжать использовать приложение.

## Основные аспекты тестирования пользовательского интерфейса

Тестирование пользовательского интерфейса включает в себя проверку следующих аспектов:

1. **Внешний вид интерфейса**: проверка соответствия дизайна макетам, правильность отображения на разных устройствах и разрешениях, соблюдение стилей и цветовой гаммы.
2. **Элементы управления**: проверка корректной работы кнопок, полей ввода, выпадающих списков и других элементов управления, а также их доступности для пользователя.
3. **Навигация**: проверка простоты и понятности перемещения между разделами приложения, наличие хлебных крошек, работоспособность ссылок и меню.
4. **Отзывчивость**: проверка скорости загрузки страниц, отсутствия зависаний и ошибок при взаимодействии с интерфейсом.
5. **Сообщения об ошибках и подсказки**: проверка наличия и корректности отображения сообщений об ошибках, а также наличия подсказок и инструкций для пользователя.

### Разработка приложения, программного обеспечения или веб-сайта состоит из разработки компонентов, таких как серверы, базы данных и т.д. Соединение, которое объединяет и облегчает связь между этими компонентами, называется интерфейсом. Говоря простыми словами, это ПО, состоящее из набора команд и сообщений.

### Связь между различными компонентами программного обеспечения, приложения или веб-сайта может влиять на общую производительность, следовательно, интерфейс нуждается в проверке не меньше остальных составляющих.

**Ниже приведены три этапа тестирования в жизненном цикле интерфейса:**

1) Конфигурация и разработка:

Когда интерфейс настроен и как только начинается разработка, конфигурации должны быть проверены в соответствии с требованиями.

2) Валидация:

Интерфейс должен быть протестирован после завершения разработки, это может быть частью модульного тестирования.

3) Техническое обслуживание:

После того, как все программное обеспечение будет готово, отслеживается производительность интерфейса, его ошибки и соответственно вносятся исправления.

**Типы тестирования интерфейса:**

Тестирование интерфейса в основном выполняется на уровне обмена сообщениями системной архитектуры. В основном это связано с тестированием REST API, веб-службы SOAP с JSON или XML-форматом.

**Тестирование интерфейса обычно включает в себя следующие виды:**

* Модульное тестирование: тестирование функциональности каждой отдельной операции.
* Функциональное тестирование: тестирование функциональности более широких сценариев, связанных с созданием тестового примера, валидацией, регрессией и т. д.
* Нагрузочное тестирование: проверка работоспособности под нагрузкой, в основном с использованием функциональных тестовых примеров.
* Тестирование безопасности: тестирование механизма безопасности включает в себя тестирование на проникновение, а также проверку контроля доступа, шифрование и т. д.
* Тестирование рабочего процесса: это делается для того, чтобы обработка рабочего процесса движком интерфейса соответствовала ожиданиям.

Как и любое другое тестирование, тестирование интерфейса играет важную роль, так как оно обеспечивает правильную бесперебойную работу в будущем и высокую производительность различных приложений и систем. Тестирование интерфейса также важно и при проверке взаимодействия нашего приложения с другими приложениями.

**Лучшие инструменты для тестирования интерфейса**

Ручное тестирование интерфейса - очень утомительный, сложный и трудоемкий процесс.

Лучшим методом для тестирования интерфейса является использование автоматизации. Отсюда следует список инструментов, которые помогут вам как можно быстрее провести данный тип тестирования, и он включает в себя следующее.

**3.4 Тестирование совместимости**

**Тестирование совместимости** проверит совместимость приложения с различными устройствами, браузерами, операционными системами. При тестировании веб-сайта необходимо убедиться, что он отображается и выполняет свои функции одинаково во всех браузерах. Важно предоставить одинаковый опыт для пользователей, независимо от того, какой тип устройства они используют -десктоп, планшет или мобильное устройство. Не стоит забывать, что сайт – это не только клиентская часть в браузере, но и бекэнд. Это не такие популярные проверки, но все же многие веб-приложения должны поддерживать работу бека на разных операционных системах, должны быть совместимы с разными версиями баз данных, да и попросту с другим ПО, работающим на том же сервере, где и бекэнд веб-приложения.

**3.5 Тестирование производительности**

**Тестирование производительности** выявляет потенциальные узкие места, влияющие на работу при интенсивном трафике или большой нагрузке. Ваши тесты должны определить допустимые рабочие нагрузки в определенных условиях. Например, оптимальное и максимальное количество одновременно работающих пользователей. Или какие элементы системы в зоне риска при граничных нагрузках на веб-приложение. Производительность – это также и масштабируемость, стабильность, надёжность работы ПО. Производительность веб-приложения напрямую связана с достижением времени отклика и пропускной способностью, поэтому нелишне будет выполнение основных пользовательских сценариев в условиях медленного интернета или низко производительные устройства. Есть целый ряд видов [тестирования производительности](https://qaservicelab.ru/stress-testing/), которые могут быть включены в этот список: нагрузочное, стрессовое, стабильности, объемное тестирование и тестирование масштабируемости.

**3.6 Тестирование безопасности**

**Тестирование безопасности** поможет максимально защитить ваше приложение, уберечь от кражи пользовательские данные, поддержать целостность и конфиденциальность данных приложения. Процесс поможет определить, как система защищает данные, поддерживает ли она необходимую функциональность при использовании усиленной защиты, или же, наоборот, любые настройки в плане безопасности не защищают приложение от возможного прорыва цифровой обороны. Такое тестирование поможет выявить слабые места в безопасности системы, вероятные лазейки, дать возможные рекомендации об использовании дополнительных модульных инструментов защиты. Уделять внимание [тестированию безопасности](https://qaservicelab.ru/security-testing/) рекомендуют на всех этапах разработки приложения, начиная с начального, чтобы избежать непредвиденных проблем, особенно на поздних этапах разработки, когда исправление может потребовать значительны усилий.

**3.7 Подведение итогов. Разработка предложений по улучшению ручного тестирования веб-приложения Animal House.**

Процесс ручного тестирования веб-приложения AnimaHouse требует тщательной подготовки и координации усилий всей команды. Несмотря на сложности, связанные с отсутствием технического задания, существует ряд подходов и методов, которые помогут сделать тестирование более эффективным и качественным.

Основные выводы

1. Организация процесса:
   * Четкость и структура: необходимо разработать чёткий план тестирования, который будет включать в себя все этапы и процедуры. Это поможет избежать хаоса и обеспечит последовательное выполнение задач.
   * Коммуникация: Постоянное общение между тестировщиками, разработчиками и менеджерами проекта крайне важно для оперативного выявления и устранения проблем.
2. Использование прототипов и макетов:
   * Прототипы и макеты интерфейса помогают тестировщикам лучше понимать логику работы приложения и выявлять потенциальные проблемы ещё до начала разработки.
3. Создание собственных сценариев тестирования:
   * В условиях отсутствия ТЗ тестировщики должны самостоятельно разрабатывать сценарии тестирования, основываясь на своём опыте и знаниях о продукте. Эти сценарии затем обсуждаются с разработчиками для согласования.
4. Регулярные встречи и обсуждения:
   * Проводимые регулярно встречи способствуют лучшему пониманию текущих задач и планов на будущее, а также предотвращают возникновение недопониманий.
5. Документирование результатов:
   * Тщательная документация всех шагов и результатов тестирования помогает избежать повторных ошибок и повышает прозрачность процесса.

