Записки код-ревьюера:

мыслим выше,
чем пробелы и табуляция

Александра
Качина

бизнес
вконтакте

ВКонтакте для бизнеса — ваш гид в мир бизнеса в крупнейшей российской социальной сети

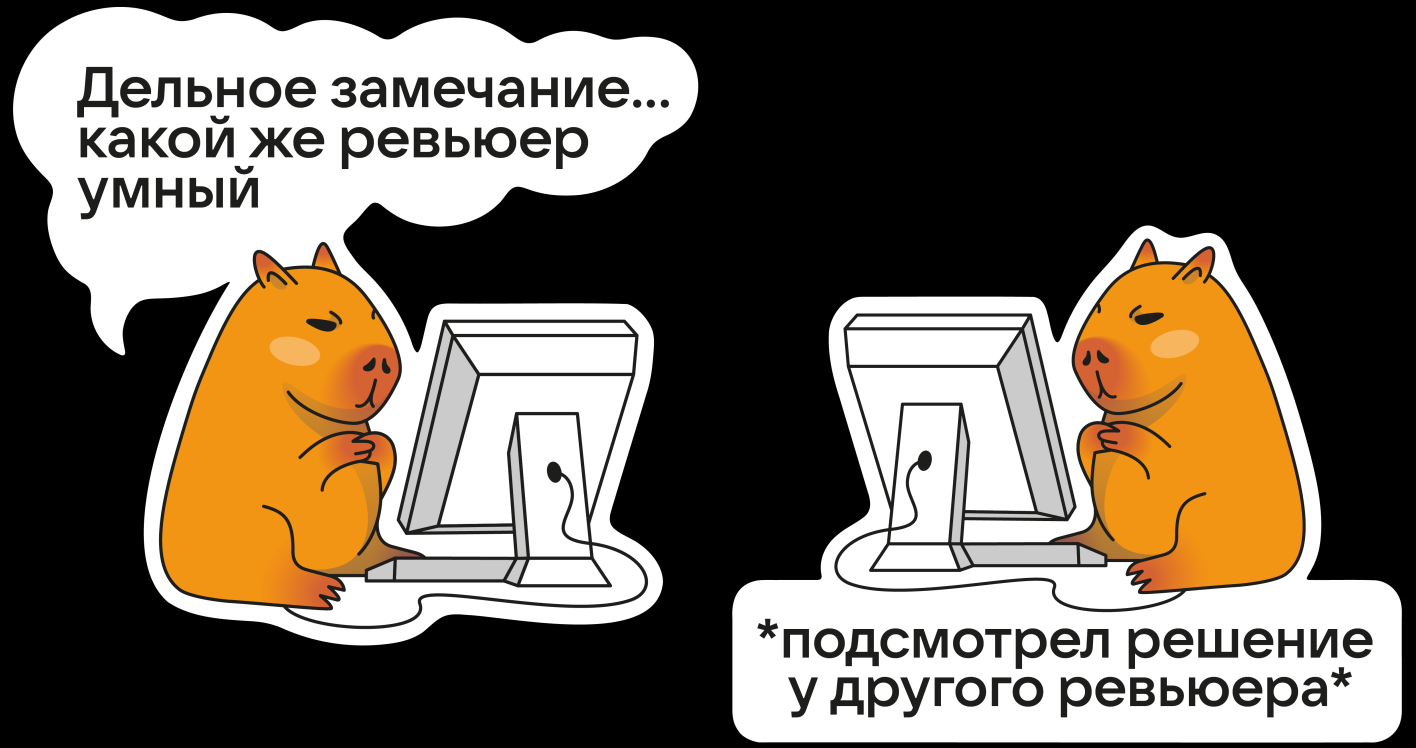
Стек

- Java
- Kotlin
- Swift
- Selenide
- Espresso
- Gradle
- TeamCity
- Allure
- TestOps
- Xcode
- XCUI Test
- Android Studio
- Charles Postman
- Swagger
- TestNG
- AssertJ

План доклада

Зачем нужен код-ревью для моего кода, и почему я должен писать комментарии в чужом коде?

1. Улучшить наше решение
2. Найти ошибки в коде
3. Поделиться собственным опытом
4. Научиться новым подходам



Что можно делать до отправки кода на merge request?

Используем возможности среды разработки

- Реформатор кода
- Пресеты
- Внешние утилиты
- Хуки
- Ошибки и предупреждения

Проблема: одинаковые комментарии о стиле кода

```
@Test(description = "Проверка открытия формы логина с главной страницы при нажатии на ‘Войти'", testName = "Открытие формы логина с главной")
```

В Code Style написано, что после и до скобок должен быть перенос строки

```
@Test(
        description = "Проверка открытия формы логина с главной страницы при нажатии на ‘Войти'",
        testName = "Открытие формы логина с главной"
)
```

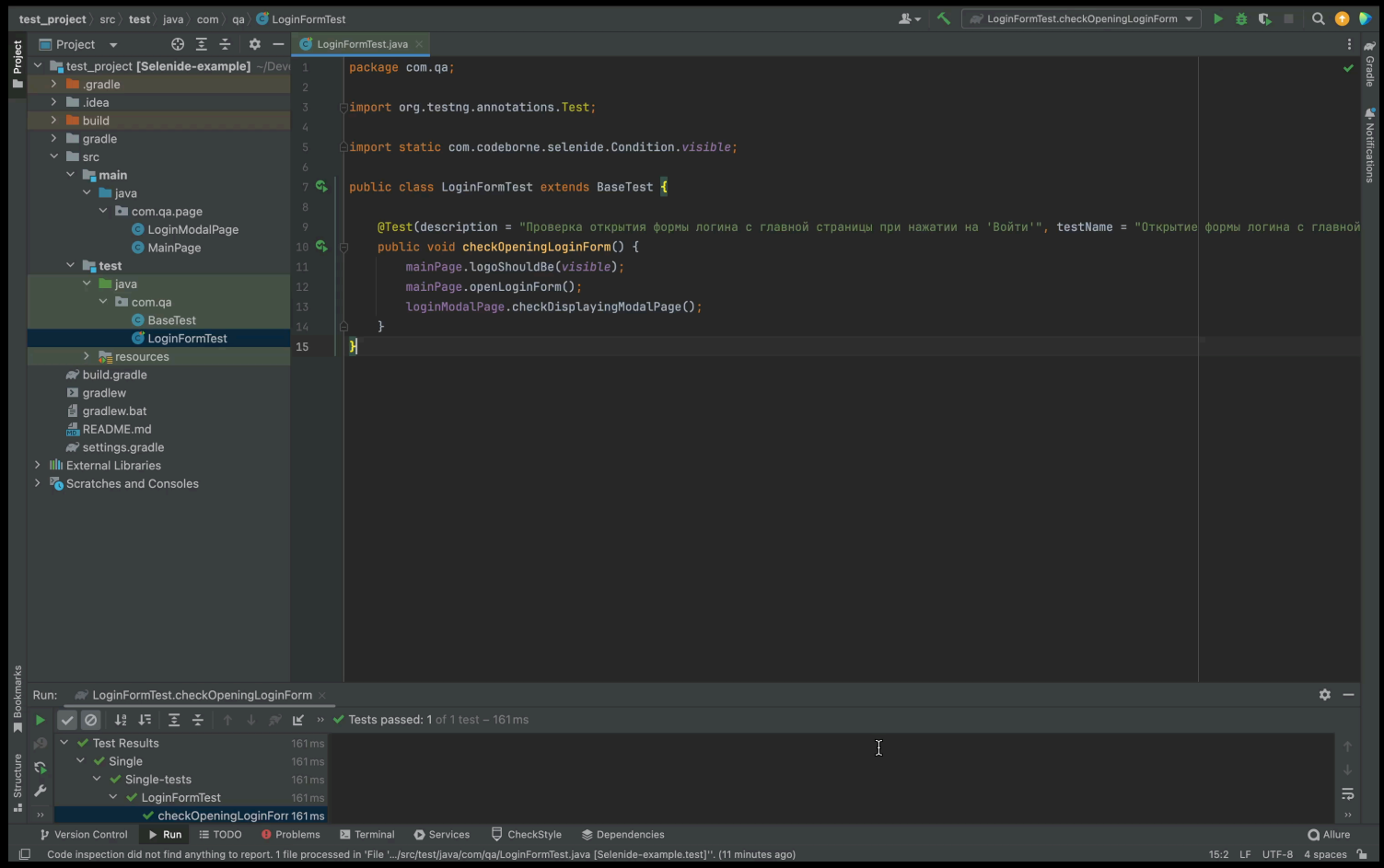
Исправь отступы, у нас в проекте используют 4 пробела

