Тестирование ПО. Уровень 1

Работа с баг-трекером

Определение и функции баг-трекера. Оформление ошибок в баг-трекере. Дополнительные инструменты в помощь тестировщику.

Оглавление

[Баг-трекинговая система](#_51m7clj0qk9f)

[Что такое баг-трекинговая система?](#_1isn9f16zlp6)

[Сущности или типы задач, которые бывают в баг-трекинговой системе](#_vfm3k4ffz72v)

[Разработка функциональности на примере Epic](#_c0b5q7blprp5)

[API-тестирование](#_vvalc4pksxzu)

[Как происходит обмен запросами и ответами с использованием API](#_n8cuz87mlrfi)

[Панель разработчика — DevTools](#_3dy6vkm)

[Оформление ошибки в баг-трекере](#_3znysh7)

[Глоссарий](#_2et92p0)

[Практическое задание](#_z8a7k43tz496)

[Требования к выполненной работе](#_12i29zkose7v)

[Дополнительные материалы](#_17dp8vu)

[Используемые источники](#_lnxbz9)

# Баг-трекинговая система

## Что такое баг-трекинговая система?

В ISTQB баг-трекинговые системы называются **инструментами управления инцидентами**.

**Инструмент управления инцидентами** — инструмент, который облегчает запись и отслеживание статуса инцидентов.

Эти системы используются для создания отчетов о дефектах, и их управлением.

Самая популярная баг-трекинговая система сейчас — JIRA, продукт компании Atlassian. Также встречаются такие баг-трекинговые системы, как YouTrack, Mantis, Redmine, Bugzilla.

Посмотрим, какие роли могут быть в баг-трекинговой системе у членов команды разработки и кто за что отвечает.

**Руководитель проекта:**

* Следит за выполнением задач.
* Следит за статусом задач.
* Меняет приоритет у задач по необходимости.
* После проверки выполнения задачи закрывает ее. Надо знать, что задача может быть закрыта не только потому что была исправлена. Она может быть закрыта, потому что изменились требования и это уже не дефект.

**Аналитик:**

* Формулирует задачи. На основании требований может создавать одиночные задачи разработчикам или целые эпики разработки большого набора функций. Эпик — задача в баг-трекере, которая может содержать в себе описание на разработку большой части приложения. Эпики будут содержать декомпозированные задачи на анализ/разработку/тестирование, или по-другому подзадачи/sub-task.
* Добавляет комментарии или пояснения в задачу в тех случаях, когда нужна консультация аналитика и когда она назначается на него.

**Разработчик:**

* Использует баг-трекер как список дел.
* Вносит информацию по тому, что было сделано, или что будет сделано по задаче. К примеру, это может быть дополнительное пояснение, как тестировать задачу, или на что стоит обратить особенное внимание.
* Прикрепляет ссылки к ветке от git, в которой происходило исправление дефекта. Не все баг-трекеры это поддерживают.
* Создаёт новую задачу сам.

**Тестировщик:**

* Создаёт новый отчет о дефекте, или баг-репорт. Иногда на проектах говорят — «создаёт» или заводит дефект. Могут выражаться на английский манер — «поднять» дефект.
* По описанию задачи проводит тестирование новой функциональности. То есть сверяет, как описана функциональность и как она на самом деле реализована.
* Проводит re-testing/confirmation testing (повторное тестирование) исправленных дефектов. То есть тестировщик повторно проходит шаги из описания, чтобы убедиться, что дефект исправлен и не воспроизводится.
* Создаёт другие задачи. Например, на разработку автотестов.

## Сущности или типы задач, которые бывают в баг-трекинговой системе

Рассмотрим на примере баг-трекинговой системы Redmine типы задач, которые могут использоваться на проектах.

**Epic** — тип задачи, которая посвящена разработке большой части функционала разрабатываемого продукта. Как правило, Epic (иногда выступает как родительская задача) разбивается на подзадачи, декомпозируется на меньшие задачи, которые можно назначить разным членам команды на проекте, чтобы ускорить процесс разработки новой функциональности.

К примеру, если начинают разработку новой функциональности «Кабинет пользователя», то в рамках этой задачи могут выделить следующие подзадачи (Sub-task):

* Добавление новых схем в существующую базу данных.
* Разработка дизайна (макета) кабинета пользователя.
* Разработка frontend- части кабинета пользователя.
* Разработка backend-части кабинета пользователя.