Предложения по улучшению

1. Минимальный набор требований:
2. Даже при отсутствии полного ТЗ стоит разработать минимальный набор требований, который бы определял основные функции и характеристики приложения. Это даст тестировщикам отправную точку для планирования и выполнения тестов.
3. Подробные прототипы и макеты:
4. Создание подробных прототипов и макетов интерфейса позволит тестировщикам глубже погрузиться в проект и заранее выявить возможные проблемы.
5. Интерактивные сессии с разработчиками:
6. Регулярные интерактивные сессии, на которых тестировщики и разработчики совместно обсуждают функциональность и сценарии использования, помогут улучшить взаимопонимание и сократить количество ошибок.
7. Автоматизация некоторых тестов:
8. Хотя основное внимание уделяется ручному тестированию, автоматизация некоторых повторяющихся или сложных тестов может сэкономить время и повысить точность проверок.
9. Постоянное обучение и развитие:
10. Тестировщики должны постоянно повышать свою квалификацию, изучая новые методики и инструменты тестирования, чтобы оставаться на переднем крае технологий и обеспечивать высокое качество работы.

Заключение

### **Заключение**

Ручное тестирование веб-приложения AnimaHouse в условиях отсутствия технического задания возможно и вполне реально.

Главное — правильная организация процесса, чёткое планирование, активная коммуникация и постоянное совершенствование навыков. Следуя предложенным рекомендациям, команда сможет значительно улучшить качество тестирования и выпустить продукт, соответствующий высоким стандартам.

Качественное тестирование веб-приложений очень важно и при должной организации позволяет не только повысить качество приложения, но и существенно сократить направленные на это расходы и усилия.

При тестировании веб-сайта или веб-приложения очень важно сделать так, чтобы он и выглядел отлично, и работал без сбоев. Вы не можете выпустить продукт, пока не будете уверены, что он понравится вашим пользователям. Именно здесь и приходит на помощь веб-тестирование. Каждый вид тестирования призван охватить определенную область, выявить крупные и мелкие проблемы, чтобы вы могли предоставить отточенный до мелочей готовый продукт.

Чтобы правильно организовать процесс тестирования, охватывающий все аспекты ПО, тщательно изучите шаги, советы и инструменты, описанные в этом руководстве. И помните, что экономия средств на грамотном тестировании в результате выльется в более серьезные финансовые потери, если после неудачного запуска придется переделывать все с нуля.

Разработку IT-решения – электронного прибора, приложения, встроенного программного обеспечения или IoT-системы – предваряет написание технического задания. Это может быть краткая спецификация или большое серьезное ТЗ – все зависит от масштабности и сложности проекта. ТЗ дает представление о назначении и функциях продукта, требованиях к разработке, ходе работ и порядке приемки готового решения.

Техническая спецификация – результат коллективного труда менеджера проекта, разработчиков, тестировщиков и, конечно, заказчика. Написание ТЗ – сложный процесс, который требует от разработчиков иметь знания и экспертизу в разработке программных и аппаратных решений, разбираться в рынке электронных компонентов, оценивать логистические маршруты, а также понятно излагать информацию. Лучше, если ТЗ напишет компания-разработчик, учтя все требования заказчика и свою экспертизу. Тогда разработка продукта будет идти быстрее и комфортнее и для исполнителя, и для заказчика.

В заключение следует отметить, что ручное тестирование является преимуществом для любой организации. Тестировщики могут обнаружить невидимые ранее проблемы и предоставить подробные отзывы о приложении, чего не может сделать автоматизация.

Ручное тестирование – это основа лучшей разработки программного обеспечения, и правильное его использование может существенно повлиять на результат.

Тестирование веб-приложений не только поможет избежать негативного опыта использования вашего продукта конечными пользователями. Такая работа повысит уровень удобства использования приложения, уровень безопасности важных данных, их сохранности и конфиденциальности, существенно снизит время отклика и повысит стабильность работы приложения при нестабильном интернет-соединении. Качественно проведенные проверки, и меры, принятые в соответствии с результатами тестирования, позволяют сделать так, чтобы ожидания пользователей, сформулированные в требованиях к веб-приложению, оказались максимально оправданы. Так или иначе, тестирование помогает сокращать траты на выпуск, доработку и дальнейшее обслуживание веб-приложения – проблемы, выявленные на этапе анализа требований, требуют гораздо меньшего вмешательства со стороны разработки и времени на реализацию правок. Это сэкономит заказчику деньги, разработчикам — время, а пользователям — нервы.

**Список используемой литературы**

*Литературные издания:*

[И. Винниченко «Автоматизация процессов тестирования»](https://software-testing.ru/books/testing-books/216-vinnichenko)

[Кент Бек «Экстремальное программирование: разработка через тестирование»](https://software-testing.ru/books/testing-books/225-beck)

[Рекс Блэк «Ключевые процессы тестирования»](https://software-testing.ru/books/testing-books/215-black)

[Роберт Калбертсон, Крис Браун, Гэри Кобб «Быстрое тестирование»](https://software-testing.ru/books/testing-books/222-rapid-testing)

[Роман Савин «Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах»](https://software-testing.ru/books/testing-books/213-savin)

[Юань Хсих, Мэтт Тэллес «Наука отладки»](https://software-testing.ru/books/testing-books/1395-2011-07-07-15-59-39)

