**Проект №4 — Работа с дефектами и Bug-tracking tools**

Привет, участники Школы 21! В этом проекте вы познакомитесь с тем, какие бывают баги, как их можно отслеживать и как с ними правильно работать. Также мы затронем тему системы контроля версий и изучим много новых терминов. Этот блок потребует от вас немного больше самостоятельности в изучении материала, но мы уверены, что вы справитесь!😊

**Instructions**

Данный проект является первым групповым проектом в курсе, поэтому перед началом работы вам необходимо распределить задачи в команде так, чтобы каждый из вас получил необходимые практические навыки (например, во время тестирования приложения каждый из вас может быть занят проверкой отведённого ему функционала, но ни в коем случае не скидывайте эту задачу на одного участника). Тимлид (иными словами — руководитель вашей команды) должен распределять задачи между участниками, а также контролировать процесс их выполнения.

Напоминаем, что все созданные отчёты и файлы нужно будет загрузить в папку src/ в корне проекта (обязательно в ветку *develop*).

**Contents**

1. [Chapter I](https://edu.21-school.ru/project/62667/task#chapter-i)  
   1.1. [Общая инструкция](https://edu.21-school.ru/project/62667/task#%D0%BE%D0%B1%D1%89%D0%B0%D1%8F-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F)
2. [Chapter II](https://edu.21-school.ru/project/62667/task#chapter-ii)  
   2.1. [Баги](https://edu.21-school.ru/project/62667/task#%D0%B1%D0%B0%D0%B3%D0%B8)  
   2.2. [Задание №1](https://edu.21-school.ru/project/62667/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-1-%D1%80%D0%B0%D0%B7%D0%BD%D0%BE%D0%B2%D0%B8%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C-%D0%B1%D0%B0%D0%B3%D0%BE%D0%B2)  
   2.3. [Локализация бага](https://edu.21-school.ru/project/62667/task#%D0%BB%D0%BE%D0%BA%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%B1%D0%B0%D0%B3%D0%B0)  
   2.4. [Приоритет и серьёзность](https://edu.21-school.ru/project/62667/task#%D0%BF%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82-%D0%B8-%D1%81%D0%B5%D1%80%D1%8C%D1%91%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D1%8C)  
   2.5. [Задание №2](https://edu.21-school.ru/project/62667/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-2-%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B1%D0%B0%D0%B3-%D1%80%D0%B5%D0%BF%D0%BE%D1%80%D1%82%D0%BE%D0%B2)
3. [Chapter III](https://edu.21-school.ru/project/62667/task#chapter-iii)  
   3.1. [Системы управления тестированием](https://edu.21-school.ru/project/62667/task#%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC)  
   3.2. [Задание №3](https://edu.21-school.ru/project/62667/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-3-notion)  
   3.3. [Задание №4](https://edu.21-school.ru/project/62667/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-4-%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D1%82%D0%B5%D1%81%D1%82-%D0%BA%D0%B5%D0%B9%D1%81%D0%BE%D0%B2-%D0%B2-testit)
4. [Chapter IV](https://edu.21-school.ru/project/62667/task#chapter-iv)  
   4.1. [Системы контроля версий](https://edu.21-school.ru/project/62667/task#%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9)  
   4.2. [Репозиторий](https://edu.21-school.ru/project/62667/task" \l "%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9)  
   4.3. [Коммит](https://edu.21-school.ru/project/62667/task" \l "%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82)  
   4.4. [Ветки](https://edu.21-school.ru/project/62667/task#%D0%B2%D0%B5%D1%82%D0%BA%D0%B8)  
   4.5. [Задание №5](https://edu.21-school.ru/project/62667/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-5-git-%D0%B8-gitlab)

**Chapter I**

**Общая инструкция**

Методология Школы 21 может быть не похожа на тот образовательный опыт, который случался с вами ранее. Её отличает высокий уровень автономии: у вас есть задача, вы должны её выполнить. По большей части вам нужно будет самому добывать знания для её решения. Второй важный момент — это peer-to-peer обучение. В образовательном процессе нет менторов и экспертов, перед которыми вы защищаете свой результат. Вы это делаете перед такими же учащимися, как и вы сами. У них есть чек-лист, который поможет им качественно выполнить приёмку вашей работы.