И так далее.

**Sub-task** — подзадача, которая может быть выделена отдельно, если родительская задача слишком сложная, и ее требуется разделить на более мелкие задачи. Подзадачи создаются «внутри» родительской задачи и связаны ею между собой.

**New feature** — тип задачи на разработку новой функциональности.

**Improvement** — тип задачи на улучшение или доработку существующей функциональности. К примеру, это может быть улучшение графического интерфейса или увеличение удобства использования приложения.

**Task** — тип задачи, который и переводится как «задача». Это может быть какая-то разовая задача, которую члены команды могут назначить сами на себя для списания рабочего времени.

Здесь надо уточнить, что работа с задачами в баг-трекинговой системе часто связана с учётом рабочего времени, потраченного на выполнение той или иной задачи. В рамках проекта это очень важно, чтобы понимать, на какую задачу ушло больше времени, чем ожидалось (то есть вышли за расчетные временные рамки), а где задача была выполнена раньше установленных сроков.

К примеру, это может быть задача на написание набора тест-кейсов, посвященных тестированию кабинета пользователя, или автоматизация уже существующего набора тест-кейсов, который входит в набор для регрессионного тестирования.

**Support** — тип задачи, который может приходить из службы поддержки.

**Bug** — ип задачи, который используется собственно для создания отчета о найденном дефекте.

Придя на проект, уточните, какие сущности используются в баг-трекинговой системе. Приведенные выше примеры могут использоваться не на всех проектах.

## Разработка функциональности на примере Epic

Мы с вами рассмотрели, какие типы задач могут быть в баг-трекинговой системе. Мы видели, что в Redmine можно не только создавать отчёты о дефектах, но и такой тип задач, как Epic.

Рассмотрим пример разработки формы регистрации в упрощенном виде.

В первую очередь, такую большую задачу не будет решать один специалист. Как правило, задача, которая требует вовлечения специалистов разной направленности, будет разбита на более мелкие подзадачи.

В нашем случае общая задача «Создание формы регистрации» может быть разбита на следующие подзадачи:

1. Разработка макета UI формы для регистрации нового пользователя.
2. Разработка frontend-части формы регистрации.
3. Разработка backend-части для формы регистрации.
4. Разработка схем в базе данных и добавление в неё таблиц, в которых будет храниться информация о новых зарегистрированных пользователях.

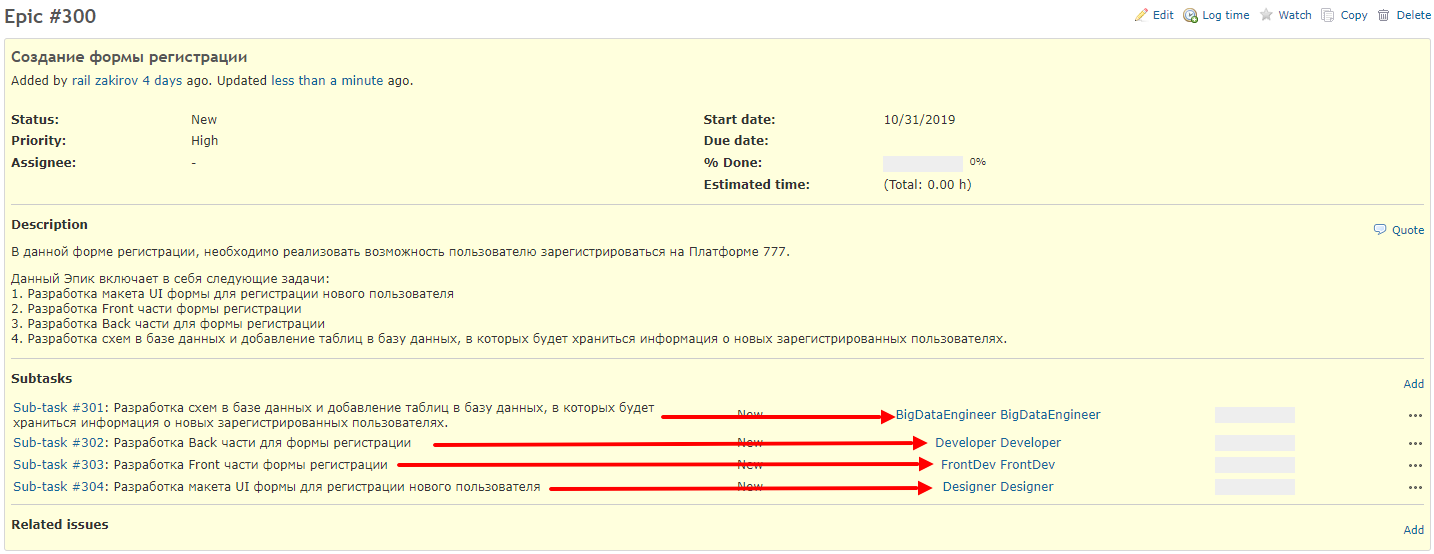
Соответственно, каждая из задач будет назначена профильному специалисту.