*Научные статьи:*

* Пример  жизненного цикла дефекта в большой команде: <http://barancev.blogspot.com/2007/11/blog-post.html>
* Описание дефекта: Severity и Priority <http://alexlobach.ru/2009/07/15/opisanie-defekta-severity-i-priority/>
* Артефакты, необходимые для тестирования: <http://alexlobach.ru/2008/09/03/artefakty-neobxodimye-dlya-testirovaniya/#more-20>
* Что необходимо для внедрения автоматизации тестирования ПО <http://www.software-testing.ru/library/testing/general-testing/549-test-automation>
* Что такое качество программного обеспечения? <http://www.software-testing.ru/library/testing/general-testing/124-2008-09-30-07-39-35>
* Место тестирования среди методов оценки качества ПО  <http://www.software-testing.ru/library/testing/general-testing/117-2008-10-13-19-25-13>
* Стратегия в тестировании <http://www.software-testing.ru/library/testing/general-testing/207-2008-10-06-10-05-53>
* Случай потерянного «IF» <http://www.software-testing.ru/library/testing/general-testing/110--lifr->
* Модульное тестирование. Зачем, как и кто  <http://www.software-testing.ru/library/testing/general-testing/77-2008-09-29-07-30-13>
* TOP 13 ошибок тестировщиков. <http://www.software-testing.ru/library/testing/general-testing/66-top-13----i----> и <http://www.software-testing.ru/library/testing/bug-tracking/65-top-13-ii->
* Эссе о критериях <http://www.software-testing.ru/library/testing/general-testing/4-2008-09-28-17-12-35>
* Мифы об автоматизированном тестировании <http://www.software-testing.ru/library/around-testing/processes/462-automation-myths>
* Определение дефекта <http://blog.shumoos.com/archives/106>
* О некоторых видах проблем, возникающих при пропуске требований: <http://blog.shumoos.com/archives/165>
* Место ОТК в разработке ПО <http://blog.shumoos.com/archives/168>
* Разбор задачи по составлению тестового набора (преобразование даты в число) <http://blog.shumoos.com/archives/107>
* Приемы тестирования полей ввода: <http://blog.shumoos.com/archives/67>
* Пример использования метода Test First при создании юнит тестов <http://blog.shumoos.com/archives/73>
* История о длинной тестовой последовательности: <http://blog.shumoos.com/archives/75>
* Объемное тестирование: <http://blog.shumoos.com/archives/77>
* Нагрузочное тестирование: <http://blog.shumoos.com/archives/79>
* Оценка целесообразности автоматизированного тестирования: <http://blog.shumoos.com/archives/92> и <http://blog.shumoos.com/archives/138>
* Введение в нагрузочное тестирование <http://blog.shumoos.com/archives/98>

*Интернет-ресурсы:*

<https://www.mathnet.ru/links/60756948fce9ba1f8e3d19ec8eee2de7/ssi179.pdf>

<https://ru.hexlet.io/courses/web-testing-basics/lessons/intro/theory_unit>

<https://habr.com/ru/articles/715376/>

<https://skillbox.ru/media/code/kto_takoy_testirovshchik_i_kak_im_stat_s_nulya/>

<https://qarocks.ru/guide-to-web-testing/>

<https://habr.com/ru/articles/772732/>

<https://software-testing.ru/library/around-testing/requirements/3567-requirements>

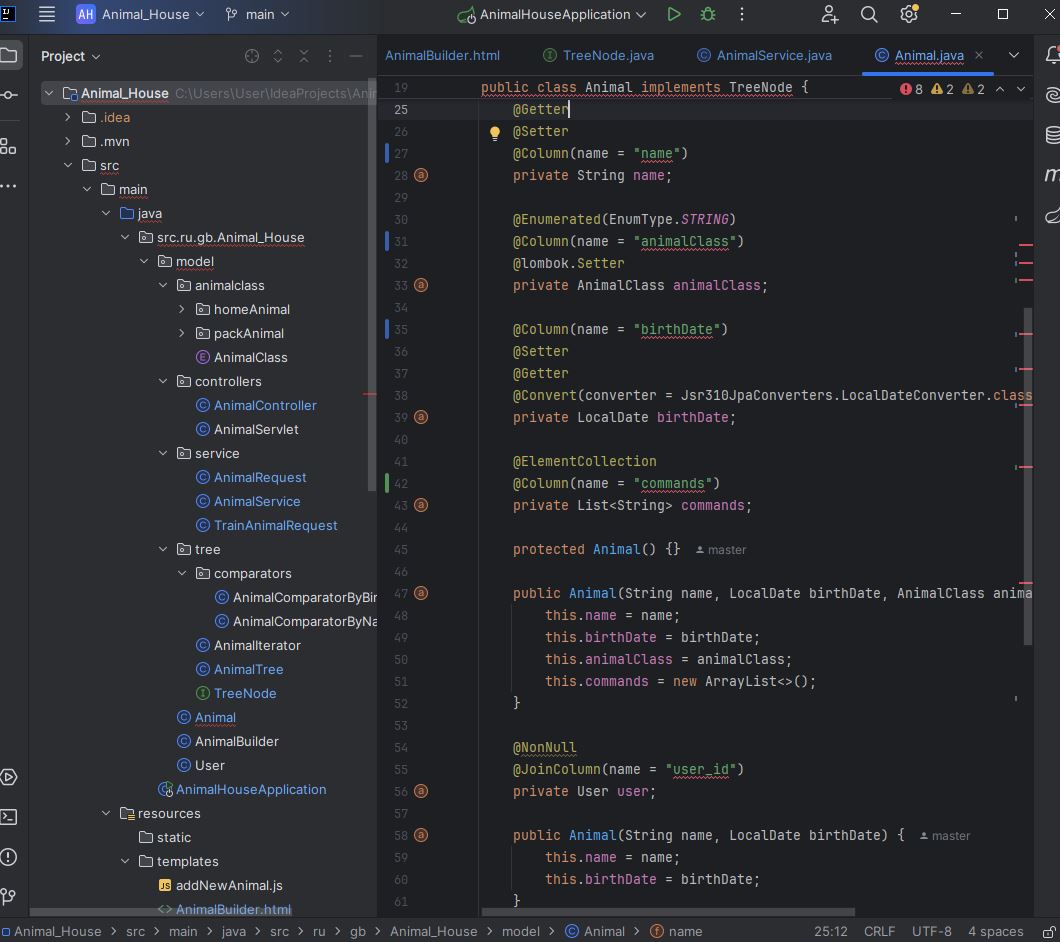
<https://sky.pro/media/ruchnoe-testirovanie-chto-eto-i-kak-nachat/>