Используем Reformat Code

1. Code Style Settings: создаём кастомные настройки и импортируем

Использование:
Code → Reformat code

Настройка:
Settings → Editor → Code Style



Используем Reformat Code

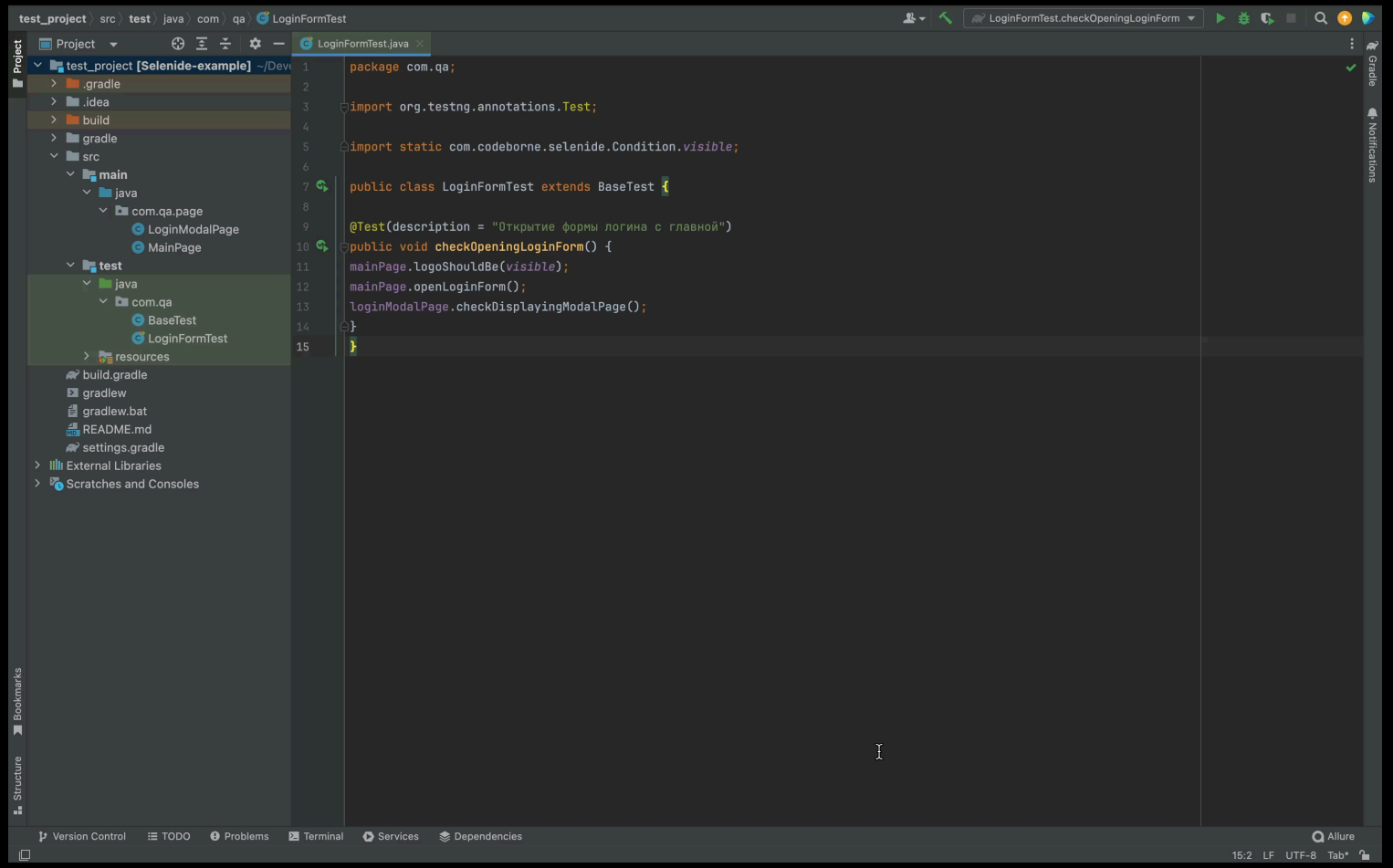
2. Используем в проекте файл .editorconfig

Настройка:

Settings → Editor → Code Style

Scheme: Default - стандартное форматирование

Enable EditorConfig support ☑



Используем Reformat Code

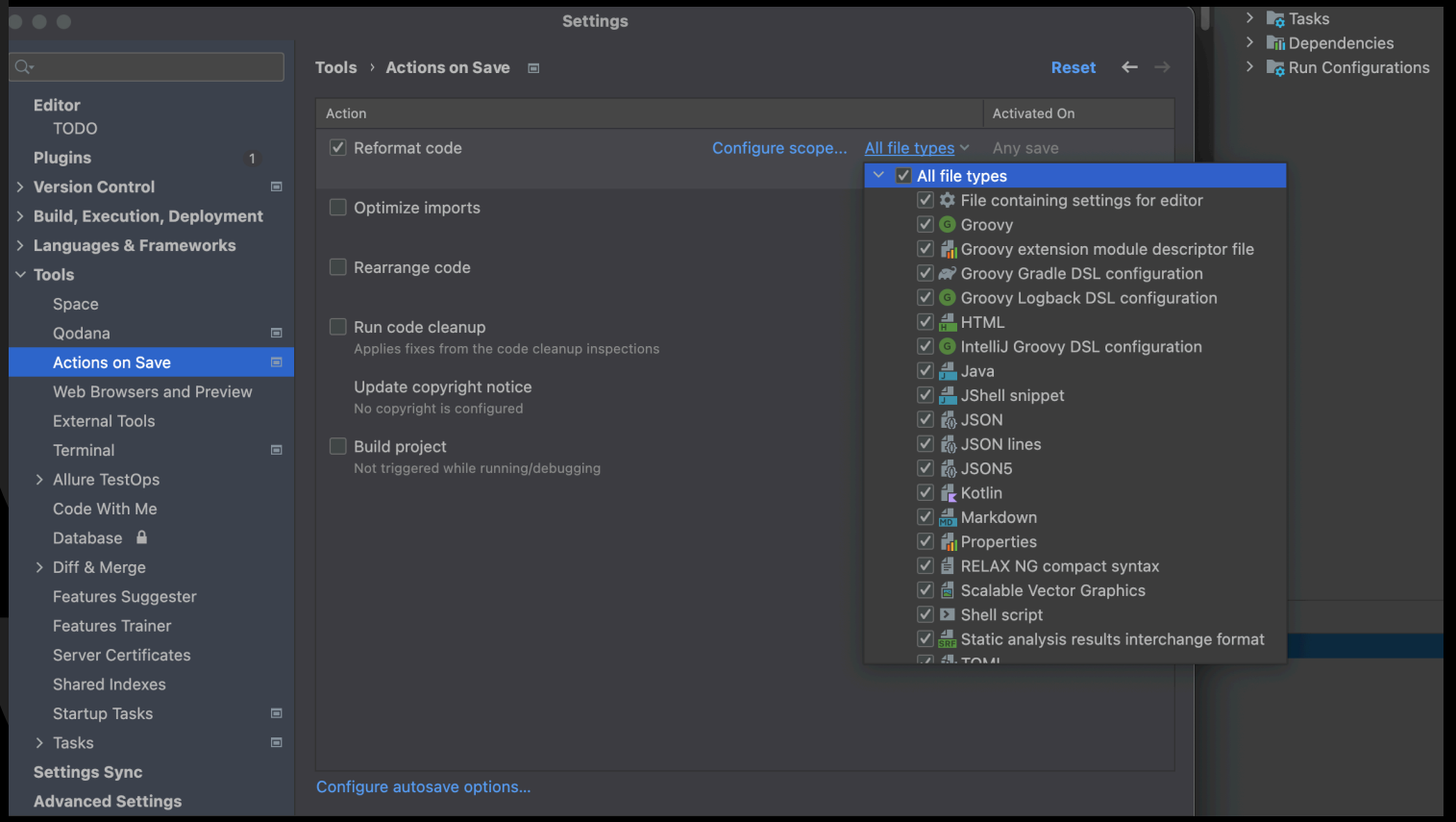
3. Автоматизируем переформатирование кода при сохранении изменённых файлов

Настройка:

Settings → Tools → Actions on Save

✓ **Reformat code**

- Configure scope
- All file types
- Whole file
- Changed lines



Внешние утилиты

Инструмент для изменения физического вида кода: добавить отступы, пробелы, переносы строк и т.д

Примеры:

- Prettier — JavaScript
- Clang-Format — C, C++, Objective-C
- Black — Python
- Gofmt — Go

Настройка

1. Установка утилиты с помощью систем управления пакетами
2. Настройка файла конфигурации в корне проекта / Интеграция с IDE через плагин / Настройка автоматических проверок / Интеграция с Git

В отличие от линтеров они не проверяют код на ошибки, которые потенциально способны причинить ущерб работоспособности

Git Hooks

Скрипты, которые Git выполняет перед или после того, как определенные важные действия произведены, например, commit, push или pull

Клиентские хуки:

1. pre-commit — проверка соответствия кода стандартам, запуск тестов, автоматическое форматирование
2. commit-msg — проверка содержания и формата коммит-сообщения
3. post-commit — уведомление о коммите

Серверные хуки:

1. pre-receive — проверка и изменение входящих коммитов
2. update — для вызова на каждый пуш ветки
3. post-receive — для развертывания кода и отправки уведомлений

Git Hooks

Как настроить хуки

1. Каталог .git/hooks с файлами расширения .sample

2. Создать файл без расширения и сделать его исполняемым:

```
touch pre-commit # Создает файл хука pre-commit
chmod +x pre-commit # Делает файл исполняемым
```

3. Создание скрипта

```
#!/bin/sh
# pre-commit
# Запуск линтера или утилиты для проверки стиля кодирования
bad_code_style=$(lint-command)

# Если линтер нашёл ошибки, отменяем коммит
if [ ! -z "$bad_code_style" ]; then
echo "Код не соответствует стилю кодирования. Пожалуйста, исправьте ошибки перед коммитом."
exit 1
fi
```

Проблема: комментарии об отсутствии однотипных параметров

```
@Owner(KAPIBAGA)
@Severity(NORMAL)
@AllureId("111111")
@Test(description = "Проверка открытия формы логина с главной")
```