Разработка схем базы данных будет назначена специалисту по базам данных.

Разработка макета (дизайна) UI-формы для регистрации нового пользователя будет назначена дизайнеру.

Разработка frontend-части на основании макета будет назначена на frontend-разработчика.

Разработка/реализации бизнес логики и backend-части будет назначена на backend-разработчика.



Команда тестирования проведёт проверку каждой задачи по отдельности, а также тестирование полной реализации задачи после завершения Epic.

Ниже примерный список задач на тестирование (и не только), которые будут у специалиста во время разработки такой большой задачи:

1. Протестировать требования — функциональные, технические или детальные. Всю документацию, которая будет доступна.
2. Во время тестирования требований — составление чек-листа с дальнейшим составлением тест-кейсов (верхнеуровневых или низкоуровневых).
3. Тестирование макета/дизайна формы регистрации ещё до начала разработки frontend-части
4. Первичное тестирование/проверка реализации создания таблиц в базу данных, когда инженер по базам данных закончит свою задачу.
5. Тестирование backend-части.
6. Тестирование frontend-части.
7. Проведение тестирования после полной реализации Epic.

В ходе разработки каждая из задач, вероятно, может делаться параллельно другой. Но в данном случае это не так. Если посмотреть на задачи ещё раз, можно увидеть, что работа frontend-разработчика, вернее, её начало, зависит от того, когда дизайнер закончит свою часть. То есть, когда он закончит проектирование макета UI-формы регистрации. Без этого frontend-разработчик не сможет начать работу.

Остальные же задачи могут браться в работу одновременно и реализовываться параллельно.

Также часто бывает, что задачи по разработке backend-части могут быть закончены без реализации frontend-части. В таком случае, руководствуясь принципом раннего тестирования, нам стоит начать тестировать backend-часть без окончания реализации frontend-части.

Тестировать backend-части изолированно помогает API-тестирование.

# API-тестирование

Сейчас API-тестирование очень распространено и может требоваться даже для Junior-специалиста.

Что такое API?

Application Programming Interface (API) — тип интерфейса, который использует компоненты или системы, вовлечённые в обмен информацией. Формат информации имеет определенную формальную структуру.

Другими словами, API — описание правил и способов взаимодействия программ, которые были выработаны между разработчиками.

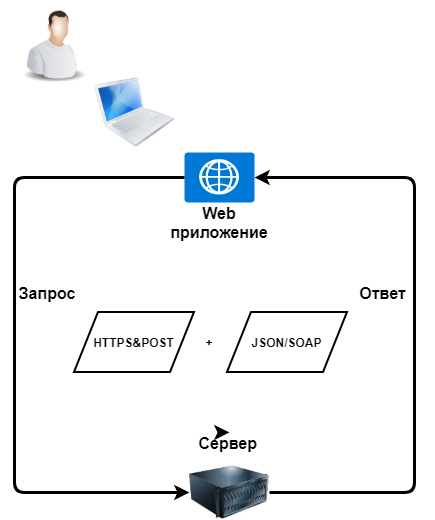
API тестирование — тестирование, выполняемое путём отправки команд в сторону приложения с использованием напрямую API приложения.

Существует несколько видов интерфейсов. Здесь же мы будем рассматривать только интерфейсы, посредством которых происходит взаимодействие с веб-приложением.

Некоторые популярные приложения дают в открытом доступе описание их API, чтобы разработчики сторонних сервисов могли создавать приложения, взаимодействующие с ним. К примеру, такой мессенджер, как Telegram.