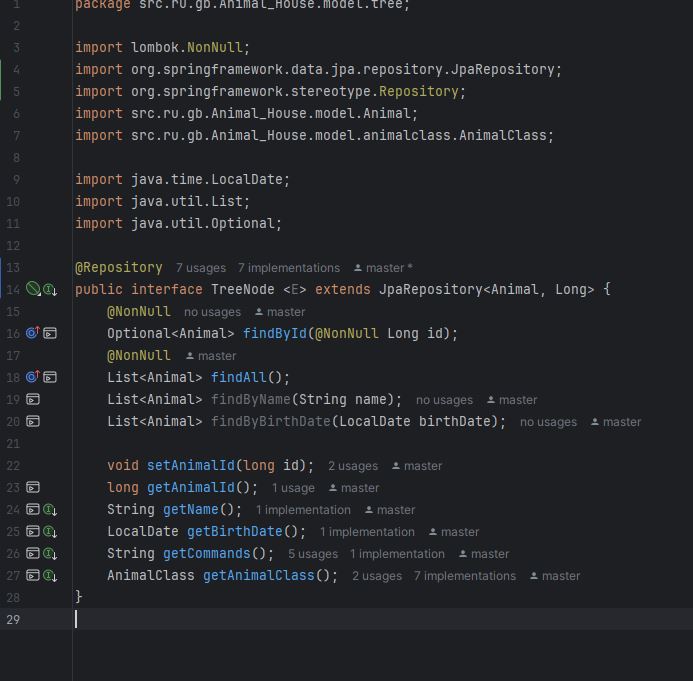
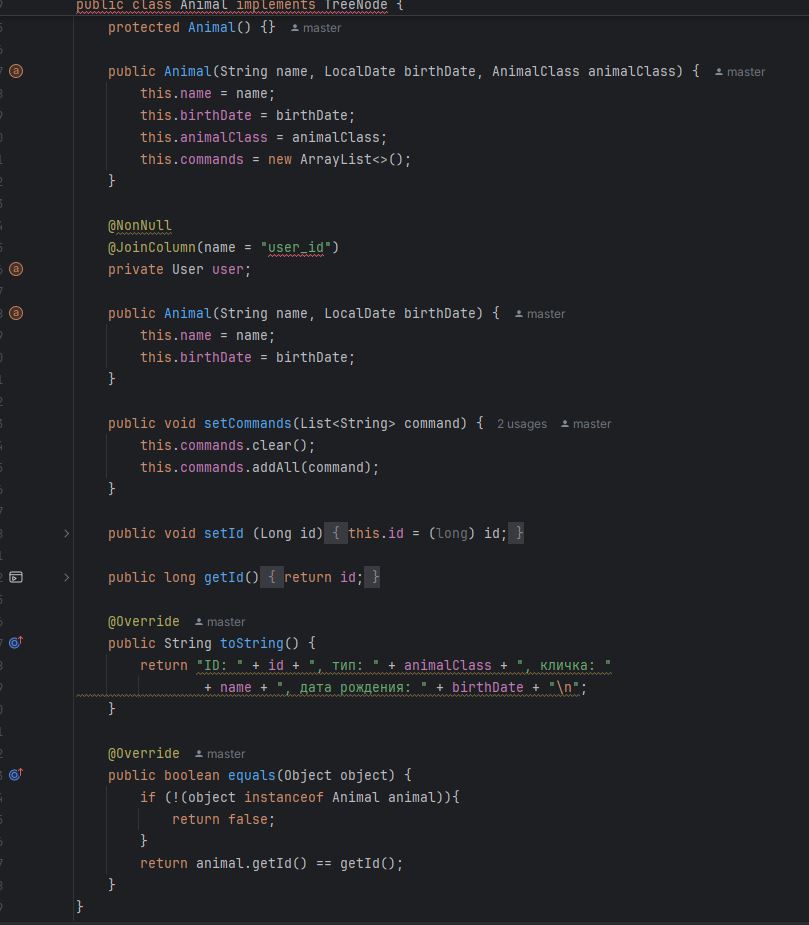
<https://qaservicelab.ru/blog/testirovanie-web-prilozhenii/>

<https://sky.pro/media/chto-takoe-testirovanie-polzovatelskogo-interfejsa/>

***Приложения.***

*Структура проекта AnimalHouse*

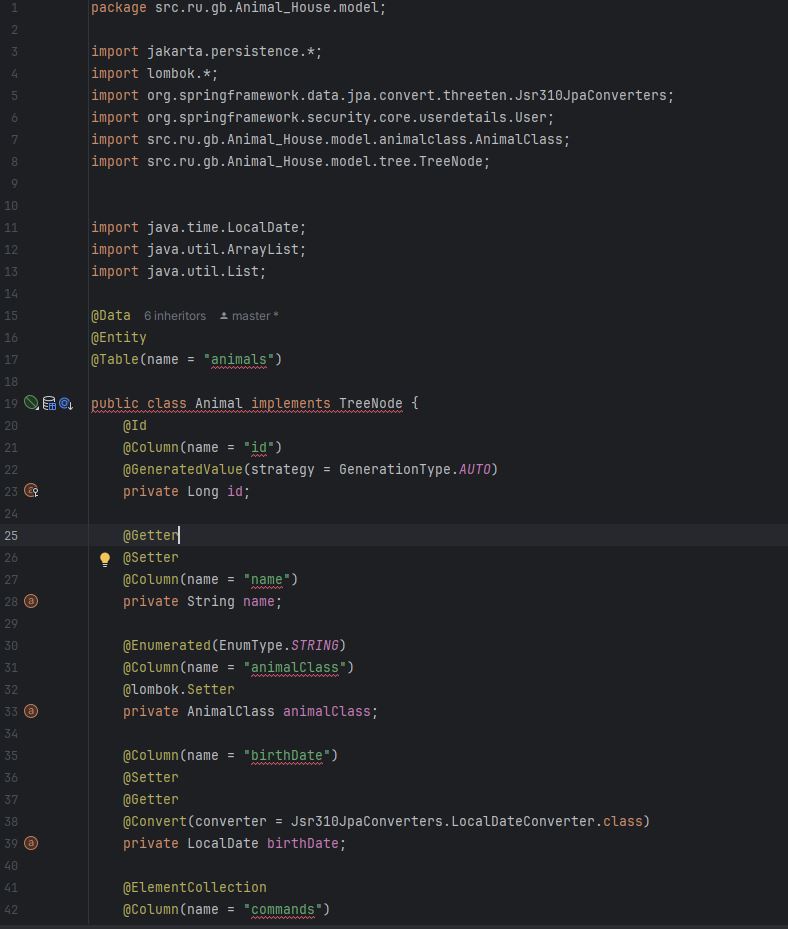
**

Animal-объект класса

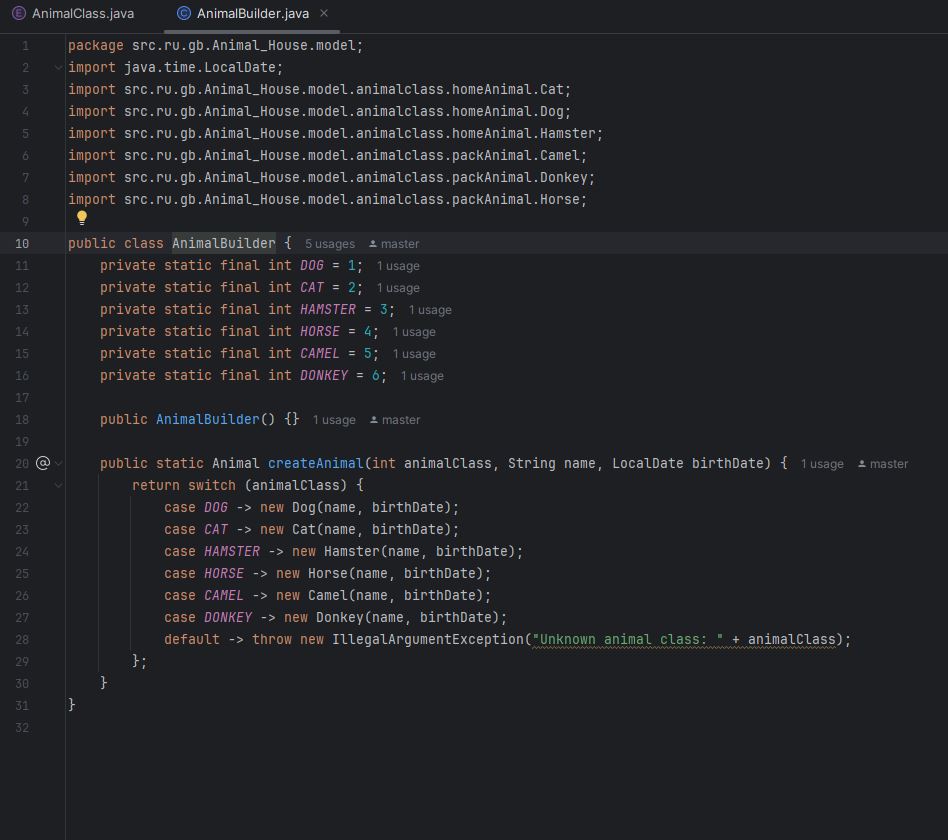
Интерфейс TreeNode наследуемый от JPARepository