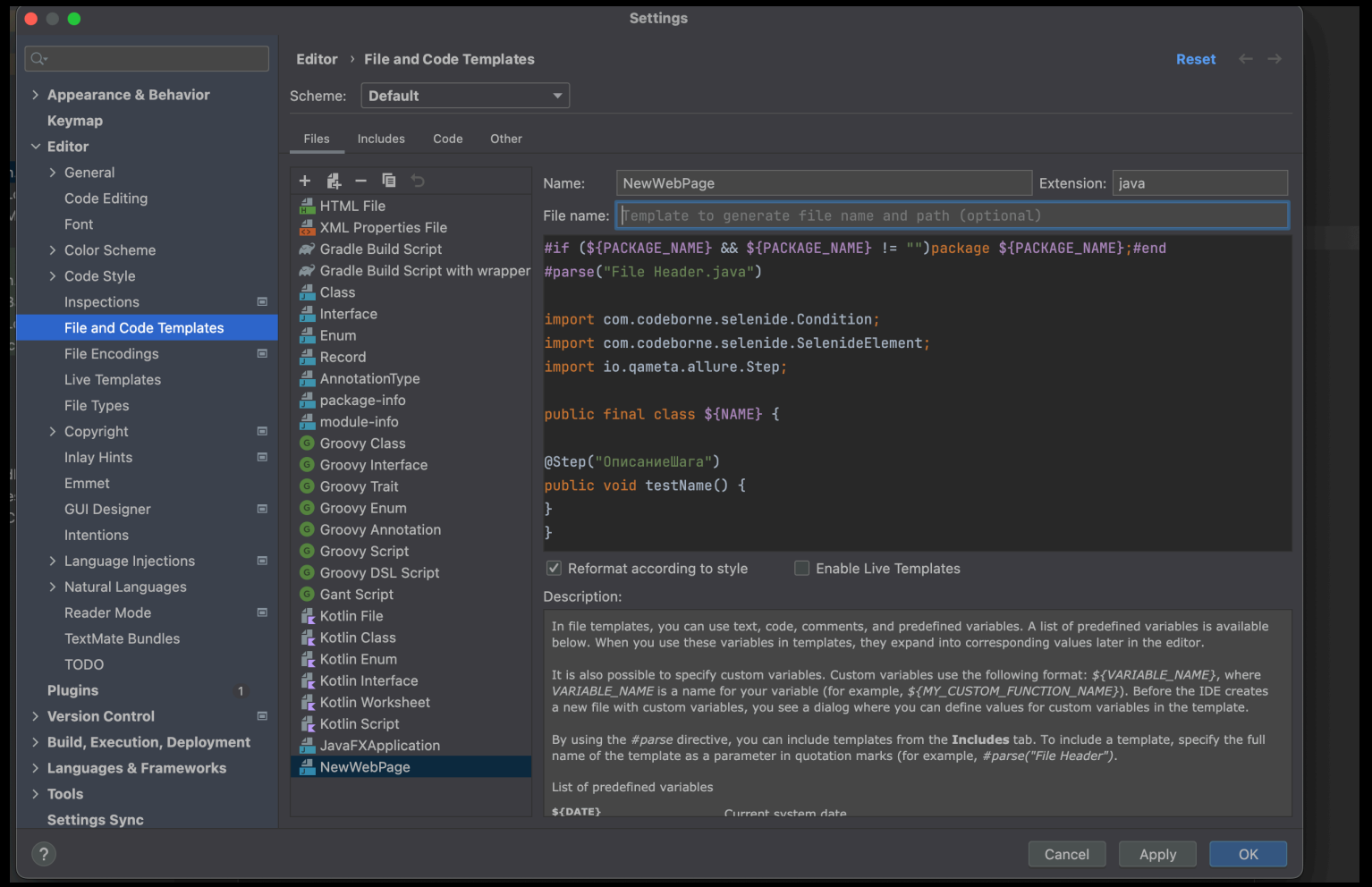
Кажется, ты забыл указать аннотацию с меткой @SprinterLabels(SMB). Она нужна, чтобы размечать тесты нашего бизнес-юнита

Пресеты

Шаблоны для создания новых файлов

Настройка:

Settings → Editor → File and Code Templates

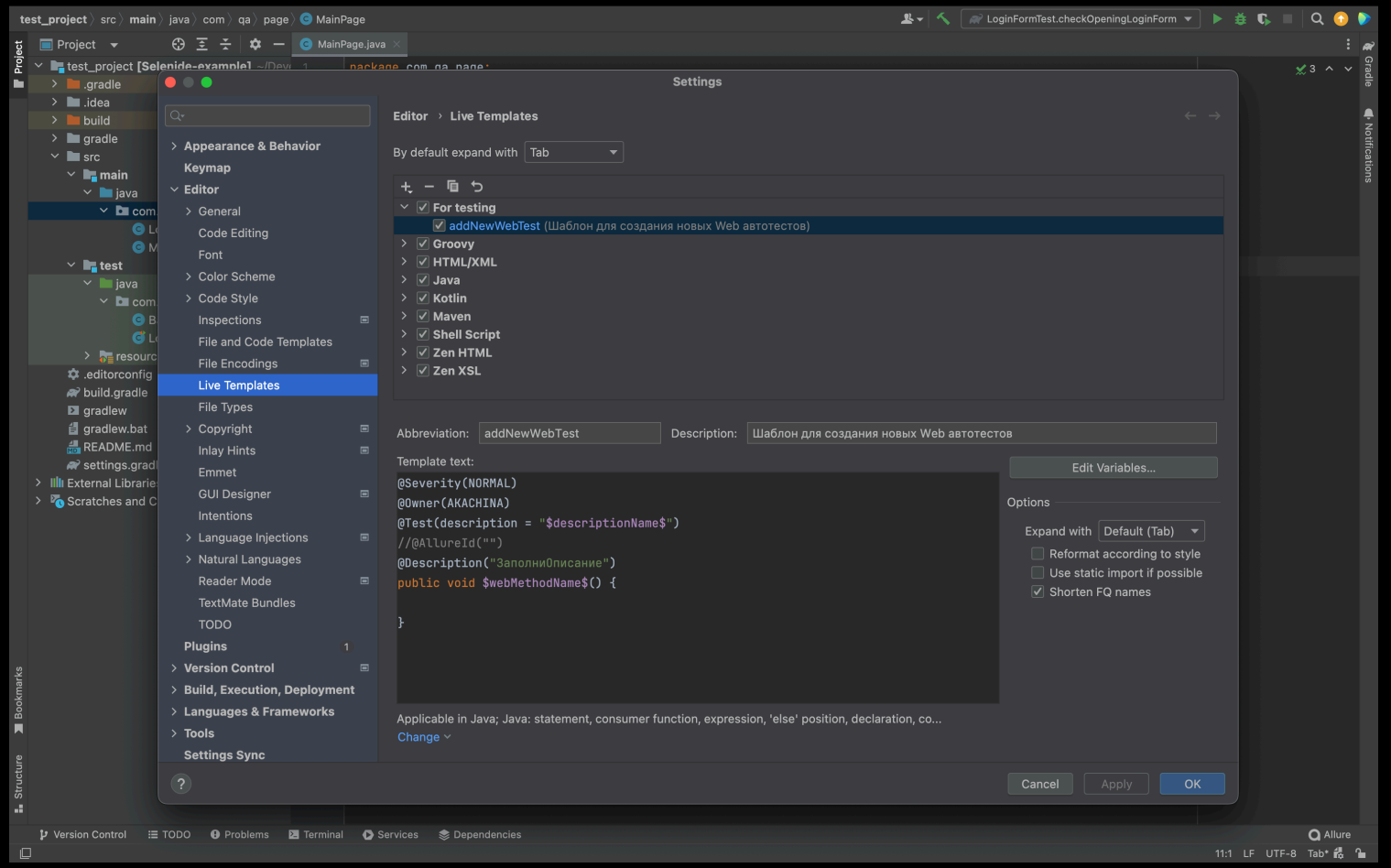


Пресеты

Шаблоны для создания одинаковых блоков кода

Настройка:

Settings → Editor → Live Templates



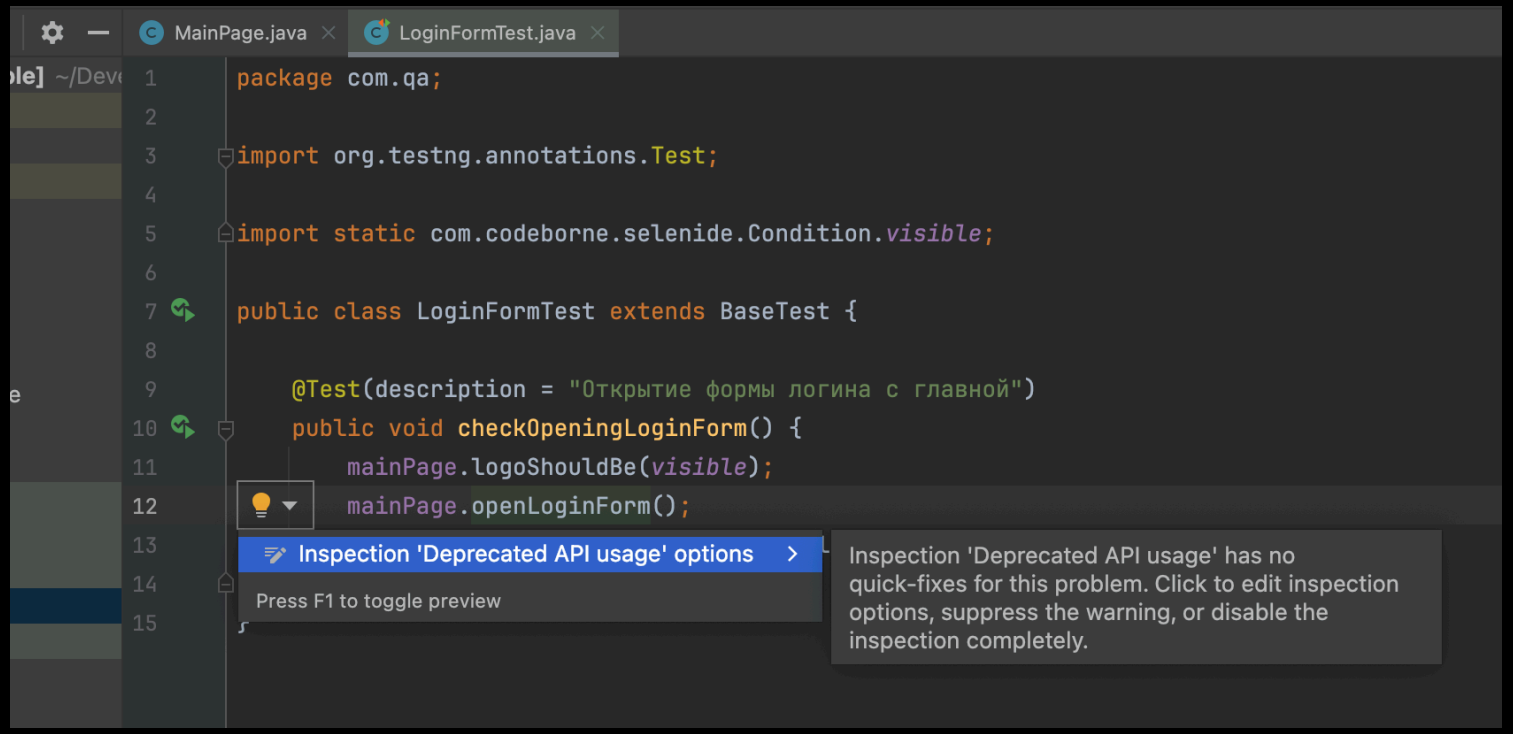
Проблема: ревьюеры повторяют замечания среды разработки

```
public void checkOpeningLoginForm() {
    mainPage.logoShouldBe(visible);
    mainPage.openLoginForm();
```