## Как происходит обмен запросами и ответами с использованием API

Есть приложение, или клиент, который установлен на ПК пользователя. Клиентом может быть браузер. Когда вы вводите адрес сайта, который хотите посетить, клиент отправляет запрос на сервер (в определенном формате): «Дай мне информацию с этого сайта». Сервер в ответ присылает запрашиваемую информацию. Это наиболее упрощённый вариант взаимодействия клиента и сервера. Это такой высокоуровневый метод абстракции. Отправка запроса на получение информации происходит на API приложения, которое установлено на удалённом сервере.



В описание документации к API входит:

* описание отправляемого запроса,
* описание данных в запросе, то есть тех данных, которые примет приложение,
* описание данных в ответе.

**Описание отправляемого запроса может выглядеть следующим образом:**

1. Тип запроса, к примеру, POST.
2. End point или адрес, куда мы отправляем запрос — адрес ресурса (https://petstore.swagger.io) и сам ресурс (resource, к примеру v2/pet). Например, <https://petstore.swagger.io/v2/pet>. То есть Endpoint = Base URL + Resource.

**Описание данных в запросе может выглядеть следующим образом:**

1. Тип текстового формат документа, к примеру, JSON.
2. Содержание, к примеру:

|  |
| --- |
| {  "id": 25,  "category": {  "id": 0,  "name": "string"  },  "name": "test",  "photoUrls": [  "string"  ],  "tags": [  {  "id": 0,  "name": "string"  }  ],  "status": "available" } |

**Описание данных в ответ может выглядеть следующим образом:**

1. Статус — код 200 для успешного запроса.
2. Содержание:

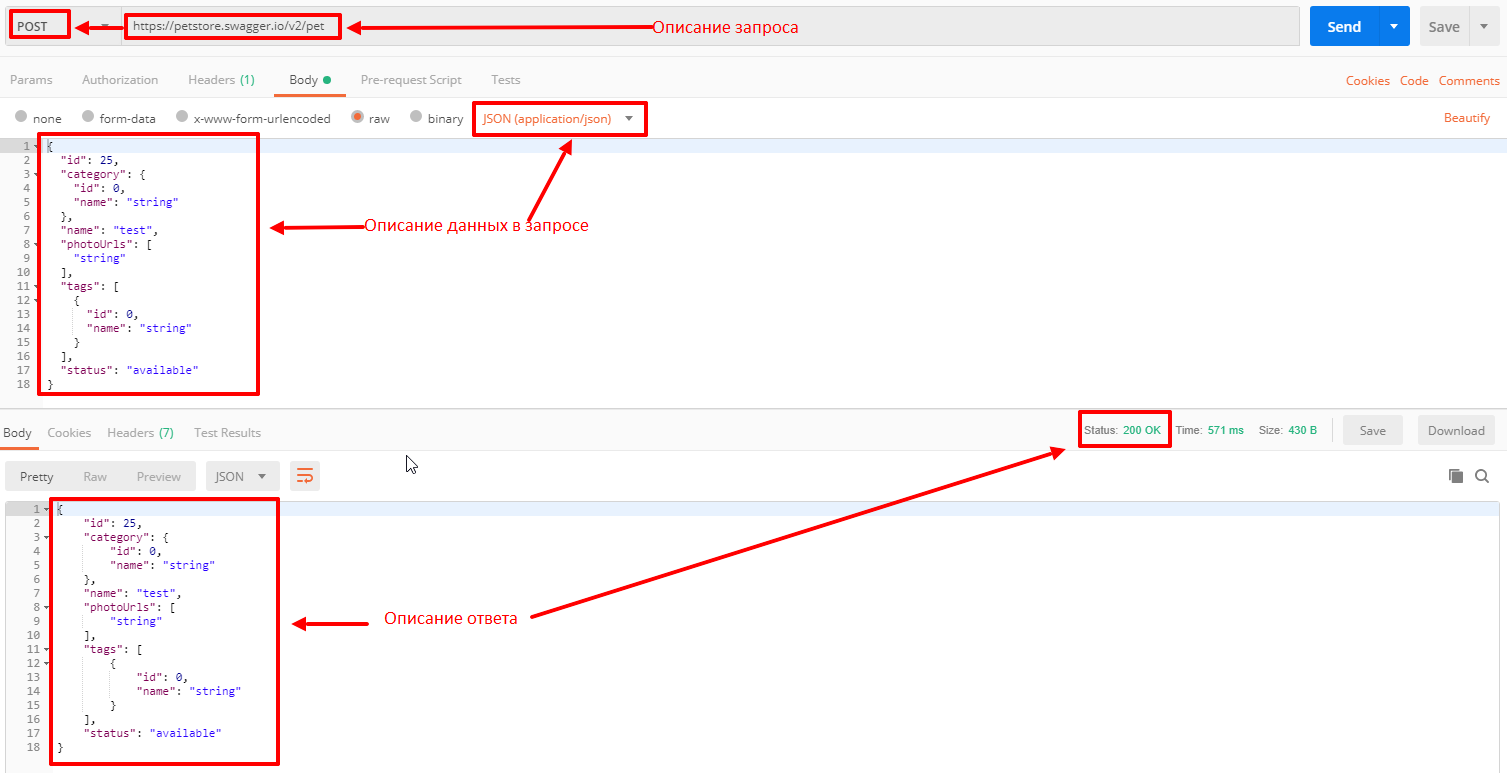
|  |
| --- |
| {  "id": 25,  "category": {  "id": 0,  "name": "string"  },  "name": "test",  "photoUrls": [  "string"  ],  "tags": [  {  "id": 0,  "name": "string"  }  ],  "status": "available" } |