Наследник AnimalTree интерфейса TreeNode для реализации методов программы\_1

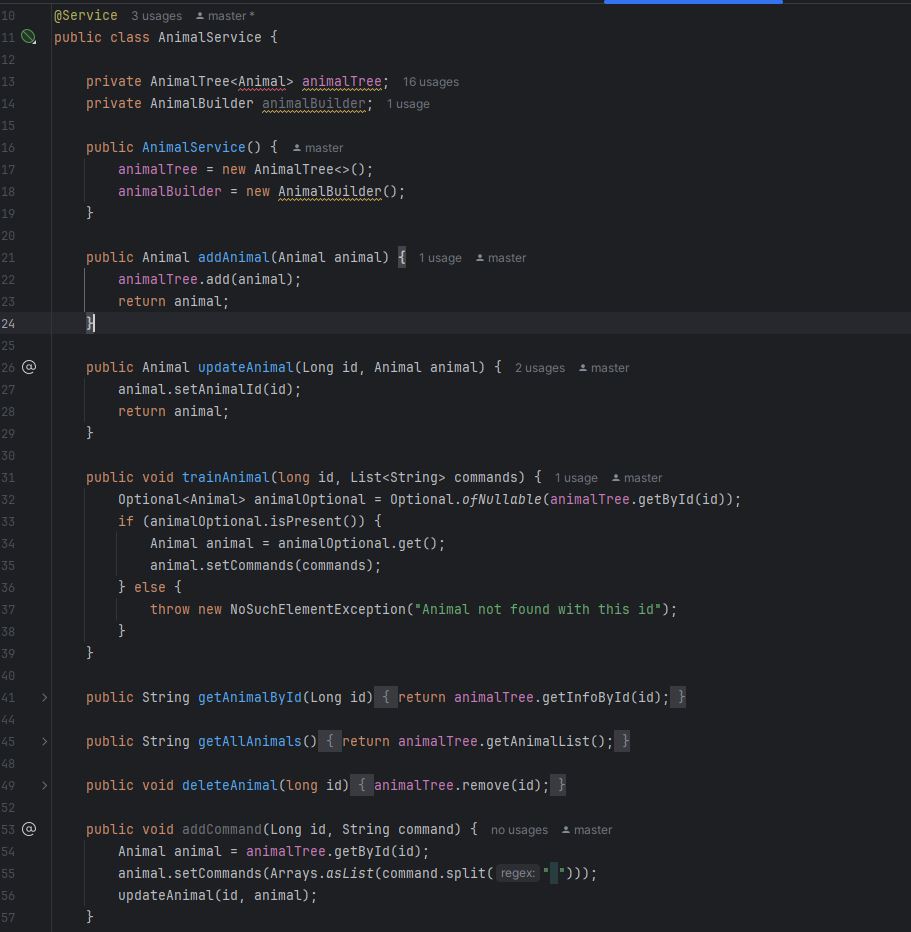


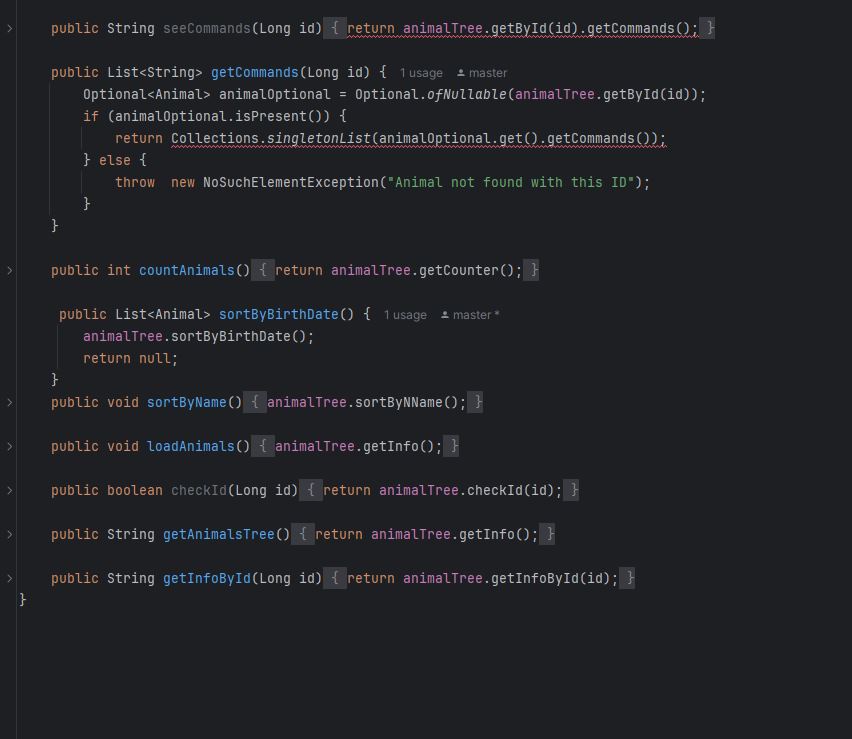
*Наследник AnimalTree интерфейса TreeNode для реализации методов  *