Роль Школы 21 заключается в том, чтобы обеспечить через последовательность заданий и оптимальный уровень поддержки такую траекторию обучения, при которой вы не только освоите hard skills, но и научитесь самообучаться.

* Не доверяйте слухам и предположениям о том, как должно быть оформлено ваше решение. Этот документ является единственным источником, к которому стоит обращаться по большинству вопросов;
* твоё решение будет оцениваться другими учащимися;
* подлежат оцениванию только те файлы, которые вы выложили в GIT (ветка develop, папка src);
* в вашей папке не должно быть лишних файлов — только те, что были указаны в задании;
* не забывайте, что у вас есть доступ к интернету и поисковым системам;
* обсуждение заданий можно вести и в Rocket.Chat;
* будьте внимательны к примерам, указанным в этом документе — они могут иметь важные детали, которые не были оговорены другим способом;
* и да пребудет с вами Сила!

**Chapter II**

**Баги**

Баг (дефект) — это неожиданный/некорректный результат работы программы.

От этапа разработки, на котором будут выявлены баги, зависит очень многое. Чем раньше мы выявим баг, чем раньше сообщим о нём разработчику, тем меньше будет накопления ошибок в программном коде разрабатываемого продукта и тем быстрее пользователи смогут задействовать функции нашего приложения без скрытой угрозы в виде пойманного бага.

Цена каждого найденного дефекта, пусть даже небольшого, разнится на каждой стадии жизненного цикла. Представим ситуацию, как с горы катится снежный ком: если он начинает скатываться с самой верхушки, то селению под горой угрожает нешуточная опасность — а если за два метра до забора, то ничего страшного не произойдёт. Так и здесь: баг, который не был найден на первой стадии жизненного цикла, закравшийся в самое ядро приложения, может отразиться критическими ошибками для пользователя, и именно поэтому ваша работа очень важна.

**Задание №1. Разновидность багов**

А какие бывают баги?🤔 Изучите этот вопрос самостоятельно. В файле exercise1.md опишите, какие существуют типы багов, и к каждому приведите по два примера.

Пример оформления отчёта:

**## Функциональные баги**

Функциональные баги — дефекты, при которых некоторые функции в приложении работают не так, как было задумано. Например, "тут должны быть ваши примеры".

**## ...**

**Локализация бага**

Допустим, вы обнаружили баг. Что делать дальше? Для начала попробуйте его воспроизвести при тех же условиях. Если вдруг воспроизвести баг не удалось, то нужно разобраться, точно ли все условия и шаги были соблюдены. Например, ошибка может воспроизводиться один раз только после обновления страницы. Для того, чтобы более точно определить условия воспроизведения бага, необходимо его локализовать.

Существует несколько шагов для локализации дефекта:

1. Для начала нужно проверить, соответствует ли результат тестирования ожидаемому результату. Справиться с этой задачей помогает техническое задание (в частности, требования к продукту), где должна быть задокументирована информация о тестируемом ПО и его функционировании.
2. Необходимо выявить причины возникновения бага. Он может быть вызван только на стороне пользователя? Или всему виной неверная работа сервера? Про то, как работать с запросами и выявлять ошибки на стороне сервера мы поговорим в шестом проекте, а пока что будем искать баги в клиентской части!🙂
3. Нужно проверить баг в разных браузерах, если эти браузеры также указаны в требованиях в качестве поддерживаемых.
4. Проверить в разных версиях ПО. Можно проверить основную стабильную версию или, если баг был найден уже в основной версии, проверить предыдущую.

После всех проделанных шагов необходимо также зафиксировать результат — это очень поможет разработчикам быстрее понять проблему! Например, вы можете прикрепить логи, если они есть, скриншот (снимок экрана) или скринкаст (запись с экрана).

**Приоритет и серьёзность**

Допустим, в результате тестирования корзины товаров, вы обнаружили 2 бага: "Отсутствие кнопки оплаты заказа" и "Отсутствие отступа в карточке выбранного товара". Какой из багов больше нуждается в быстром исправлении? Разумеется, тот, который не позволяет пользователю продолжать нормально пользоваться всеми функциями приложения, то есть первый баг имеет гораздо больший приоритет, чем второй.

**Приоритет (Priority)** — это специальный атрибут, демонстрирующий очередность выполнения задачи или устранения дефекта. Тут всё просто — приоритет может быть High (высшим), Medium (средним) или Low (низким). Иногда выделяются "Очень высокий" и "Очень низкий" приоритеты. Зачастую *приоритетность дефекта или задачи* проставляют сами разработчики или руководители команд (тимлиды). Тестировщики же определяют *приоритетность тест-кейсов*, степень **серьёзности** бага.

**Серьёзность (Severity) бага** — специальный атрибут, который характеризирует влияние бага на общую функциональность разрабатываемого приложения. Выделяют следующие виды:

* *Blocker* (блокирующий) — баг полностью блокирует выполнение поставленного функционала, и нет никакой возможности его обойти;
* *Critical* (критический) — баг, блокирующий некоторую часть функционала, но существует альтернативный способ для его обхода;
* *Major* (значительный) — баг, демонстрирующий некорректную работу определенной части созданного функционала;
* *Minor* (несущественный) — баг, который по факту не относится к внутреннему функционалу продукта. Например, сюда часто относят баги в удобстве пользования приложением;
* *Trivial* (тривиальный) — баг, который никак не влияет на функционал приложения, прост в исправлении, но всё ещё является багом 🤭. Сюда можно отнести простые баги в контенте, простые визуальные баги.

Говоря про приоритет и серьёзность, стоит сказать про хотфиксы и патчи, с этими словами вы нередко будете сталкиваться в своей работе. Хотфикс (hotfix) — дословно переводится с английского как "горячее исправление" — это срочное исправление ошибки или уязвимости в приложении, преимущественно исправляют баги в каком-то одном модуле программы. Патч (с английского patch — заплатка), иногда называемый просто фиксом (исправлением), представляет собой небольшой фрагмент программного кода/обеспечения, который используется для исправления каких-либо дефектов. Ключевым отличием хотфикса от патча или фикса является скорость реализации.

После выявления бага, его локализации, определения приоритетности и серьёзности, составляется баг-репорт, о котором мы говорили в одном из предыдущих проектов.

**Задание №2. Составление баг-репортов**

Перейдите на [страницу с тестовой формой авторизации](http://testingchallenges.thetestingmap.org/login/login.php). Данная форма авторизации должна реализовывать следующие требования:

* Корректными данными для авторизации являются: username — admin, password — admin;
* В форме авторизации выводится сообщение Logged In Successfully при вводе корректных авторизационных данных;
* В форме авторизации выводится сообщение Enter Both Username And Password, если хотя бы одно из полей не заполнено;
* В форме авторизации выводится сообщение Username Not Found, если введено любое другое значение username;
* В форме авторизации выводится сообщение Incorrect password, если введён верный username и неверный пароль;
* Все сообщения выводятся над кнопкой Log In;
* Форма авторизации предлагает выбрать один из четырёх языков интерфейса: английский, французский, итальянский, немецкий;
* Выбранный язык не сбрасывается после попытки авторизации.

Проведите интуитивное тестирование, сравнивая получаемые результаты с описанными выше требованиями. Если фактический результат расходится с требованиями, то опишите найденный баг (составьте баг-репорт). Не забудьте указать к каждому багу степень его серьёзности! Все баг-репорты сохраните в файле exercise2.md, разделяя каждый из них заголовками. Например:

**# Баг-репорты**

**## Баг №1**

...

**## Баг №2**

...

**Chapter III**

**Системы управления тестированием**

Когда тестирование выходит за рамки обычной проверки формы авторизации, когда речь заходит о каком-нибудь большом веб-приложении с множеством функционала, то нужно начинать думать, где мы будем хранить нашу тестовую документацию. Мы ведь не будем всё это хранить в текстовых документах или excel-табличках? Тут нам на помощь приходят системы управления тестированием (Test Management System, TMS)! Системы управления тестированием используются для хранения информации о том, как должным образом проводить тестирование. Они помогают оптимизировать процесс тестирования, позволяют создавать целые тест-планы и выстраивать стратегию тестирования.

В этом курсе мы поработаем с системой [TestIT](https://testit.software/). Однако, так как в бесплатной версии веб-приложения документация в пространстве может храниться не больше 14 дней, позже мы будем также работать с [Notion](https://www.notion.so/).

**Задание №3. Notion**

На данный момент одним из самых используемых инструментов для создания и хранения входных артефактов является [Confluence](https://www.atlassian.com/software/confluence). Бесплатная версия Confluence сейчас недоступна, поэтому в проекте будет использоваться схожий по функциональности инструмент Notion. Перейдите на официальный сайт [Notion](https://www.notion.so/) и зарегистрируйтесь.

В Notion создаётся пространство команды, и аналитик размещает информацию о том, чего хотят клиенты или бизнес-пользователи, создаёт диаграммы. Каждое пространство в Notion предназначено для хранения полезной информации по какому-либо проекту. Как правило, пространство называется так же, как и проект в Jira или любой другой системе учёта компании. В пространстве можно создавать статьи, добавлять их в избранное, подписываться на изменения статей или целого пространства, получать новости из блога и много другое. Первым делом руководителю команды необходимо создать *teamspace* (рабочее пространство для команды) и добавить туда всех участников команды.

Далее вашей команде нужно спланировать пространство таким образом, чтобы в нём была структура, понятная для всех участников команды. Обратите внимание, что наименование страниц при этом должно быть расширяемым и удобным для использования в долгосрочном проекте.

Вам нужно будет протестировать некоторые части системы [TestIT](https://testit.software/), которую мы упоминали ранее. TestIT — это современная система управления тестированием. Перейдите на официальный сайт этой системы, затем зарегистрируйтесь в облаке. Тимлид должен создать общее рабочее пространство и добавить туда остальных участников команды:

В рабочем пространстве Notion создайте таблицу, в которой опишите все необходимые сценарии для тестирования всех функциональных возможностей, которые предоставляет страница "Ручные тесты" в системе TestIT. Таблица должна содержать минимум 3 колонки:

* Номер сценария
* Название
* Шаги

Пример таблицы:

Полученную таблицу экспортируйте в формате PDF и загрузите в папку src/ в корень проекта под названием exercise3.pdf.

**Задание №4. Создание тест-кейсов в TestIT**

В TestIT в своём пространстве создайте новый проект (это может снова сделать руководитель). Если время рабочего пространства уже на исходе, то потребуется создать новое! На основе сценариев, полученных в предыдущем задании, составьте тест-кейсы. Не забудьте описать предусловия, ожидаемые результаты, к каждому тест-кейсу добавить приоритет (это важно для порядка, в котором данные тест-кейсы будут выполнять) и время, которое потребуется на его выполнение.

Полученные тест-кейсы экспортируйте в формате xlsx и загрузите в папку src/ в корень проекта под названием exercise4.xlsx.

**Chapter IV**

**Системы контроля версий**

Всё это время в тексте проектов и заданий, в общих чатах вы могли встречаться со словами "ветки", develop, merge request'ы (они же MR), "пуши" и прочее. Настало время разобраться в том, что это такое.😁

Разработчики, пишущие код, где-то должны его хранить, возвращаться к старым версиям, обмениваться им между собой. Для этих целей идеально подходит система контроля версий. Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Более подробно о системах контроля версий вы можете узнать по [этой ссылке](https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control). Выделяют локальные, централизованные и распределенные системы контроля версий. Подробно мы их рассмотрим позже на примере репозитория.

Самой популярной подобной системой является Git (отсюда название таких популярных систем, как GitLab и GitHub). Git показывает очень высокую производительность в сравнении со множеством альтернатив.

**Репозиторий**

Ключевым объектом, который лежит в основе Git'а, является репозиторий. Репозиторий — это все файлы, находящиеся под контролем версий, вместе с историей их изменения и другой служебной информацией. Это очень похоже на обычную папку, только которая хранит в себе историю любых изменений, позволяет создавать копии и уже в них вносить изменения, а после "сливать" их друг с другом.

Репозиторий Git можно создать, либо выбрав любую папку на компьютере, либо клонировав себе уже существующий репозиторий с платформы.

В *локальных* системах контроля версий репозиторий хранится и используется на одном устройстве, но работать с такой системой может только один разработчик. В *централизованной* системе репозиторий хранится на одном сервере. Работать с такой системой может сразу несколько клиентов. Но лучше всего для большого количества разработчиков подходят *распределенные* системы контроля версий, к которым относится и Git. Такая система представляет собой облачное хранилище: каждый пользователь хранит на своем устройстве весь репозиторий целиком, и по мере изменения репозитории синхронизируются.

**Коммит**

**Commit** (с английского "фиксировать") — это операция, которая берет все выбранные изменения и отправляет их в репозиторий. Коммиты позволяют разбить процесс разработки на отдельные шаги. То есть коммит — это та точка, к которой можно вернуться, если в процессе разработки возникнут какие-то проблемы.

Коммит помимо изменений содержит в себе название, время внесения изменений, имя автора, собственный ID, ID родительского коммита и прочее. В GitLab'е это выглядит следующим образом:

Изменения в рамках одного коммита подчиняются определенным, установленным командой разработчиков правилам и рекомендациям, касающимся именования, описания и содержания коммитов. Как правило, рабочий процесс представляет собой цикл: коммит — изменение файлов — коммит.

**Ветки**

Представьте, что несколько разбработчиков будут работать в одной папке одновременно. Если они будут вносить правки в один и тот же файл, будут возникать конфликты и другие неудобства. А если один разработчик внёс изменения, а другой всё ещё правит старую версию файла? Ветвление позволяет создавать под каждую задачу создавать отдельную *ветку* разработки, не затрагивая основную версию. То есть использование ветвления позволяет решать отдельные задачи, не вмешиваясь в основную линию разработки.

По умолчанию, имя основной ветки в Git — main (ранее было *master*).

P.S.: Всю информацию про Git можно найти в [официальной документации](https://git-scm.com/doc).

**Задание №5. Git и GitLab**

Установите к себе на компьютер [Git](https://git-scm.com/). Перейдите на официальный сайт [GitLab](https://about.gitlab.com/) и зарегистрируйтесь. Руководитель команды должен создать в GitLab **публичную** группу, добавить в неё всех участников команды с ролью Maintainer. Используя логин тимлида на обучающей платформе Школы 21, группу нужно назвать следующим образом: school21\_логин-тимлида, например, school21\_asumia.

1. В группе создайте **публичный** пустой репозиторий ("Create blank project");
2. Инициализируйте репозиторий и добавьте в main ветку файл README.md;
3. Склонируйте данный репозиторий себе на компьютер;
4. Пусть каждый участник команды в файле README.md напишет свой логин (с платформы Школы 21) и закоммитит изменения в main ветку;
5. Создайте ветку develop;
6. Сделайте develop ветку основной (вместо main);
7. В develop ветке в файле README.md вместо описания своей команды ответьте на следующие вопросы:
   * Что такое git bash?
   * Что делает команда git pull? Чем она отличается от git push?
   * Что такое merge request?
   * Чем между собой отличаются команды git status и git log?
   * Что такое submodule? С помощью какой команды можно добавлять сабмодули в свой репозиторий?
8. Далее исследуйте *доску задач GitLab*. Для этого можно воспользоваться [документацией GitLab Issues](https://docs.gitlab.com/ee/user/project/issues/managing_issues.html). На доску задач добавьте задачу с названием Test issue и заассайните (Assign) её на своего тимлида. Создайте на основе этой задачи merge request.

После выполнения вышеперечисленных задач создайте файл exercise5.md и вставьте в него ссылку на свою группу GitLab.

**Double-check**

Перед загрузкой выполненного проекта в репозиторий перепроверьте наличие всех необходимых файлов, которые требовалось создать во время его выполнения:

exercise1.md

exercise2.md

exercise3.pdf

exercise4.xlsx

exercise5.md

💡 [Нажми здесь](https://forms.gle/cqhdZsEa1x1uTPEY7) **чтобы отправить обратную связь по проекту**.