Стоит помнить про группы статус-кодов ответов, ответов сервера на запрос со стороны клиента. Среди них:

* 1xx: Informational (информационные);
* 2xx: Success (успешно отравленные);
* 3xx: Redirection (перенаправление);
* 4xx: Client Error (ошибка клиента);
* 5xx: Server Error (ошибка сервера).

Одно из наиболее популярных приложений, используемое для тестирования API, — Postman.

Ниже представлен пример отправки запроса через Postman.

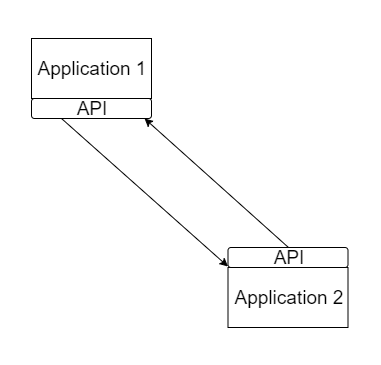


Взаимодействие с API может происходить:

* Внутри системы, когда несколько компонентов одной системы (платформы) взаимодействуют между собой для обмена данными и их получения.



* Взаимодействие между системой и внешними сервисами.



Когда вы проводите тестирование API и сталкиваетесь с ошибками, в отчёте о дефекте необходимо будет полностью указывать:

1. Какой запрос и куда вы отправляете.
2. Какие данные вы отправляете.
3. Какой ответ вам приходит.
4. Сверку с ожидаемым ответом.

# Панель разработчика — DevTools

При проведении тестирования веб-приложений один из основных инструментов, которым пользуются инженеры по тестированию — панель разработчика, или Dev Tools. Сейчас, наверное, в каждом современном Desktop-браузере есть подобный инструмент. У каждого браузера свои особенности, но в целом они одинаковы.

Используя панель разработчика, вы можете находить дополнительную информацию, необходимую для разработчика, чтобы понять, в чем именно заключается проблема. Поэтому навык работы с панелью разработчика крайне полезен, даже для начинающего тестировщика.

Мы с вами будем рассматривать панель разработчика на примере браузера Chrome. Панель разработчика встроена в браузер Google Chrome, помогает редактировать веб-страницы прямо в браузере и выявлять потенциальные проблемы очень быстро.

Нажмите Ctrl + Shift +I, чтобы открыть панель разработчика.