*Данный класс AnimalBuilder создает подклассы животных*

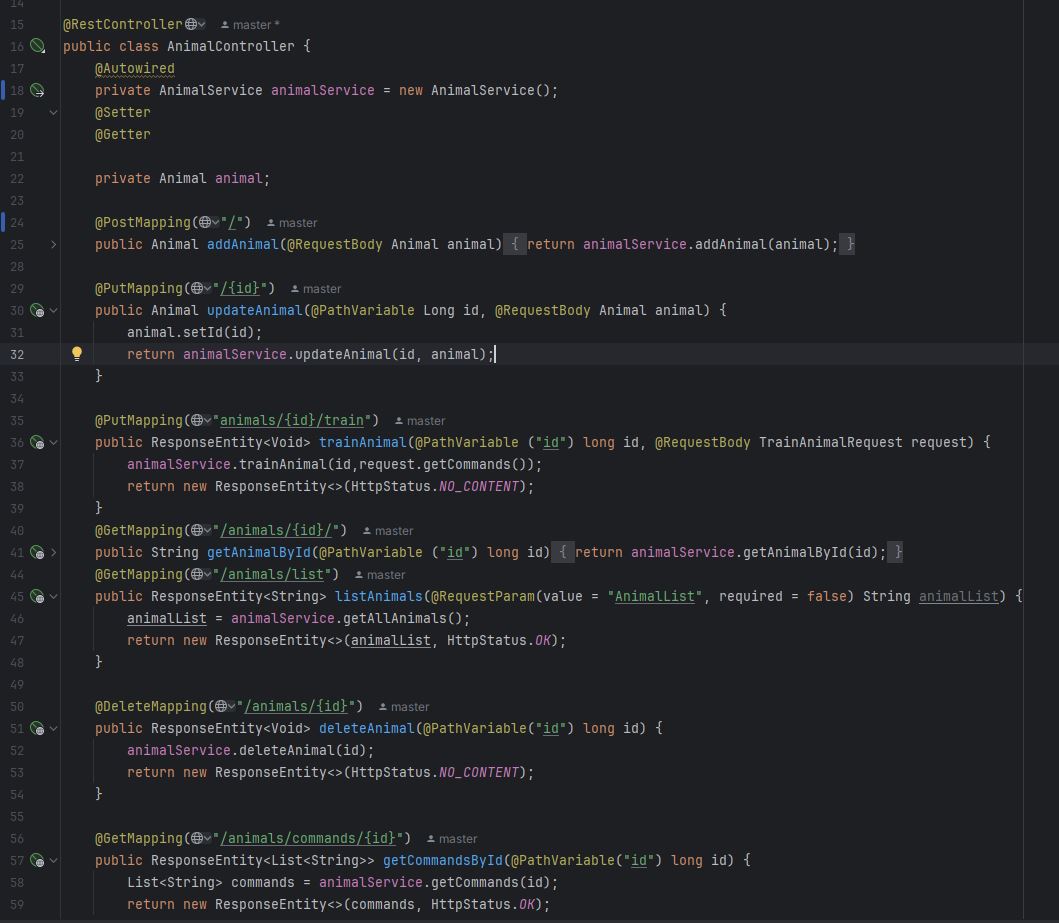
**

Главный сервис программы с аннотацией Service от SpringBoot\_1

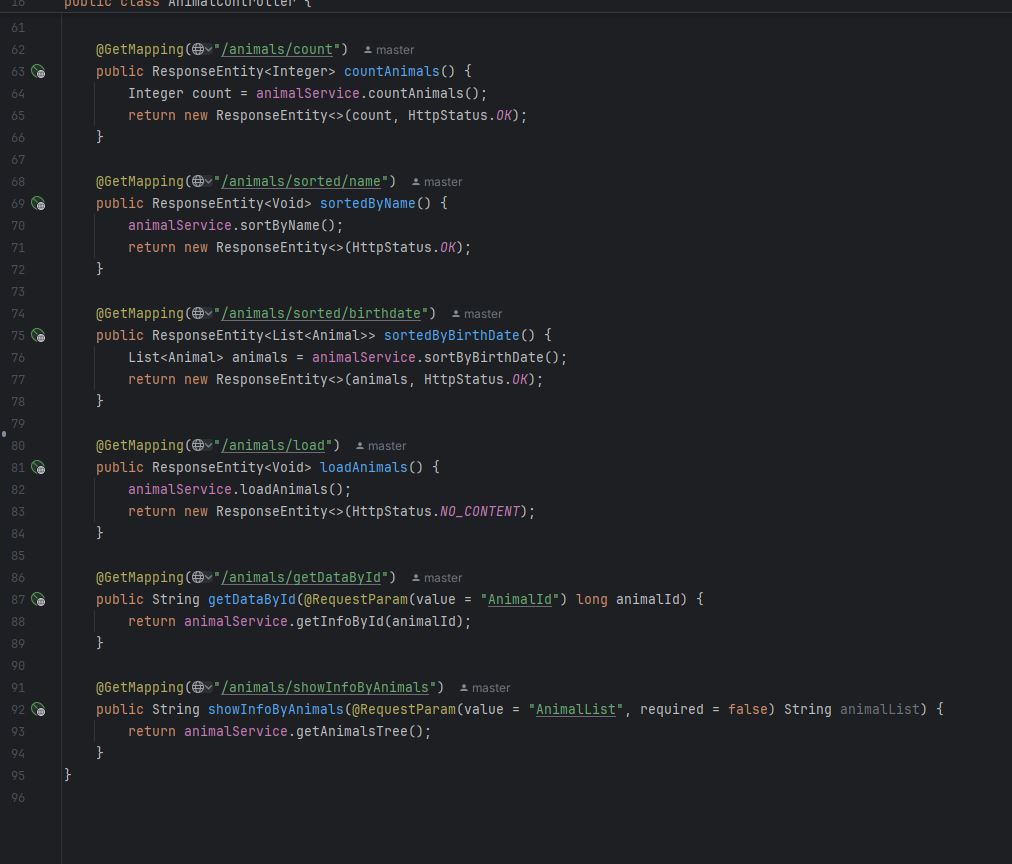


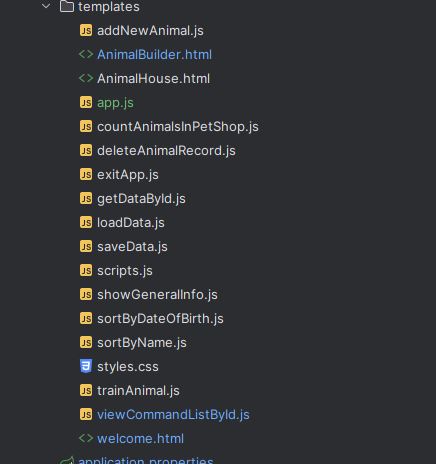
Главный сервис программы с аннотацией Service от SpringBoot\_2