Метод openLoginForm() не нужно использовать, он устарел. Замени его на showLoginForm()

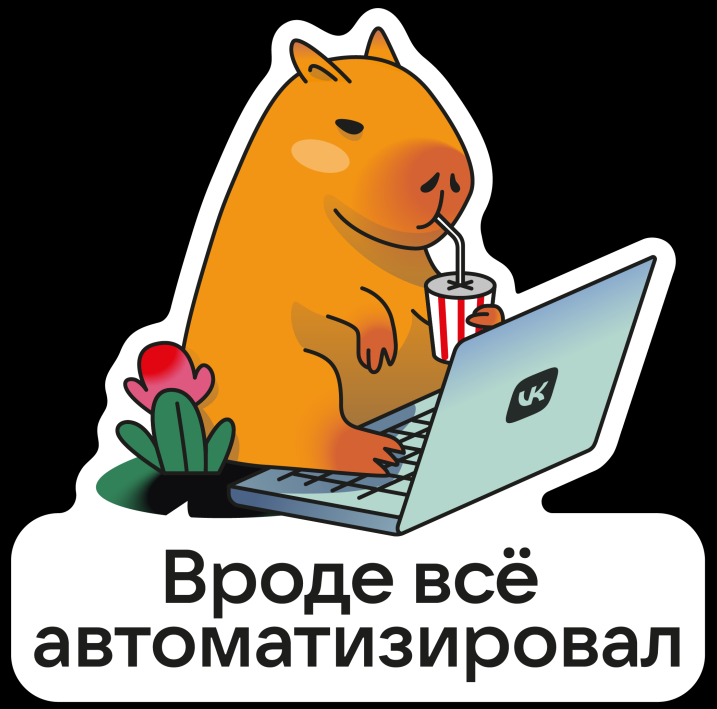
Errors Warnings

- Использование устаревших конструкций
- Необработанные исключения
- Потенциально неправильное использование логики программы



Что можно делать во время ревью merge request'а?

1. Линтеры
2. Боты
3. Ревью в IDE
4. Функции комментирования



Проблема:
авторы кода забывают проверять код-стайл. Ревьюеры пишут одинаковые комментарии

```
public void checkOpeningLoginForm() {
mainPage.logoShouldBe(visible);
mainPage.openLoginForm();
```

Здесь снова проблема с отступами :(

Линтеры

Инструмент для анализа кода, который помогает обнаруживать ошибки, несоответствия стилевым стандартам и потенциальные уязвимости для повышения качества и безопасности исходного кода

Примеры:

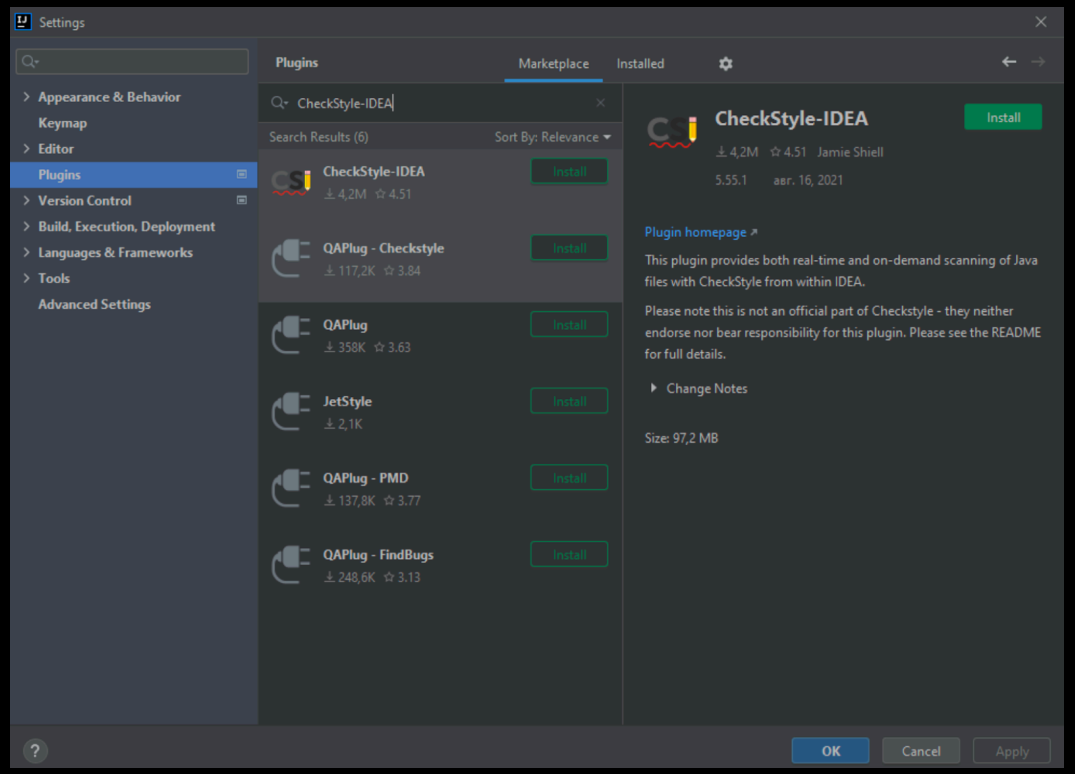
- ESLint - JavaScript
- RuboCop - Ruby
- Pylint - Python
- Checkstyle - Java

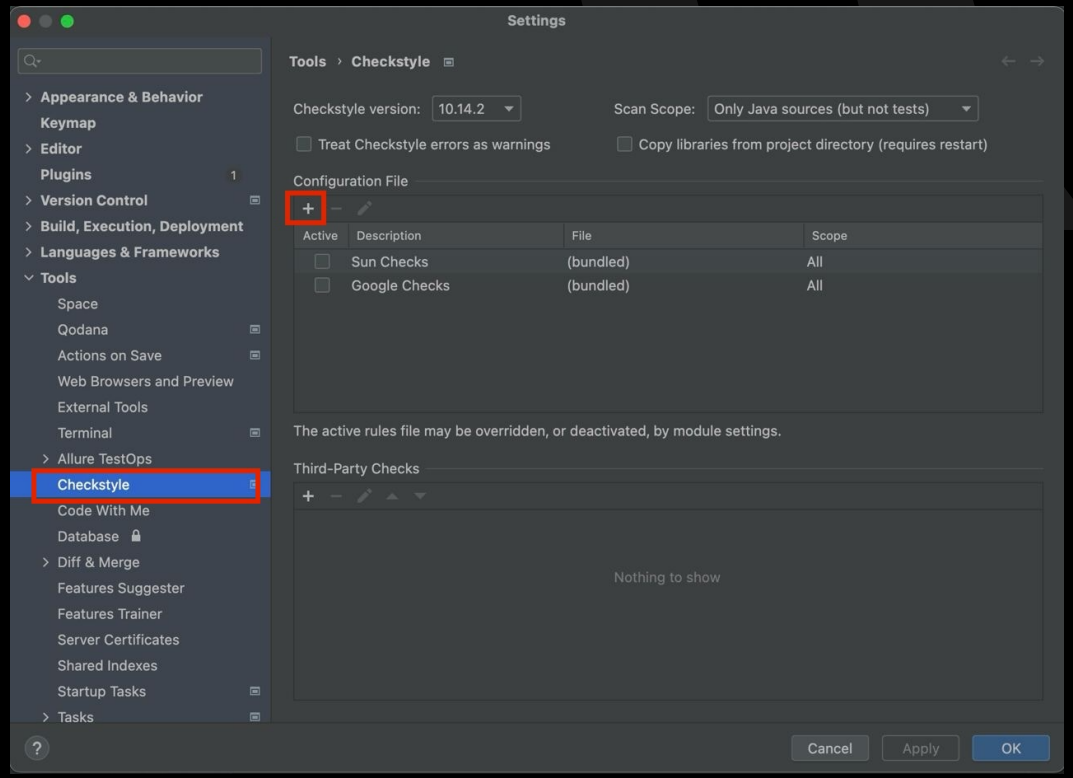
Линтеры

```
<configuration>
    <module name="Checker">
        <module name="TreeWalker">
            <module name="Indentation">
                <property name="basicOffset" value="4" />
                <property name="braceAdjustment" value="0" />
                <property name="caseIndent" value="4" />
            </module>
            <module name="MethodName">
                <property name="format" value="^[a-z][a-zA-Z0-9]*$" />
            </module>
            ...
        </module>
    </module>
</configuration>
```

```
Запуск из командной строки
java -jar <path_to_checkstyle_jar> -c <path_to_checkstyle_xml> <path_to_source_files>
```

Линтеры





Настройка плагина:
Settings → Plugins

Конфигурация для своего проекта:
Settings → Other Settings → Checkstyle

Линтеры

Настройка линтера в систему сборки на примере Checkstyle и gradle

1. Подготовить файл конфигурации checkstyle.xml
2. Добавить плагин Checkstyle в build.gradle

```
plugins {
  // Применить плагин Checkstyle к проекту
  id 'checkstyle'
```

3. Настроить Checkstyle в build.gradle

```
checkstyle {
  toolVersion = '8.29'  // Указывает версию Checkstyle
  configFile = file("${project.rootDir}/config/checkstyle/checkstyle.xml")  // Путь к файлу конфигурации
  ignoreFailures = false  // Определяет, должен ли сборка продолжаться, если обнаружены ошибка стиля
  sourceSets = [sourceSets.main] // Директории или файлы для проверки
}
```

4. Запустить Checkstyle

```
./gradlew checkstyleMain  // Проверить основной исходный код в src/main/java
./gradlew checkstyleTest  // Проверить код тестов в src/test/java
```