В открывшейся директории вам будут доступны следующие панели:

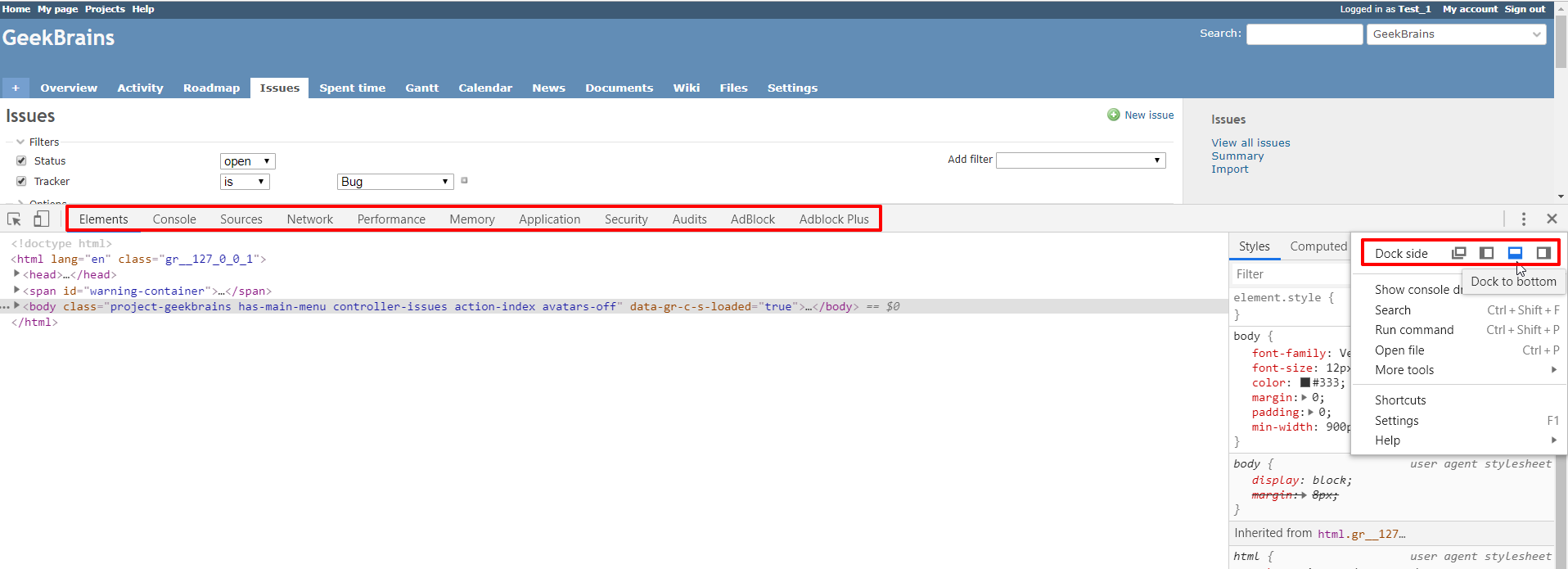
1. Elements.
2. Console.
3. Sources.
4. Network.
5. Performance.
6. Memory.
7. Application.
8. Security.

На начальном этапе нам необходимо знание только первых четырех панелей.

Панель Elements предназначена, чтобы изменять структуру документа HTML (DOM) и его стили (CSS).

Панель Console нужна для просмотра сообщений и запуска JavaScript-кода из самой консоли. Панель Sources используется в основном для дебаггинга JavaScript-кода.

Панель Network предназначена для просмотра запросов и данных, которыми обменивается клиент (браузер) и сервер (удалённый сервер).



Помимо первых четырех панелей инструментов разработчика, существует несколько полезных инструментов, которые могут пригодится во время тестирования:

1. Переключение панели устройства (toggle device bar) — имитация отображения контента на мобильных девайсах.
2. Создание скриншотов при помощи встроенных инструментов в панели разработчика. Для этого используйте комбинацию Ctrl + Shift + P, далее вбейте в строку screenshot и выберите пункт, необходимый вам.
3. Включение режима ограничения (или дросселирования) скорости интернета (Throttling).

# Оформление ошибки в баг-трекере

При оформлении отчетов о дефектах рекомендуется соблюдать определённые правила, которые помогают легче понять, в чем заключается описываемая проблема (дефект).

1. Старайтесь следовать правилу «Что? Где? Когда?».
2. Старайтесь быть лаконичными при описании дефекта в теме дефекта.
3. Каждый дефект должен быть воспроизведен повторно перед написанием отчёта.
4. Отчёт о дефекте должен быть составлен сразу, не откладывайте на потом.
5. Минимизируйте количество шагов в описании.
6. Пишите техническим языком с применением принятой на проекте терминологии.
7. Прикрепляйте дополнительные файлы (логи, скриншоты, видео).