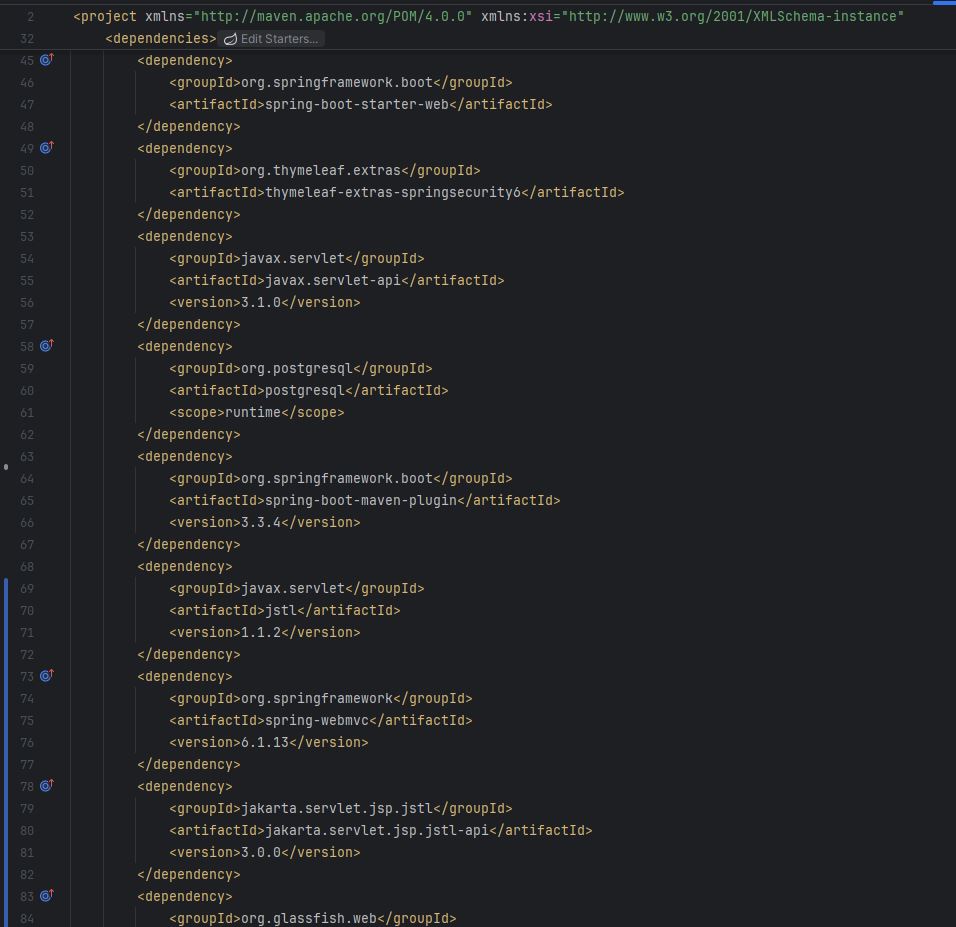
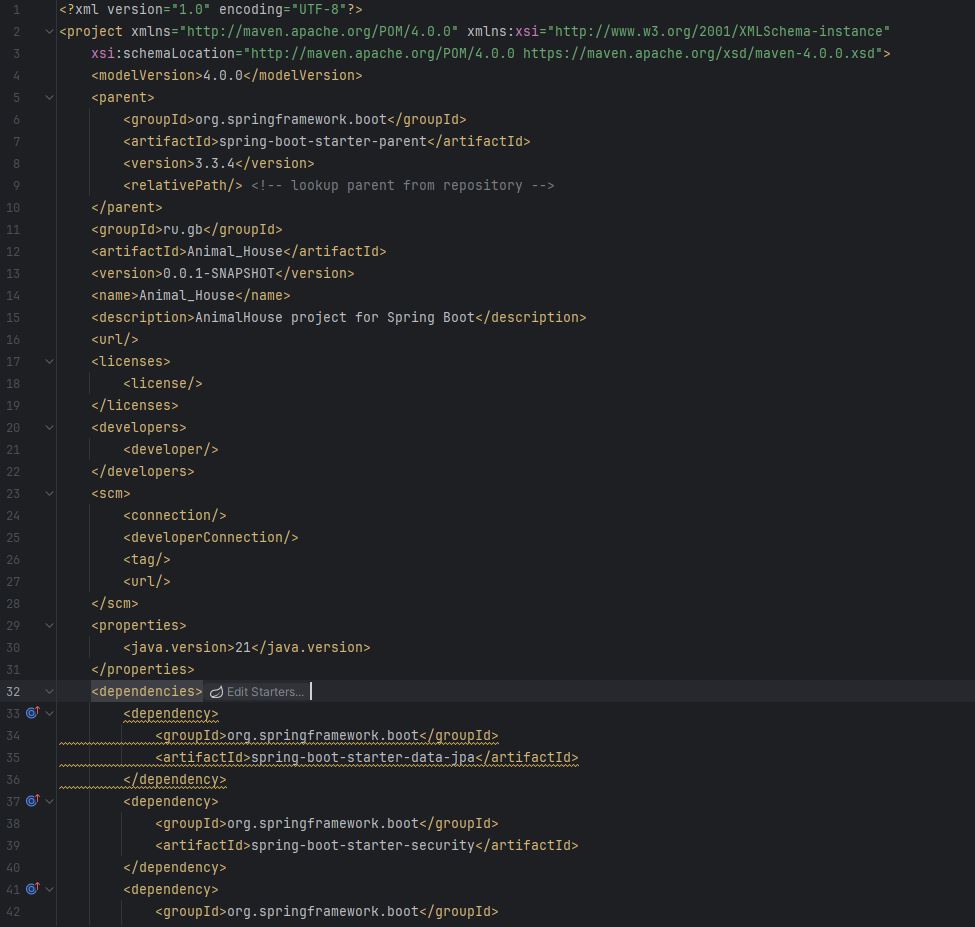
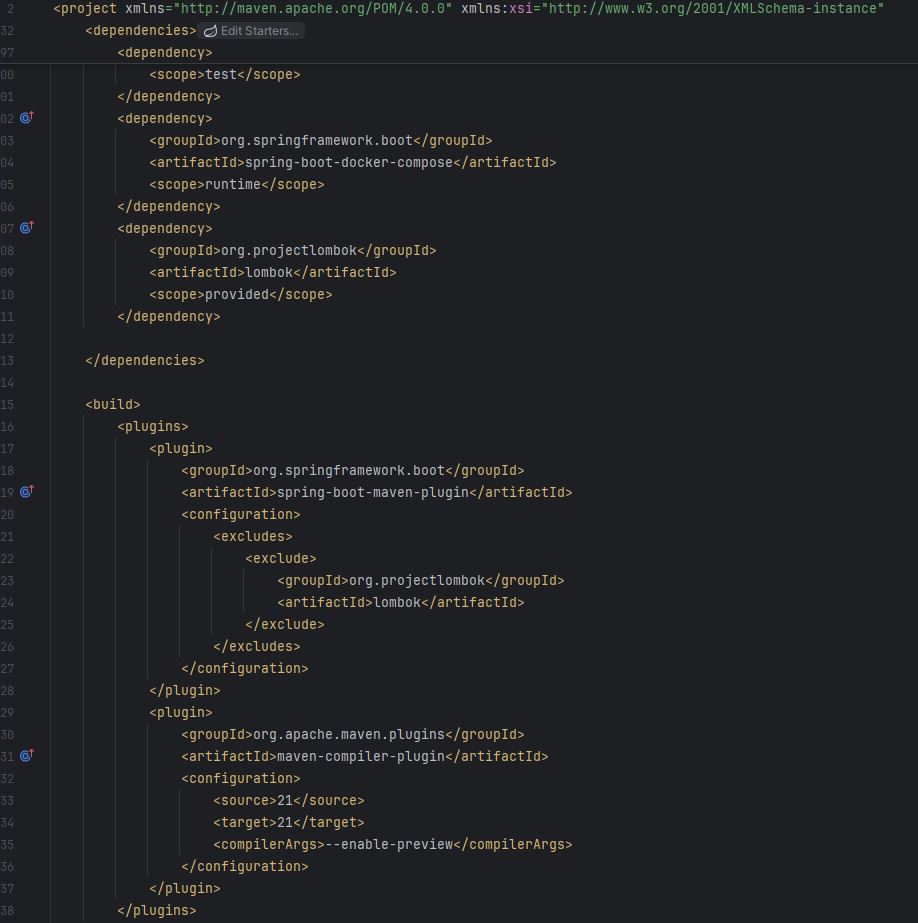
AnimalController для связывания сервиса и основой логики программы\_1

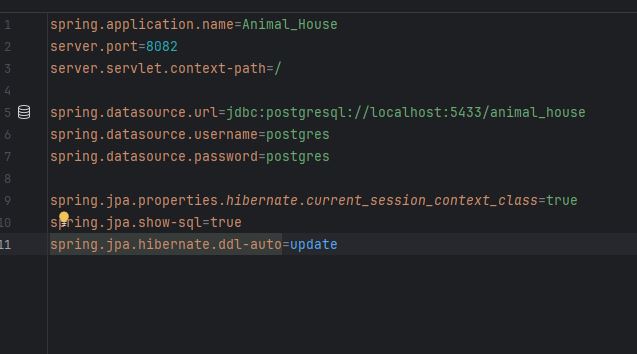


AnimalController для связывания сервиса и основой логики программы\_2

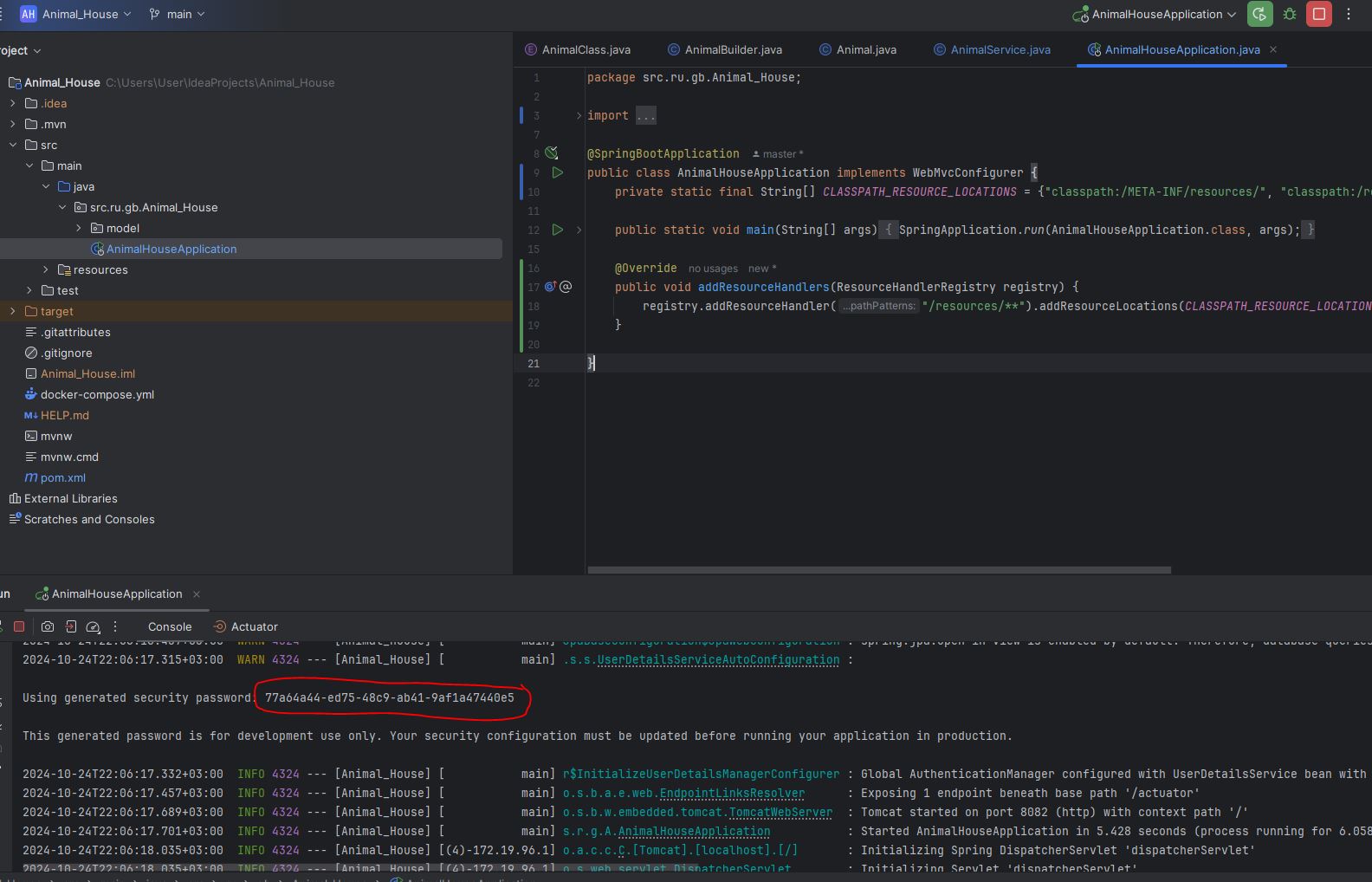


Templates используются программой для создания интерфейса в браузере, скрипты выполняют логику программы и связаны с базой данных

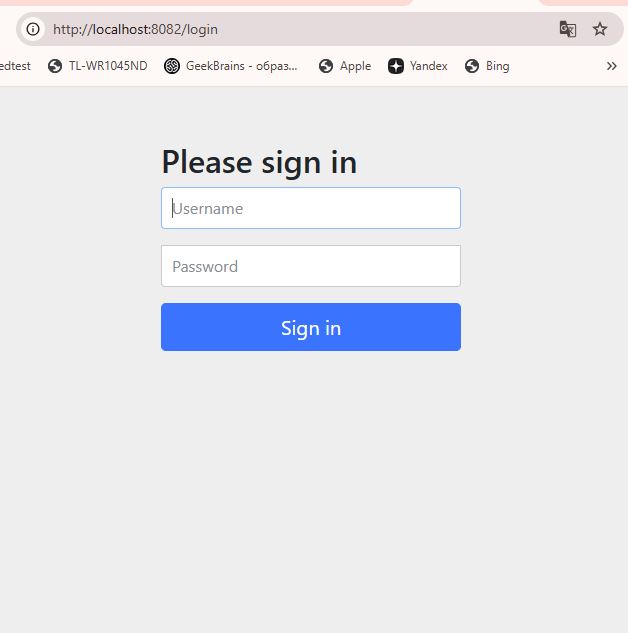
Файл\_pom.xml-зависимости(dependencies)\_1

С помощью файла application.properties заданы настройки для интегрирования со средой SpringBoot

Запуск приложения на SpringBoot с аутентификацией пользователя user (генерация пароля автоматически)



Подключение к системе AnimalHouse



Запуск основного меню приложения AnimalHouse