Отчёты в формате XML и HTML в ${projectDir}/build/reports/checkstyle/

Линтеры

Настройка линтера в систему сборки на примере Checkstyle и gradle

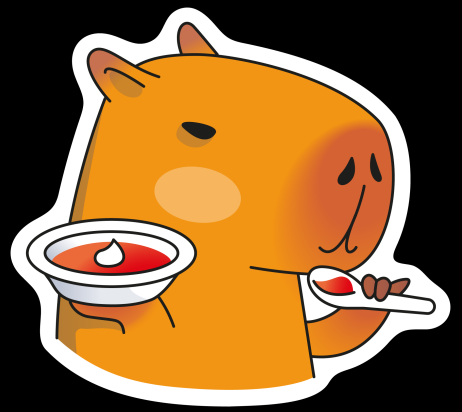
5 Настроить порядок выполнения во время сборки в build.gradle

```
check.shouldRunAfter checkstyleMain, checkstyleTest
```

6 Настроить проверку на CI. Пример файла .gitlab-ci.yml

```
checkstyle:
  stage: checkstyle
  script:
    - ./gradlew checkstyleMain checkstyleTest  # Запускаем задачи Checkstyle
  only:
    - merge_requests  # Определяем, что задача Checkstyle запускается только для merge
request'ов
  artifacts:  # Сохраняем отчеты Checkstyle в качестве артефактов
    paths:
      - build/reports/checkstyle  # Путь к отчетам Checkstyle
    expire_in: 1 week  # Устанавливаем срок хранения артефактов — 1 неделя
```

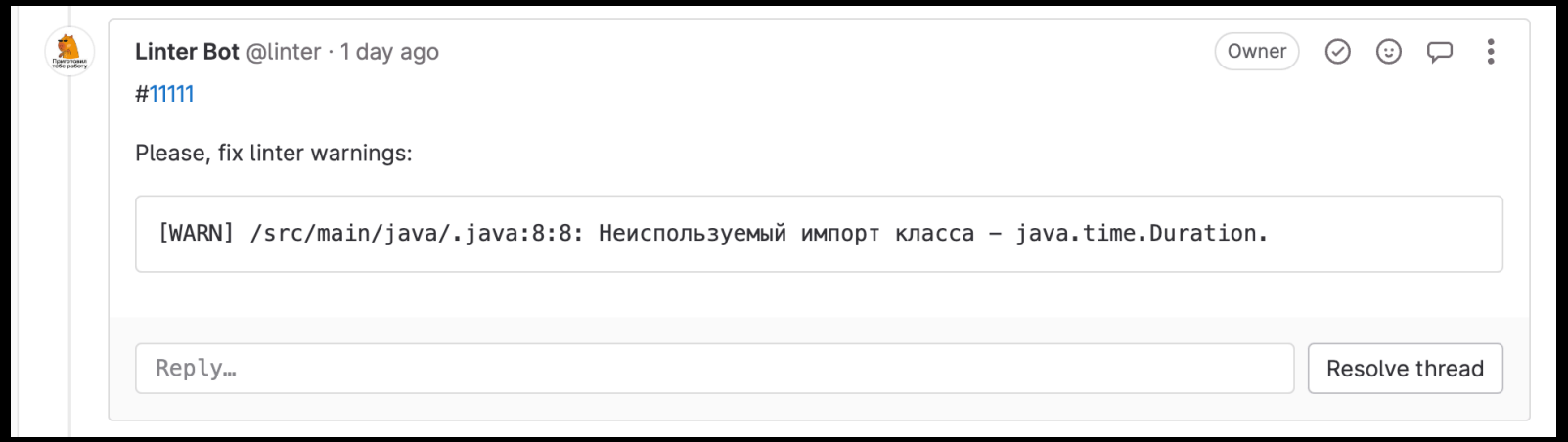
Боты для линтеров



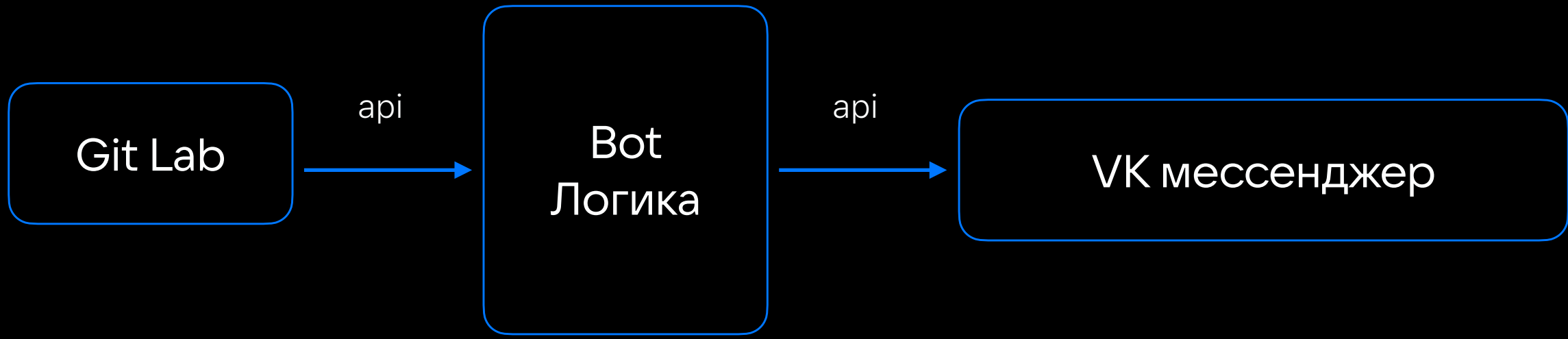
Получаем данные

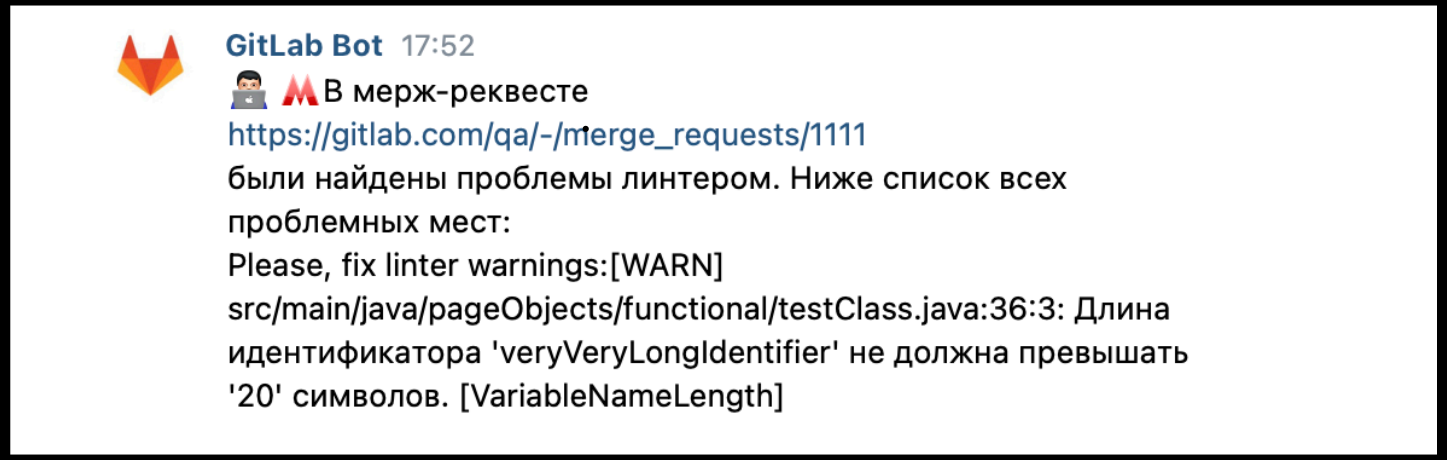
Анализируем данные

Пишем в комментарий merge request'а

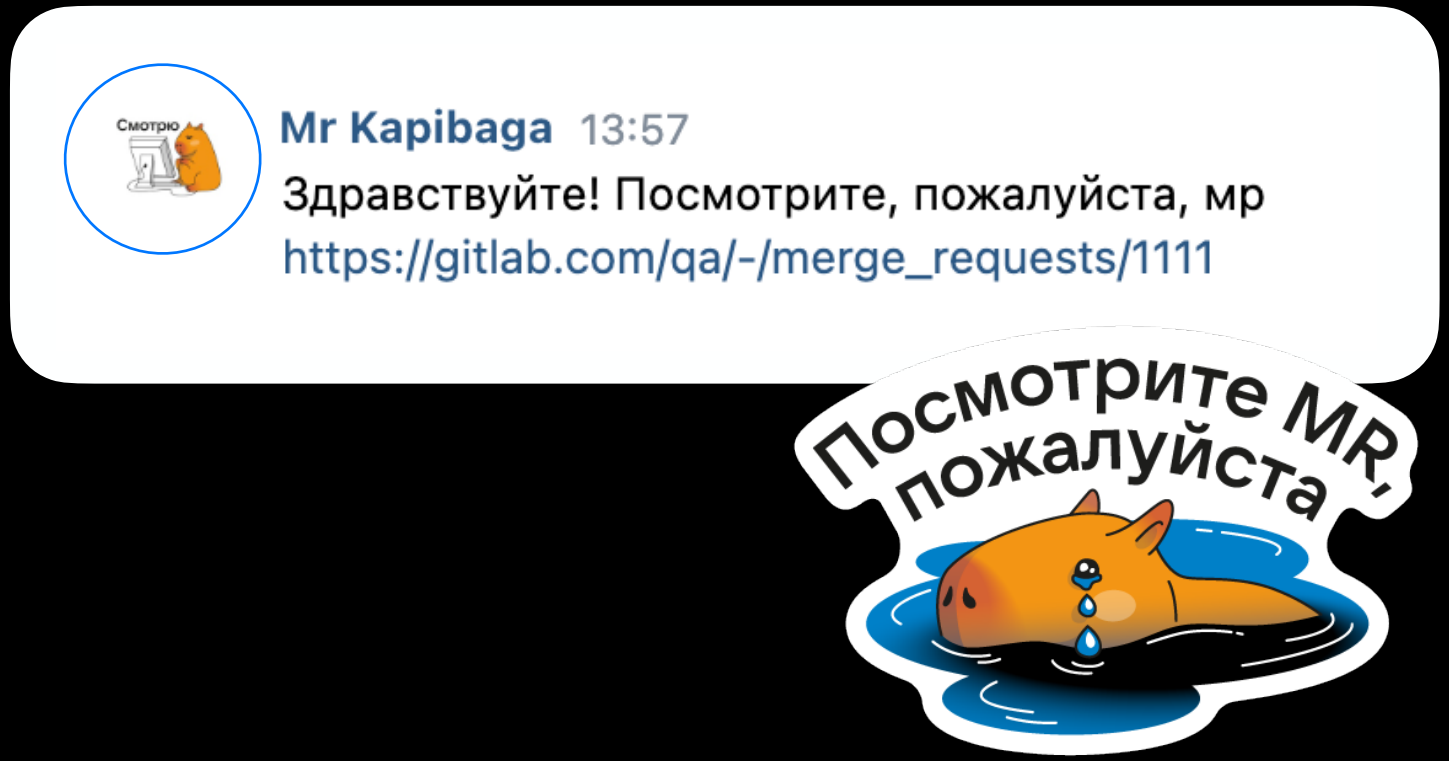


Боты для линтеров

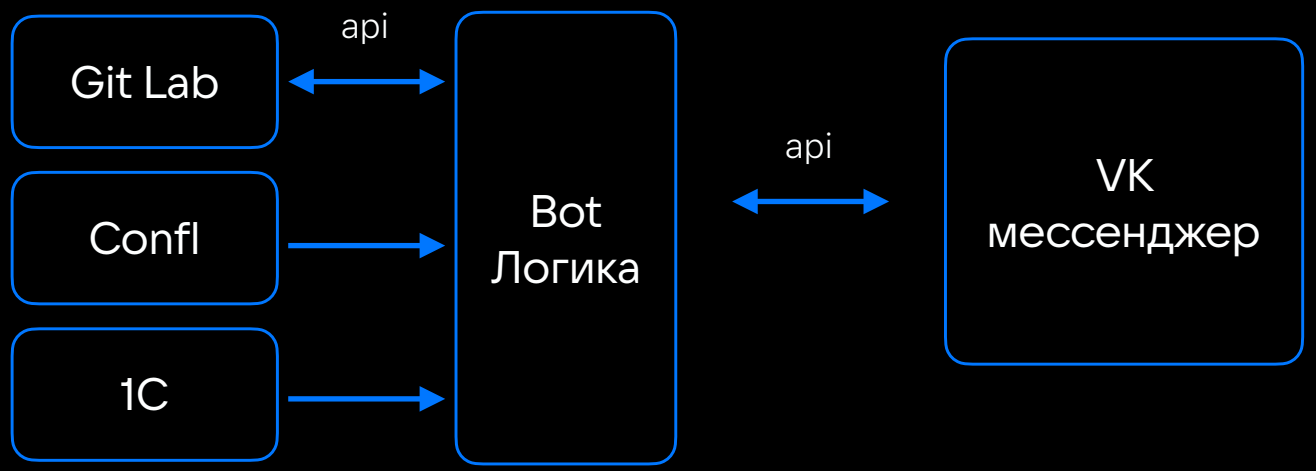




Проблема: ревьюеры не подключаются к MR'ам



Боты для чатов



Ссылка на проект	Ссылка на группу ревьюверов	Люди – исключения для назначения	Слать ли уведомления о новых MP в личку?	Сколько ревьюверов назначать?
12121 (ID)	1221 (ID)	kapibaga_reviewer@vk.team	true	1

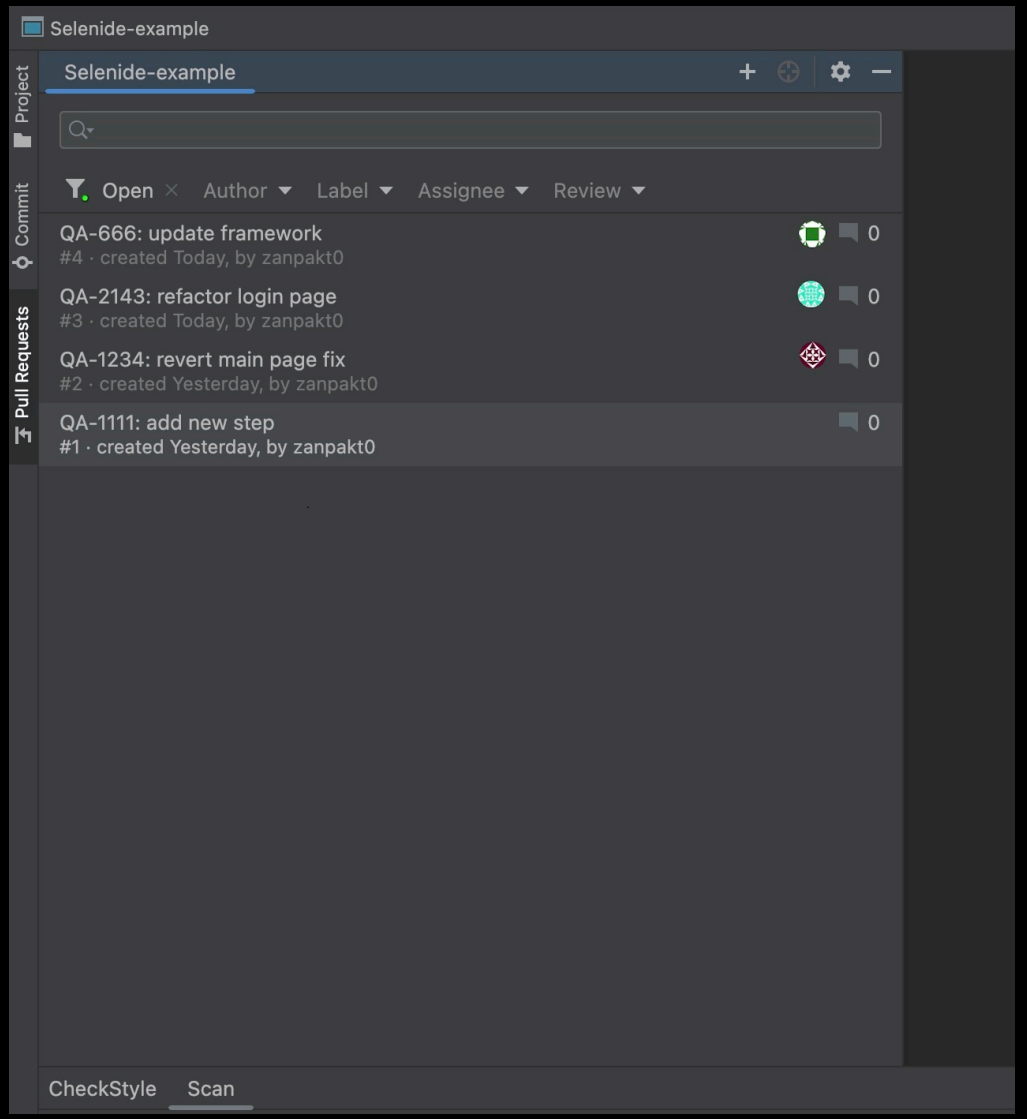
Проблема:
не всегда удобно проводить ревью в системе управления репозиториями

```
public class LoginFormTest extends BaseTest {
```

10 тысяч новых строк?
Может, стоило декомпозировать задачу?

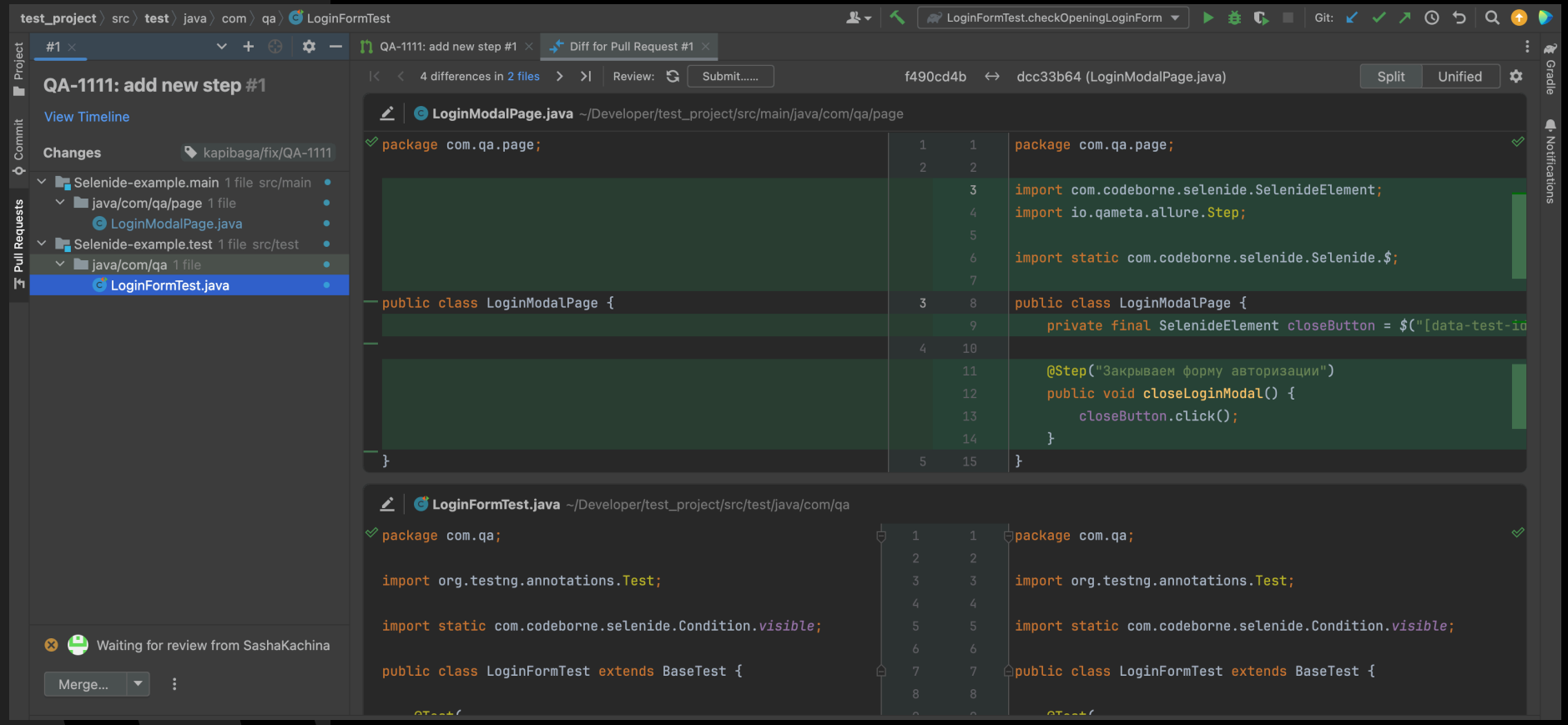
Ревью в среде разработки

Список мерж реквестов



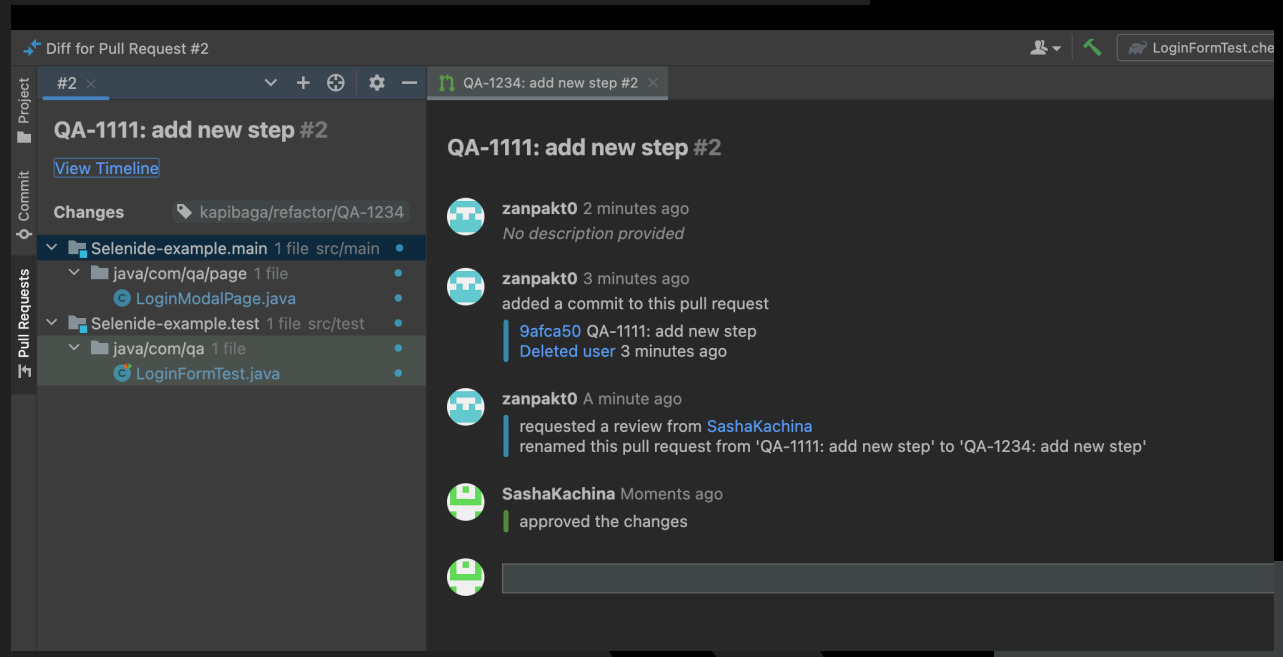
Ревью в среде разработки

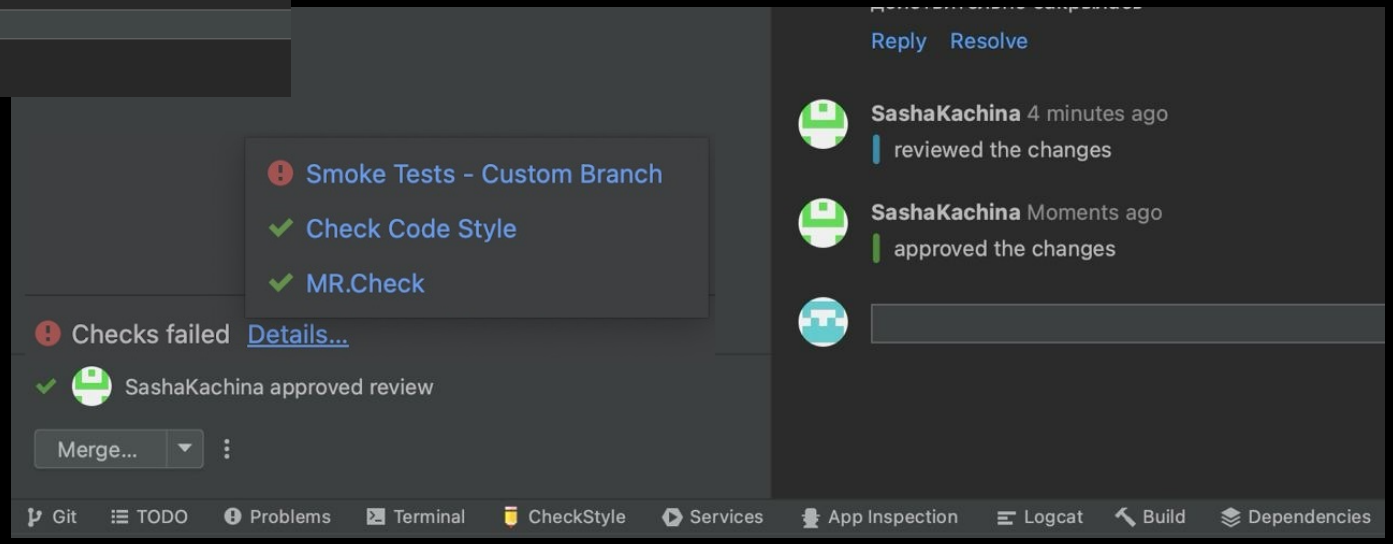
Изменённые файлы и блоки кода



Ревью в среде разработки

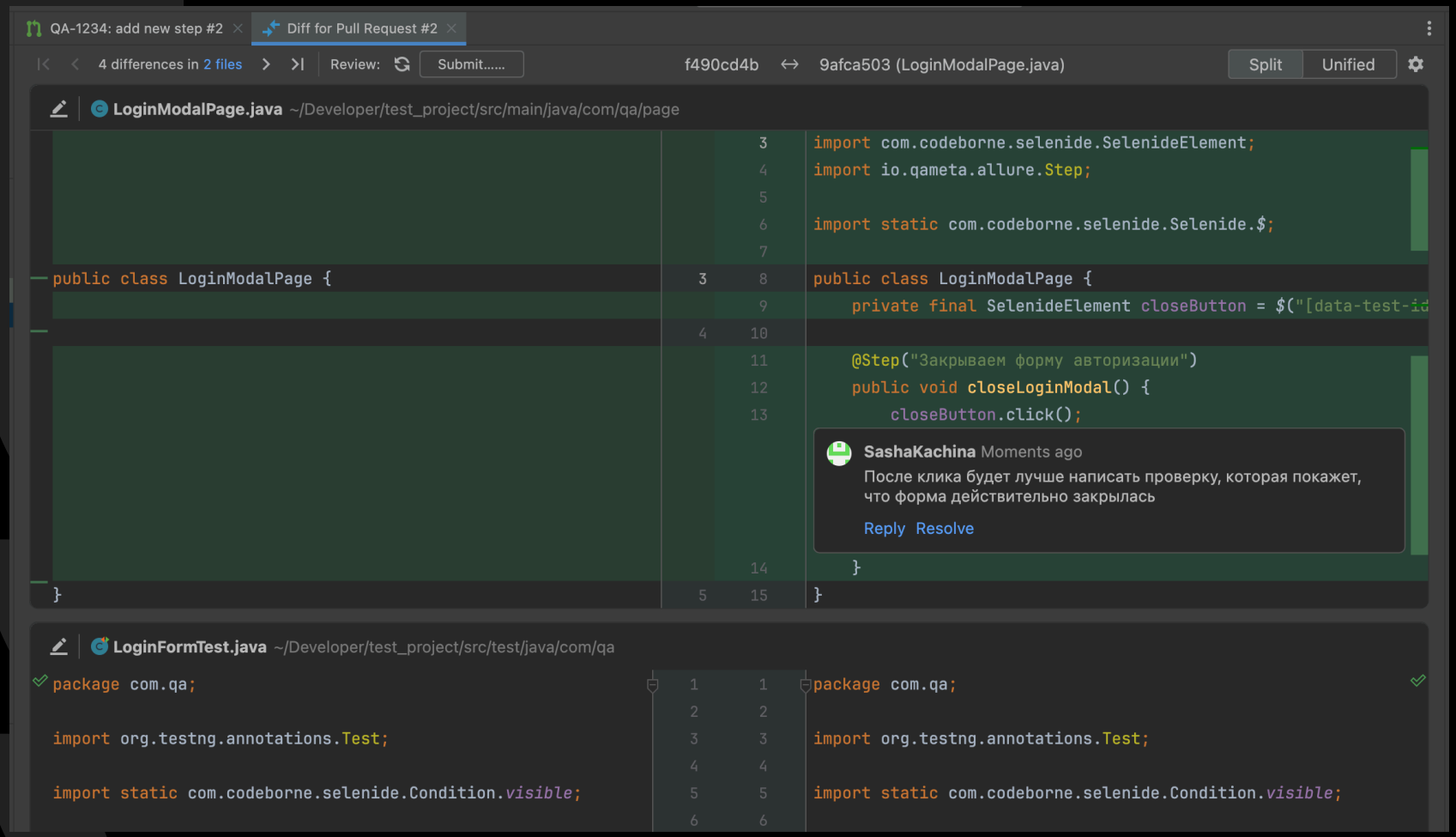
История мерж реквеста и ошибки пайплайна





Ревью в среде разработки

Комментирование



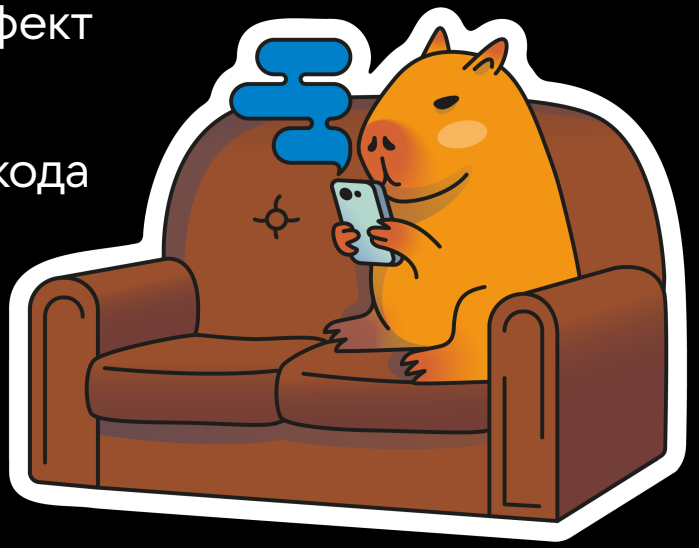
Функции комментирования

Саджесты:

1. использование специального синтаксиса для комментирования
2. принятие или отклонение предложения автором кода

Преимущества подхода:

1. ускорение процесса ревью
2. точность коммуникации
3. образовательный эффект
4. улучшение качества кода



Какие могут еще остаться ошибки на ревью?

1 Бездумная копипаста

```
@Test(
        description = "Пользователь добавляет сопутствующие
товары к товару в магазине»
)
```

Описание скопировано из соседнего теста, выглядит как не то, что должно быть)

```
  @Owner(KAPIBAGA)
  @Test(description = "Удаление товара из закладок на новой
витрине сообщества")
  @SetUps({"openExtendedStore"})
```

А для чего здесь включается расширенный магазин? Вроде бы дальше работа идет с пейджом базового магазина

Какие могут еще остаться ошибки на ревью?

2 Неинформативные комментарии для отчётов о падении или их отсутствие

```
// Assert
assertThat(marketItems.getItems()).as("Вернулось кол-во
товаров, отличное от ожидаемого")
```

Не совсем информативное сообщение
об ошибке, лучше написать, например,
что количество товаров должно быть равно нулю

```
// Assert
assertThat(responseId).isPositive();
```

Не хватает комментариев, которые будут
отображаться при падении тестов

Какие могут еще остаться ошибки на ревью?

3 Отсутствие проверки после совершения действия

```
elements.photoEditorButton
    .shouldBe(visible, TIMEOUT)
    .click();
```

Проблема этого степа в том, что если клик не сработает, то степ все равно будет зеленый, а ошибка возникнет в другом степе. Поэтому, если клик — последняя операция в степе, его нужно проверять

Какие могут еще остаться ошибки на ревью?

4 Оставление в коде сервисных инструментов

```
@Test(description = "Создание услуги", invocationCount = 50)
```

Нужно будет удалить invocationCount, чтобы тест не заддосил инфру (полагаю, это случайно попало в коммит)

Какие могут еще остаться ошибки на ревью?

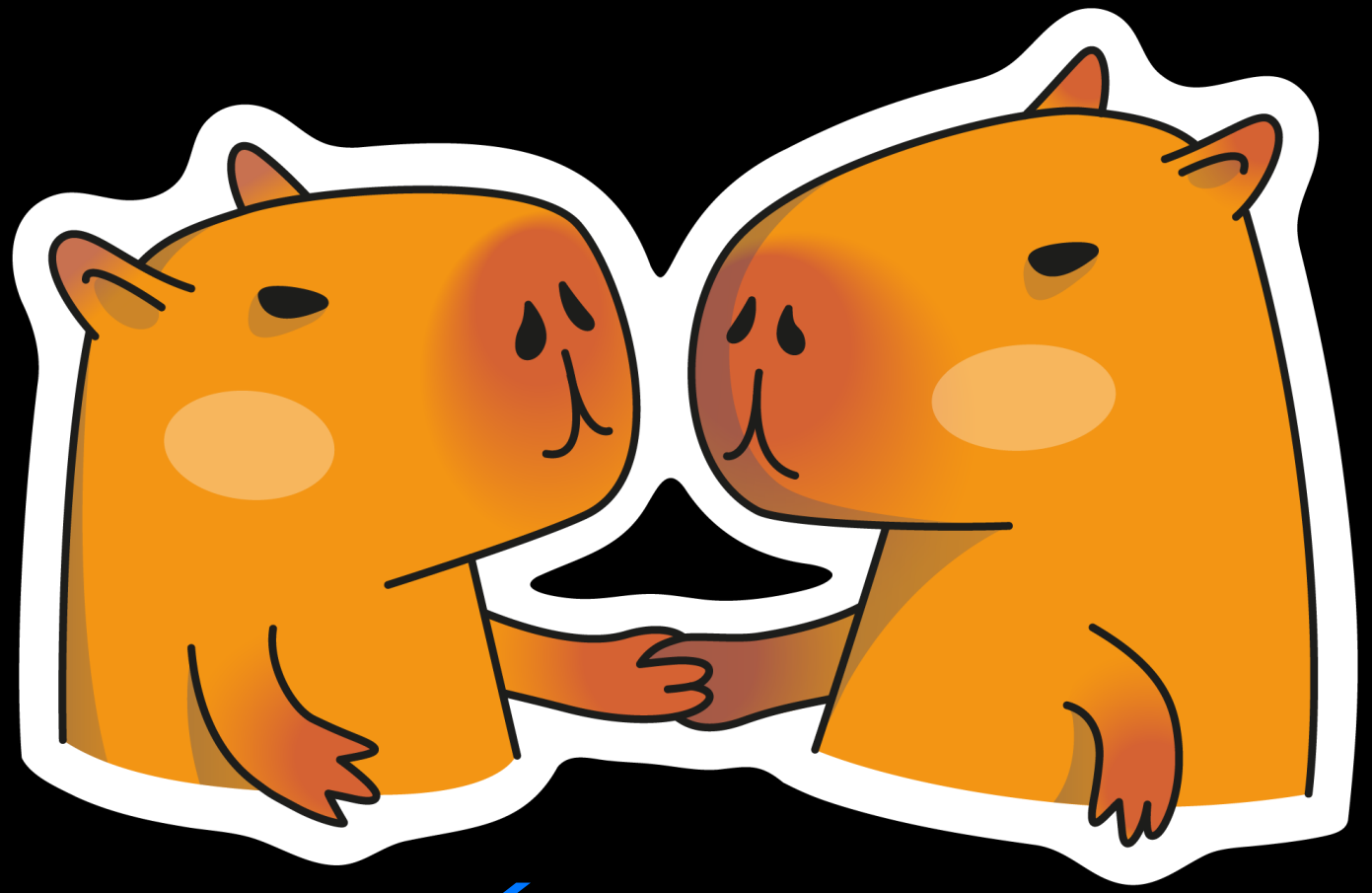
5 Отсутствие выноса кода в общий метод

```
// Assert
assertThat(getResponse.get(0).getId()).as("Поле id не должно
быть пустым»).isNotEmpty();

assertThat(getResponse.get(0).getName()).as("Поле name не
должно быть пустым").isNotEmpty();
```

Эти проверки выглядят одинаково, можно общий метод сделать в классе с ассертами

Какие могут еще остаться ошибки на ревью?



6 Братский апрув

Какие могут еще остаться ошибки на ревью?

Рекомендации

Составьте чеклист для написания автотеста:

1. Проверить аннотации
2. Проверить, как будет отображаться лог в каждом шаге, если тест упадёт
3. Проверить, остались ли в коде сервисные методы для дебага и т.д.

Как проводить код-ревью?

Рекомендации

- Будьте тактичны
- Фокусируйтесь
- Пишите комментарии, подкрепленные аргументами
- Пишите опциональные комментарии, указывая об этом
- Запрашивайте уточнения
- Поставьте себе ограничения по времени
- Используйте чек-лист для код-ревью

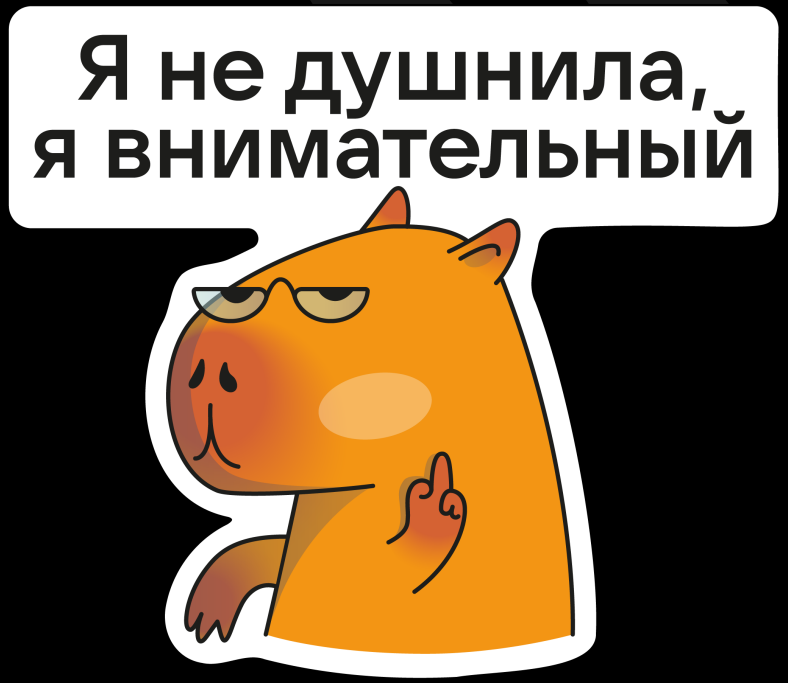
Как проводить код-ревью?

Рекомендации

- Не пишите неинформативные комментарии
- Воздержитесь от шуток и иронии
- Не используйте открытую агрессию
- Не уводите диалог в личные сообщения
- Следите за пассивной (скрытой) агрессией и токсичностью

Сделаем выводы

1. Ревью кода — это полезно
2. Использование возможностей среды разработки помогает избегать ошибок
3. Автоматизация проверки кода на CI и настройка уведомлений помогает ускорить фиксы типовых ошибок
4. Быть душнилой и токсиком — не круто



Контакты

Telegram стикеры
VKMarketQA

Александра Качина