Также есть дополнительные советы, которые могут пригодиться вам в дальнейшей работе на проектах:

1. Старайтесь часть самих логов прикреплять в описании. Это поможет при поиске дефектов по сообщениям об ошибках, которое выдаёт само приложение.
2. Используя видео, вы можете короче описать дефект и избежать недопонимания.
3. Прикрепляйте ссылки к требованиям, это поможет избежать споров и сэкономить время.
4. Не плодите дубликаты дефектов — прежде чем составлять отчёт, спросите у коллег, создавал кто-нибудь такой дефект или нет.
5. Указывайте версию ПО и тестовый стенд (окружение), на котором был обнаружен дефект.
6. Попробуйте воспроизвести дефект, следуя вашим собственным шагам.

Также есть советы, которых стоит придерживаться, когда вы только обнаружили дефект:

1. Найдя баг и удостоверившись, что это баг, сразу составляйте отчёт о дефекте.
2. Найдя баг, постарайтесь сразу запомнить шаги по его воспроизведению.
3. Если отложить составление отчёта о дефекте, то последующее исправление его может быть уже сложнее.
4. Если пропустить баг или пройти мимо, он может вам напомнить о себе с очень плохой стороны в будущем.
5. Найдя баг, точно разберитесь, в чем заключается ошибка.
6. Найдя баг и разобравшись, в чем заключается ошибка, старайтесь точно назначить ему соответствующую степень его важности и срочности.
7. Найдя один дефект, посмотрите рядом — есть ли другие дефекты.
8. Делайте скриншоты дефектов.

# Глоссарий

**Инструмент управления инцидентами** — инструмент, который облегчает запись и отслеживание статуса инцидентов.

**API-тестирование** — тестирование, выполняемое путём отправки команд в сторону приложения используя его API напрямую.

# Практическое задание

**Используя приложенную таблицу, выполните задания:**

1. Ответьте на вопросы в приложенной форме.
2. Составьте чек-лист для раздела «Вебинары» на GeekBrains.
3. \* Решите Challenge 4.
4. \* Зарегистрируйтесь в JIRA, исследуйте функциональные возможности.

## Требования к выполненной работе

1. Домашнее задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, и вместо ФИО укажите свои фамилию и имя.

# Дополнительные материалы

1. [18 Best Bug Tracking Software: Top Defect/Issue Tracking Tools Of 2020](https://www.softwaretestinghelp.com/popular-bug-tracking-software/).
2. [Best Bug Tracking Software](https://www.g2.com/categories/bug-tracking?utf8=%E2%9C%93&order=popular).
3. [Jira Software](https://www.atlassian.com/software/jira).
4. [Знакомимся с REST API](https://geekbrains.ru/events/1260).
5. [Клиент-серверные приложения (только теоретическая часть)](https://geekbrains.ru/events/108).
6. [Обзор всех инструментов разработчика Chrome DevTools](https://habr.com/ru/company/simbirsoft/blog/337116/).
7. [Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools).
8. [Отработка навыка работы с API](https://reqres.in/).
9. [Работа с Jira](https://bytextest.ru/2018/08/31/jira-dlya-novichkov/).
10. [Пример работы с API](https://yandex.ru/dev/direct/doc/start/format-docpage/).
11. [Так ли уж сложно описывать баги понятно?](http://okiseleva.blogspot.ru/2012/12/blog-post_14.html)
12. [Там баг в коде был, поправил, тестируй!](http://okiseleva.blogspot.ru/2016/01/blog-post_18.html)
13. [Мысли о баг-репортах. Часть I. Title, он же «Заголовок»](http://barbaricqa.com/blog/archives/472).
14. [Как эффективно сообщать об ошибках](http://www.chiark.greenend.org.uk/~sgtatham/bugs-ru.html).
15. [10 правил хорошего тона при описании багов](https://habrahabr.ru/company/docsvision/blog/264163/).
16. [Распространенные ошибки при составлении баг-репортов](https://habrahabr.ru/post/156069/).

# Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

1. Software Testing — Base Course (Svyatoslav Kulikov).
2. [Redmine Wiki](https://www.redmine.org/projects/redmine/wiki/RedmineIssueTrackingSetup).
3. [Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools).
4. [Список кодов состояния HTTP](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP).