Санкт-Петербургский
Государственный Университет

Математико-Механический факультет
Кафедра Системного Программирования

Божнюк Александр Сергеевич

Инструмент для сравнения Jupyter ноутбуков в интегрированной среде разработки DataSpell

Отчет по производственной практике

Консультант:
DataSpell Software Developer А.В. Вокин

Научный руководитель:
ст. преп. С.Ю. Сартасов

Санкт-Петербург
2022

Содержание

Введение **2**

1. Цели и задачи **3**

2. Обзор предметной области **4**
2.1. Проект Jupyter 4
2.1.1. Jupyter Notebooks 4
2.1.2. Особенности Jupyter Notebooks: Jupyter ядра и конвертация 4
2.2. Обзор алгоритмов сравнения 5
2.3. Обзор конкурентов 6
2.3.1. Найденные конкуренты 6
2.3.2. Библиотека nbdime 7
2.3.3. Visual Studio Code 9
2.3.4. Выводы из анализа конкурентов 10
2.4. Обзор платформы IntelliJ 11
2.4.1. Документ 11
2.4.2. Редактор 11
2.4.3. Точки расширения 11

3. Ход работы **12**
3.1. Основная идея 12
3.2. Первая попытка реализации подмены документа 13
3.3. Вторая попытка реализации подмены документа 14
3.4. Исправление проблем с интерфейсом 15
3.5. Сравнения ячеек с кодом Markdown 19
3.6. Сравнение выводов ячеек 21
3.6.1. Извлечение информации о выводах ячеек 21
3.6.2. Сравнение информации о выводах ячеек 22
3.6.3. Визуализация сравнения выводов ячеек 23
3.6.4. Итоговый алгоритм сравнения выводов ячеек 24
3.7. Инструмент слияния 25

4. Результаты **27**

5. Список литературы **28**

Введение

На текущий момент все более популярной становится такая технология, как Jupyter Notebooks [1]. Это среда разработки, которая позволяет работать с документами, содержащими не только исходный код, а также текст и различные компоненты визуализации (графики, картинки и так далее). Она позволяет создавать интерактивные документы, в которых можно производить запуск кода, находящегося в спецальных ячейках. В отдельных ячейках также может находиться код на языке разметки Markdown, что позволяет стилизовать текстовую информацию внутри документа. Вывод, который производит исходный код, также отображается в документе. При всем этом каждый Jupyter Notebook из себя представляет JSON документ. Основная цель этого проекта - поддержка интерактивного анализа данных и научных вычислений на всех языках программирования посредством разработки программного обеспечения с открытым исходным кодом.

Компания JetBrains [2] создала свою интегрированную среду разработки (IDE) для работы с Jupyter Notebooks и для Data Science в целом - DataSpell [3]. Данная IDE к функциональности ноутбуков Jupyter добавляет функциональность стандартных IDE от компании JetBrains: помощь при написании кода, интеграция с системами контроля версий, работа с терминалом, работа с базами данных, настройка среды разработки при помощи плагинов, большая интерактивность и многое другое.

Существует инструмент сравнения файлов (Diff Tool) [4]. Он используется для генерации разницы между файлами или их версиями (в случае работы с системами контроля версий). В JetBrains существует своя реализация инструмента сравнения, которая качественно выполняет возложенные на неё задачи и при этом не вызывает нареканий у пользователей. Однако, в случае с файлами Jupyter данный инструмент, вместо того, чтобы показывать разницу содержимого самих ноутбуков, показывает разницу для JSON файлов. Это делает инструмент в случае работы с ноутбуками крайне неудобным: довольно трудно найти нужные изменения, возникают проблемы с работой дополнительной функциональности инструмента разницы в IDE.

Поэтому требуется создать инструмент сравнения, который бы учитывал структуру ноутбуков Jupyter и в удобном виде отображал бы информацию об изменениях. Также нужно добиться корректной работы в случае интеграции проекта с системами контроля версий (например, Git), а для этого нужно разработать отдельный инструмент слияния (Merge Tool), отвечающий за эту работу. К тому же, существуют различные вопросы проектирования, которые требуется разрешить: сравнение ячеек с обработанным кодом на языке Markdown, сравнение выводов ячеек. В конце концов, потребуется аккуратная интеграция новой созданной функциональности в DataSpell.

1. Цели и задачи

Целью данной работы является создание инструмента сравнения для ноутбуков Jupyter в интегрированной среде разработки DataSpell. Для достижения цели были поставлены следующие задачи:

1. Провести обзор существующих подходов к алгоритмам сравнения.
2. Провести обзор конкурентов, имеющих свою реализацию инструмента сравнения файлов.
3. Провести обзор платформы IntelliJ.
4. Реализовать инструмент сравнения файлов для Jupyter ноутбуков и интегрировать его в DataSpell. При этом разобраться с такими сопутствующими задачами, как сравнение выводов ячеек и сравнение ячеек с кодом Markdown.
5. Реализовать инструмент слияния для интеграции с системами контроля версий.

2. Обзор предметной области

2.1. Проект Jupyter

Проект Jupyter - это проект, созданный с целью облегчения разработки программного обеспечения с открытым исходным кодом и интерактивных вычислений на различных языках программирования. Был выделен из IPython в 2014 году Фернандо Пересом [5] и Брайаном Грейнджером [6]. Проект назван в честь трех основных языков программирования, поддерживаемых им: Julia, Python и R. Jupyter также ставит перед собой цель поддержать Data Science и научное программирование на всех языках программирования.

2.1.1. Jupyter Notebooks

Jupyter Notebooks - это интерактивная вычислительная среда для создания в браузере специальных документов - Jupyter ноутбуков.

Ноутбук Jupyter - это документ, который содержит упорядоченный список ячеек ввода/вывода. В каждой ячейке ввода может содержаться код или текстовая информация (с использованием языка разметки Markdown). В каждой ячейке вывода может находиться текстовая информация, графическая информация, картинки или даже интерактивный вывод. Каждый ноутбук имеет расширение ".ipynb" , однако внутри он представляет из себя JSON документ.

2.1.2. Особенности Jupyter Notebooks: Jupyter ядра и конвертация

Как было сказано ранее, Jupyter Notebook способен работать с разными языками программирования. Для этого используются ядра Jupyter (Jupyter Kernels). Ядро Jupyter - это программа, которая отвечает за обработку различных видов запросов (выполнение кода, дополнение кода) и взаимодействует с другими компонентами Jupyter. Ядра не обязаны быть подключены только к одному документу: они могут работать с несколькими клиентами сразу. По умолчанию Jupyter Notebook поставляется с одним ядром - IPython. Обычно ядра позволяют работать только с одним языком программирования. Сейчас существует огромное множество Jupyter ядер для многих языков программирования [7].

Ноутбук Jupyter может быть конвертирован в HTML, LaTeX, PDF, Markdown, Python, ReStructuredText форматы через интерфейс Jupyter или через командную строку.

2.2. Обзор алгоритмов сравнения

Ниже приведены алгоритмы, наиболее используемые для выполнения такой задачи, как сравнение файлов.

Первая версия алгоритма сравнения, который мог подойти для сравнения файлов, была описана в 1976 году в статье "An Algorithm for Differential File Comparison" [8]. Здесь описывается подход, который основан на известной задаче "Поиск наибольшей общей подпоследовательности (Longest Common Subsequence problem, LCS). Основная суть заключается в том, чтобы найти те строки, которые не изменились между файлами. Описывается модифицированное решение задачи LCS, благодаря которой достигается большая эффективность. Для улучшения производительности также описываются методы хеширования, сортировки в классах эквивалентности, слияния через бинарный поиск и динамическое перераспределение памяти.

В работе "An O(ND) Difference Algorithm and Its Variations" [9] описывается улучшенная версия алгоритма сравнения. Здесь приводится алгоритм, основанный на задаче нахождения кратчайшего пути в графе редактирования (edit graph). В статье показывается эффективность данного метода по сравнению с подходами, описанными выше.

Предыдущие два алгоритма и их различные модификации используются в платформе IntelliJ для инструмента сравнения.

2.3. Обзор конкурентов

При планировании решения задачи стали появляться вопросы архитектурного характера. Рассмотрим эти вопросы:

1. Сравнение ячеек с кодом на языке разметки Markdown. Проблема заключается в том, что при отображении разметки происходит её обработка с целью изображения, например, математических выражений в удобном для глаза виде. Однако довольно трудно спроектировать сравнение уже обработанного текста. Требуется решение, которое было бы не сильно затратным, но при этом выполняло задачу - вывод на экран информации об изменении в коде Markdown.

2. Сравнение выводов ячеек. Среди выводов ячеек может быть текст, диаграммы, картинки, и даже динамические графики (такие графики можно генерировать с помощью библиотеки matplotlib). Возникает вопрос в методе сравнения таких выводов и отображения соответствующей информации о факте произошедших изменений.

3. Какие дополнительные возможности потенциально еще можно реализовать?

Для ответа на эти вопросы было принято решение провести анализ конкурентов, которые в той или иной степени смогли реализовать инструмент сравнения.

2.3.1. Найденные конкуренты

В результате поиска были найдены следующие конкуренты:

- **Библиотека nbdime** [10]. Это самая известная библиотека с открытым исходным кодом, которая позволяет сравнивать Jupyter ноутбуки. Её также можно использовать при работе с системами контроля версий.

- **Visual Studio Code (VS Code)** [11]. Это редактор исходного кода, разработанный компанией Microsoft. Используется для кроссплатформенной разработки. Включает в себя отладчик, инструменты для работы с системой контроля версий Git, IntelliSense (технология автодополнения кода от Microsoft), средства для рефакторинга. VS Code поддерживает Jupyter ноутбуки [12], тем самым являясь главным конкурентом DataSpell. В нем также реализован инструмент для сравнения Jupyter ноутбуков.

- **DagsHub** [13]. Это платформа для совместной работы в области науки о данных и машинного обучения с открытым исходным кодом. Она позволяет делиться с другими проектами, переиспользовать другие, собирать проекты. По принципу работы схожа с

GitHub [14]. Естественно, в DagsHub есть интеграция с системами контроля версий, и появляется потребность сравнивать Jupyter ноутбуки. Поэтому там тоже был реализован свой инструмент для этого.

- **Curvenote** [15]. Это платформа, в которой также есть интеграция с Jupyter ноутбуками. Она позволяет создавать технические статьи и отчеты, которые могут включать в себя картинки, тексты и прочие важные элементы. Также позоляет сравнивать Jupyter ноутбуки.

- **ReviewNB for GitHub** [16]. Это проект, целью которого является обеспечить удобство проводить проверку (code review) Jupyter ноутбуков на GitHub. Также включает в себя инструмент для сравнения ноутбуков.

В результате анализа этих конкурентов было замечено, что DagsHub, Curvenote и ReviewNB не привносят новых возможностей и отличий по сравнению с nbdime и VS Code. Поэтому далее в тексте более подробно пойдет речь именно об этих двух конкурентах.

2.3.2. Библиотека nbdime

Библиотека nbdime имеет позволяет строить разницу между в файлами в консоли и в браузере. В первую очередь рассматривается браузерная версия, так как именно в ней можно рассмотреть необходимые графические решения.

Если говорить о сравнении ячеек с текстом разметки, то здесь работает довольно простое и элегантное решение. Обработка разметки просто отключается, и показывается разница для исходного текста. Это можно пронаблюдать на рисунке 1.

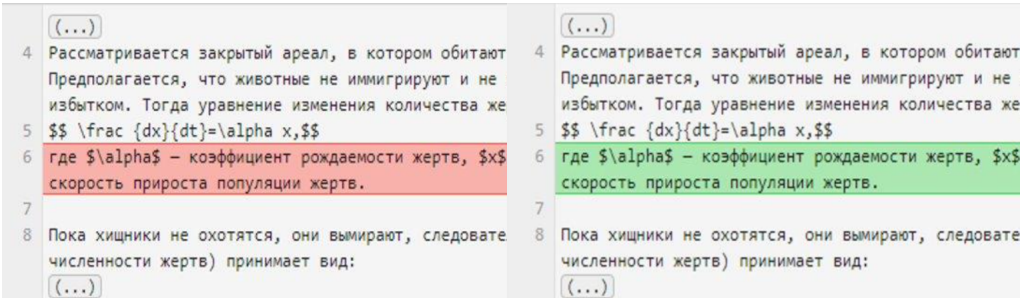

Рис. 1: Сравнение ячеек Markdown в nbdime

Если говорить о сравнении выводов ячеек, то в nbdime сравнение показывается по принципу было/стало. Это значит, что показывается состояние вывода до появления каких-либо изменений и после. При этом не дается никакой информации о природе самих изменений. Такой принцип работает для всех типов выводов ячеек. Это можно увидеть на рисунках 2 и 3.

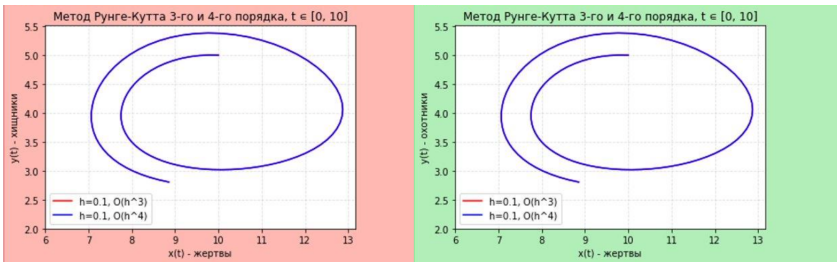

Рис. 2: Сравнение графиков в nbdime

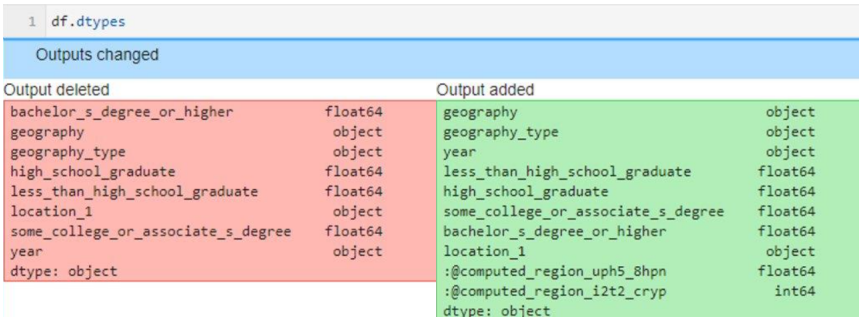

Рис. 3: Сравнение текстов в nbdime

Если говорить о дополнительных возможностях, которые предоставляет nbdime, то можно заметить следующее: если в ячейке не произошло никаких изменений, то её содержимое не показывается, благодаря чему результат сравнения не нагружает пользователя лишней информацией. К тому же, присутствует возможность раскрыть или свернуть неизменившиеся ячейки. Это можно увидеть на рисунке 4.

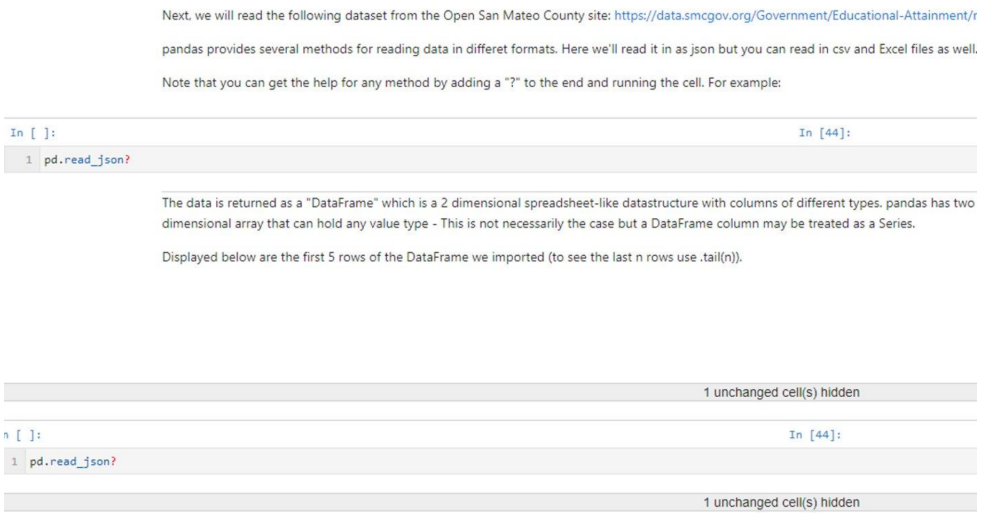

Рис. 4: Свертка неизменившихся ячеек в nbdime

2.3.3. Visual Studio Code

Visual Studio Code имеет поддержку Jupyter ноутбуков, а также инструмент для их сравнения, который также графически показывает изменения.

Если говорить о сравнении ячеек с текстом разметки, то, как и в nbdime, просто выключается обработка содержимого и показывается разница для исходного кода. Это можно пронаблюдать на рисунке 5.

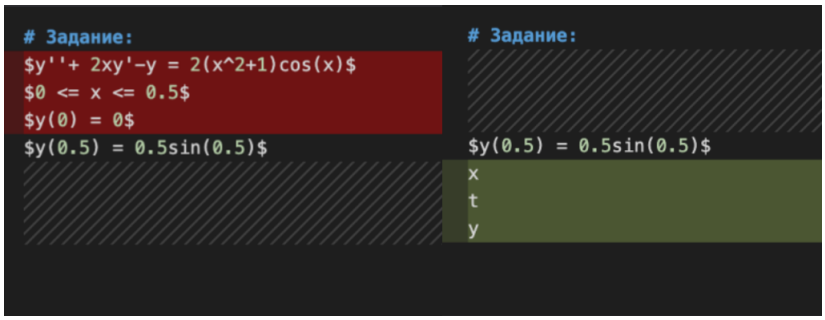

Рис. 5: Сравнение ячеек Markdown в VS Code

Если говорить о сравнении выводов ячеек, то здесь все работает также, как и у предыдущего конкурента: не показывается информации о природе изменений, работает принцип было/стало. Это можно видеть на рисунках 6 и 7.

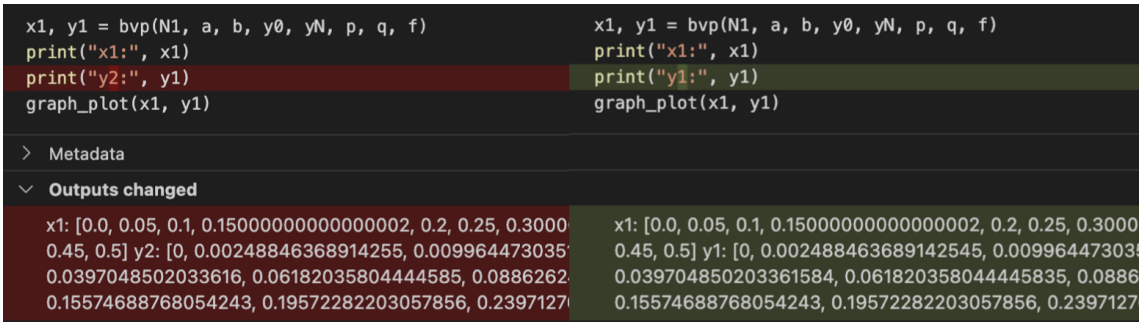

Рис. 6: Сравнение текстов в VS Code

Если говорить о дополнительных возможностях, то Visual Studio Code позволяет увидеть изменения в метаданных (metadata [17]) ноутбука. Это бывает важно, потому что метаданные способны влиять на поведение ноутбука. Эту возможность можно пронаблюдать на рисунке 8.

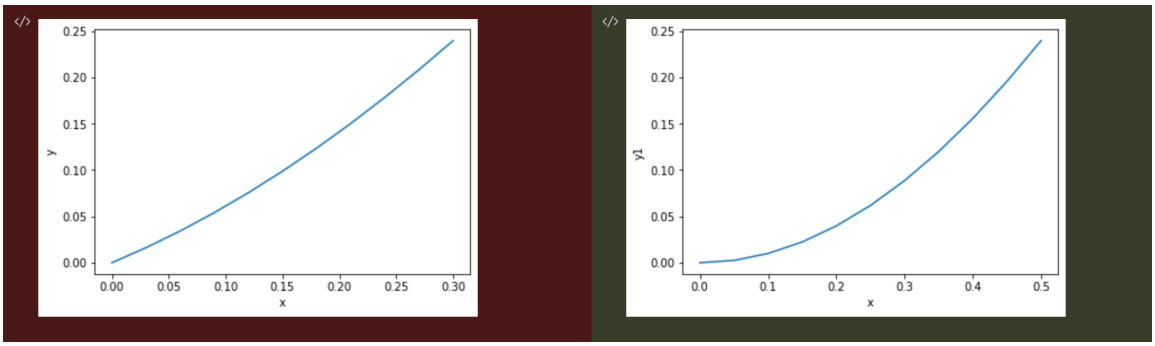

Рис. 7: Сравнение графиков в VS Code

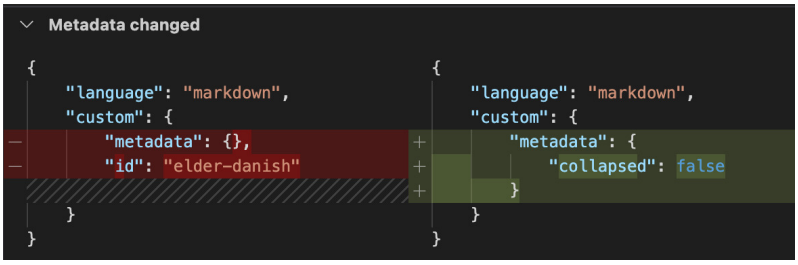

Рис. 8: Сравнение метаданных в VS Code

2.3.4. Выводы из анализа конкурентов

В итоге из анализа конкурентов можно сделать следующие выводы:

- Сравнение ячеек с кодом Markdown производится за счет отключения обработки теста и последующего сравнения содержимого.
- Сравнение выводов ячеек производится по принципу было/стало. Отображается вывод ячейки до изменений и после изменений без указания природы самих изменений.
- Среди дополнительных возможностей можно выделить следующее: сравнение метаданных и свертка ячеек, в которых не произошло изменений.

2.4. Обзор платформы IntelliJ

Платформа IntelliJ [18] является важной составляющей для построения IDE. Это проект с открытым исходным кодом. Он предоставляет необходимую инфраструктуру для поддержания возможностей различных языков программирования, а также возможностей IDE. С помощью платформы IntelliJ можно создавать плагины для среды разработки, для чего довольно часто её и используют. На базе этой платформы функционирует все IDE компании JetBrains, в том числе и DataSpell. Данный раздел ставит перед собой цель описать составляющие платформы IntelliJ, которые будут необходимы для дальнейшего хода работы.

2.4.1. Документ

Документ (Document) [19] - это одна из важнейших сущностей, которыми оперирует платформа IntelliJ. Он содержит в себе последовательность символов Unicode, которую можно считать и модифицировать. Документ позволяет работать с содержимым файла как с обычным текстом, что можно удобно использовать в работе.

2.4.2. Редактор

Редактор (Editor) [20] - также одна из базовых сущностей в платформе IntelliJ. Он позволяет настраивать и использовать различную функциональность, связанную с IDE. Например, редактор поддерживает систему кареток, декораторов (Inlays), различные реакции на действия пользователя. Их можно изменять под нужды конкретной среды разработки.

2.4.3. Точки расширения

Точка расширения (Extension Point) [21] - это сущность платформы IntelliJ, которая позволяет одним плагинам расширять другие плагины. В данной работе рассматривается исключительно интерфейсная точка расширения (Interface extension point), которая позволяет расширять плагины посредством нового исходного кода. Для работы с ними требуется создать интерфейс и зарегистрировать его в файле `plugin.xml`.

Далее при работе с точками расширения используются расширения (Extensions) [22]. Именно они и увеличивают функциональность плагина. Для работы с расширением требуется реализовать интерфейс точки расширения и зарегистрировать класс в `plugin.xml`. Далее через API платформы можно получить список расширений для конкретной точки и использовать их.

3. Ход работы

3.1. Основная идея

В рамках системы ноутбуки Jupyter представляются при помощи двух документов:

- JSON документ. Этот документ содержит в себе все содержимое файла ноутбука.
- Документ ноутбука. Этот документ содержит информацию о содержании ячеек в упрощенном виде, не в JSON формате. Так, ячейки в данном документе разделяются при помощи строки `#%%` с суффиксом `md` в случае ячейки с кодом Markdown, между разделителями содержится исходный код ячейки. Его удобно использовать для визуализации ноутбуков, так как он не содержит лишней информации.

Важным элементом реализации Diff Tool в платформе IntelliJ является такая сущность, как `DiffContent`. Она представляет из себя данные, которые можно будет сравнить с другими данным. Интерфейс `DiffContent` реализуется разными классами, вроде `DocumentContent`, `FileContent` и многие другие. Эти классы отличаются способом представления данных внутри себя (в виде документа, файла и др.). В обычном режиме Diff Tool сравнивает содержимое `DocumentContent`, прибегая к сравнению содержимого документов.

Исходя из сказанного выше, становится понятной основная проблема. Во время сравнения Diff Tool вместо документов ноутбуков получает JSON документы. В конечном итоге не срабатывают алгоритмы визуализации ноутбуков, и пользователь видит сравнение JSON представлений. Идея заключается в подмене JSON документа на документ ноутбука, тем самым заставив работать алгоритмы визуализации. Для этого потребуется подменить соответствующие `DocumentContent`.

3.2. Первая попытка реализации подмены документа

Для подмены была создана точка расширения `DiffContentSubstitutor`. Интерфейс этой точки расширения имел следующие методы:

- `@NotNull Document substituteDocument(@NotNull String content, @Nullable FileType fileType)`: отвечает за подмену документа. Ему требуется тип файла и старое содержимое документа.
- `boolean canSubstitute(@Nullable FileType fileType)`: требуется для проверки того, нужно ли использовать метод подмены документа или нет. Также этот метод должен предотвращать от использования не нужных расширений.

`DiffContentSubstitutor` находится в модуле `diff.impl`. Расширением выступает класс `JupyterDiffContentSubstitutor`. Данный класс находится в модуле `jupyter-diff` и выступает в качестве реализации интерфейса, описанного выше.

Далее требовалось использовать точку расширения. В модуле `diff.impl` есть класс `DiffContentFactory`, который отвечает за создание `DocumentContent`. Необходимо было встроить точку расширения в методы, которые отвечают за генерацию `DocumentContent` для сравнения файлов.

Данный подход имеет достаточно большое количество проблем:

1. Вмешательство во внутренний код платформы, что могло привести к проблемам внутри неё. Более того, здесь можно наблюдать нарушение принципа открытости и закрытости SOLID.
2. Проблемы с дальнейшим расширением. Так, например для исправления сравнения ячеек с текстом Markdown, пришлось бы создавать большое количество точек расширения и встраивать их в код платформы.
3. Проблемы с работой встраиваемых точек расширения. При работе с точками расширения перебираются все возможные расширения, относящиеся к данной точке. Для того, чтобы срабатывало нужное расширение, существует метод `canSubstitute(...)`. Однако может произойти следующее: другой человек создает свое расширение для точки `DiffContentSubstitutor`, и определяет метод `canSubstitute(...)` неверно. Это может привести к тому, что будут срабатывать неверные расширения и к неверному поведению кода платформы.

В связи с проблемами, описанными выше, было принято решение изменить подход к подмене документа.

3.3. Вторая попытка реализации подмены документа

Второй подход заключается в более правильном использовании средств платформы IntelliJ для реализации требуемого. Существует точка расширения `DiffTool`, которая отвечает за сам инструмент сравнения. Перед описанием реализации стоит рассмотреть некоторые сущности, которые требуются для работы `DiffTool`:

- `DiffContext`: контекст, который инкапсулирует в себе необходимую информацию о текущем проекте и пользовательских настройках.
- `DiffRequest`: запрос, который содержит в себе информацию, которую требуется сравнить. В качестве такой информации может, например, выступать список `DiffContent`.
- `DiffViewer`: компонент визуализации результатов сравнения.

Интерфейс `DiffTool` имеет следующие методы:

- `@NotNull DiffViewer createComponent(@NotNull DiffContext context, @NotNull DiffRequest request)`: отвечает за создание `DiffViewer` на основе контекста и запроса.
- `boolean canShow(@NotNull DiffContext context, @NotNull DiffRequest request)`: отвечает за проверку того, можно ли на основе контекста и запроса отобразить результат сравнения. Должен обязательно вызываться до вызова метода `createComponent`.

В конечном итоге имеем следующее: `DiffTool` реагирует на пришедший к нему запрос `DiffRequest`, и на основе контекста `DiffContext` сначала определяет, можно ли этот запрос отобразить. Если отобразить можно, то `DiffTool` это делает через `DiffViewer`.

Для того, чтобы осуществить требуемую подмену, нужно подменить `DiffRequest`, и уже на основе подмененного запроса строить `DiffViewer`. Для этого было создано расширение `JupyterDiffTool`.

В реализации метода `canShow` проверяется, чтобы в запросе содержался список из `DocumentContent`, и в этом списке было от одного до трех элементов. Более того, проверяются типы файлов, связанных с элементами данного списка, что позволит удостовериться, что обрабатываются файлы ноутбуков Jupyter.

В реализации метода `createComponent` производится подмена запроса (создается новый запрос с подмененным документом), а на основе контекста и подмененного запроса создаются соответствующие компоненты визуализации.

Данный подход позволил избавиться от проблем, проявившихся ранее.

3.4. Исправление проблем с интерфейсом

Подмена документов привела к тому, что при использовании инструмента сравнения Jupyter пользователь видит сравнение уже отрисованных ноутбуков. Однако, как видно на рисунке 9, требовалось исправить большое количество проблем с интерфейсом.

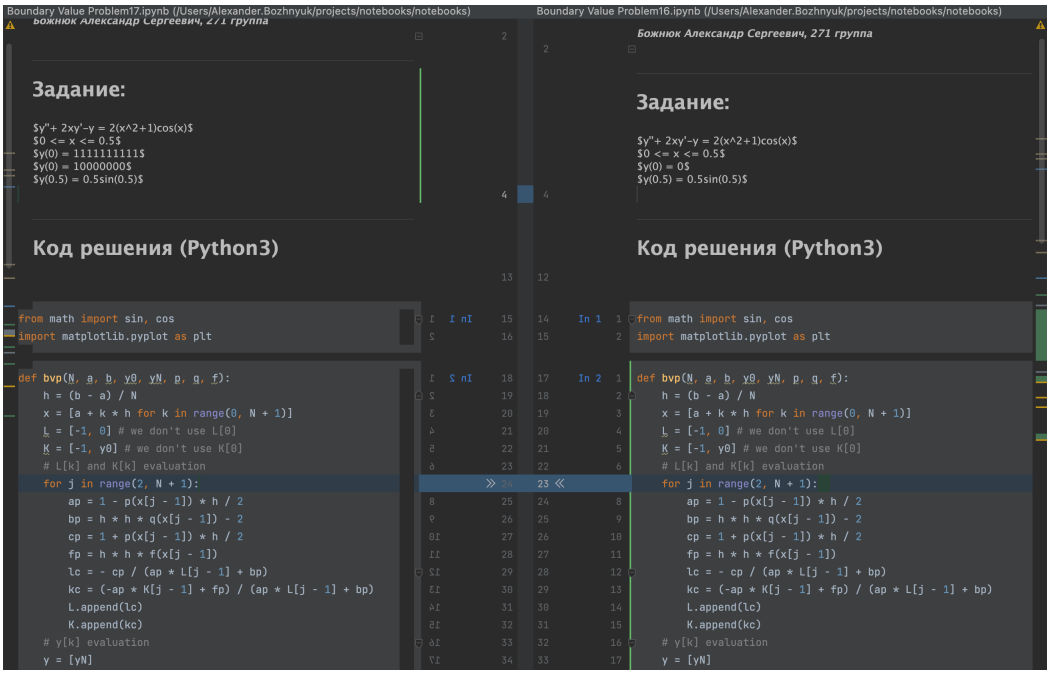

Рис. 9: Первая версия инструмента сравнения

Среди проблем можно выделить следующее:

- Артефакты при прорисовке ячеек, как видно на левом ноутбуке справа и слева.
- Слишком широкий гаттер (компонент, на котором располагаются номера линий). Также на нем отображаются лишние номера, которые никак не связаны с ячейками. Более того, следует отключить отображение количества запусков ячейки, так как здесь это не несет никакого смысла.
- Расположение якорей свертки кода неверное. Нажатие на якорь приводит к поломке интерфейса.

Первый шаг, который был предпринят по исправлению интерфейса - создание своих `DiffViewer`. Это сразу позволило более гибко настраивать не только интерфейс, но и другие составляющие визуализации. Для отображения результатов сравнения для обычных файлов (содержимое которых заворачивается в `DocumentContent`) используются классы `SimpleDiffViewer`, `SimpleOneSideDiffViewer`,

SimpleThreeSideDiffViewer. Они отвечают за отображение двухстороннего, одностороннего и трехстороннего сравнения соответственно. Благодаря отдельным инструментам отрисовки ноутбуков, которые работают независимо от отображения результатов сравнения в модуле diff.impl, отображения сравнения файлов Jupyter отличается от работы с обычными файлами только лишь некоторыми изменениями в интерфейсе. Поэтому было решено создать JupyterDiffViewer, JupyterOneSideDiffViewer и JupyterThreeSideDiffViewer при помощи раширения трех классов, описываемых выше.

Для того, чтобы внести свои корректировки в отображение результатов сравнения, у новых классов потребовалось переопределить метод onAfterRediff(). Он отвечает за выполнение действий, которые будут производиться после того, как отработали алгоритмы вычисления разницы между файлами. Также гарантируется, что данный метод работает в рамках Event Dispatch Thread. Поэтому имеет смысл операции по работе к компонентами визуализации выполнять именно в нем.

Второй шаг заключался во внесении определенных изменений в отрисовку ноутбуков. За визуализацию ноутбуков отвечают модули notebooks.visualization и jupyter, именно внутри них происходит отрисовка номеров линий на гаттере возле ячеек и чисел, отвечающих за количество запусков ячейки. Также внутри них отрисовываются ячейки и происходит другая настройка визуализации. Однако эти модули занимаются визуализацией в предположении, что ноутбук не отображается в инструменте сравнения и редактор, например, не является отзеркаленным (находится слева).

Платформа позволяет определить, когда редактор находится в режиме сравнения и является отзеркаленным. Для первого у интерфейса Editor есть метод getEditorKind(), который возвращает объект перечисления EditorKind. В данном перечисления есть элемент DIFF. Если метод getEditorKind() возвращает EditorKind.DIFF, то редактор, относящийся к ноутбуку, находится в режиме сравнения. Для второго у класса EditorImpl, который реализует интерфейс Editor, существует метод isMirrored().

Проделав оба шага, удалось добиться следующих результатов.

Во-первых, были исправлены номера линий на гаттере. Проблема в том, что внутри модуля notebooks.visualisation была сделана своя визуализация номеров линий у ячеек, однако она никак не учитывала отзеркаливание, и у ноутбука слева эти номера отображались перевернутыми. Для верного отображения модуль diff.impl сам занимается установкой номеров посредством LineNumberConverter. Это интерфейс, который имеет два метода:

- Integer convert(@NotNull Editor editor, int lineNumber): по редактору и номеру строки в документе определяет, какой номер по счету будет иметь строка из документа.

- `Integer getMaxLineNumber(@NotNull Editor editor)`: максимальное число, которое может быть среди номеров строк. Оно нужно для определения ширины области, который будут заняты номерами строк, что отразится на ширине гаттера.

Однако установленный `LineNumberConverter` по умолчанию никак не учитывает структуру документа, связанного с ноутбуком Jupyter. В итоге было проделано следующее: в `notebooks.visualization` была отключена отрисовка номеров линий в случае, если редактор используется инструментом сравнения. Однако был установлен свой `LineNumberConverter`. В нем учитывались начала и концы ячеек, а также тип их содержимого (у ячеек с кодом на языке Markdown рисовать номера строк не нужно). Установка конвертера произведена в методе `onAfterRediff()`.

Во-вторых, была произведена настройка гаттера. В `onAfterRefiff()` поменялась его ширина, а также установлен задний фон. Более того, удалось якоря, связанные со сверткой кода. Также через внесение корректировок в модуль `notebooks.visualization` была отключена в режиме сравнения прорисовка индекаторов, показывающие количество запусков ячейки.

В-третьих, исправлены артефакты в отображении ячеек. Дело в том, что ячейки отображались при помощи декораторов (inlays). Это специальные визуальные элементы, которые отображаются в редакторе и связаны с определенной позицией в соответствующем документе. Отрисовка декораторов для ячеек производилась в модуле `jupyter` в определенных позициях и никак не учитывала то, что редактор может быть отзеркален. Корректировки позиций отрисовки для отзеркаленного редактора смогли исправить ситуацию.

В итоге, на рисунке 10 можно увидеть новую версию сравнения ноутбуков:

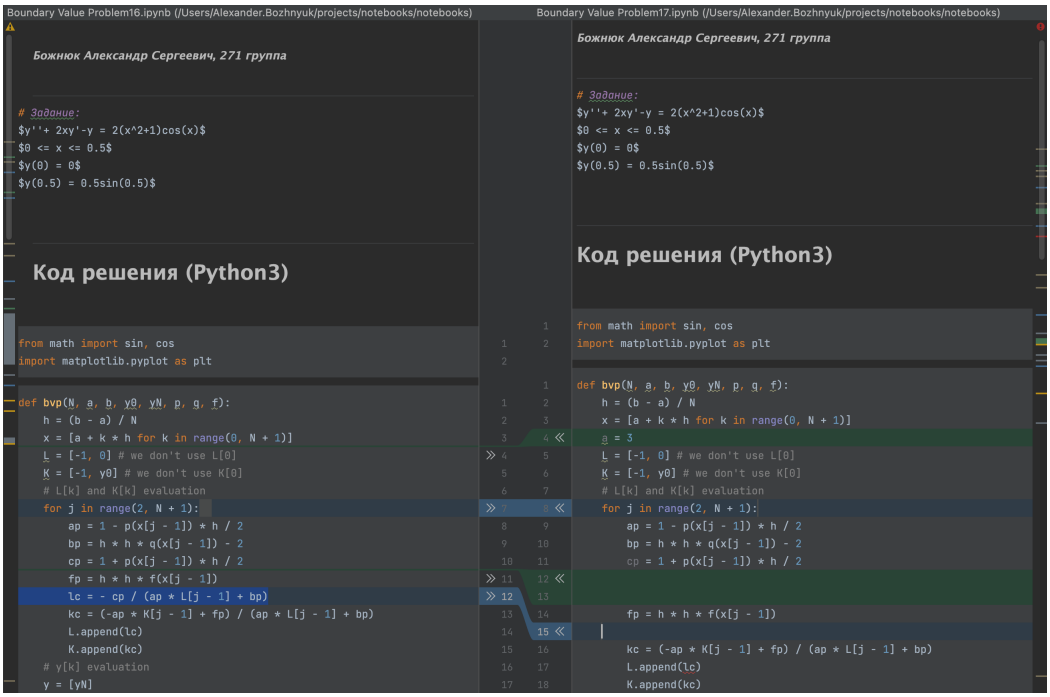

Рис. 10: Новая версия инструмента сравнения

3.5. Сравнения ячеек с кодом Markdown

На рисунке 9 можно заметить, что у двух ноутбуков есть отличия в ячейке с кодом на языке Markdown. Из-за рендеринга разница между содержимым ячеек никак не визуализируется. Из обзора конкурентов стало ясно, что лучший способ исправить ситуацию - остановить рендеринг ячеек, что позволит модулю `diff.impl` отобразить все необходимое.

Для того, чтобы отключить рендеринг необходимых ячеек требовалось определить, в каких ячейках произошли изменения. В модуле `diff.impl` за вычисление изменений, их хранение и обработку отвечают компоненты визуализации. У `SimpleDiffViewer` и прочих членов иерархии визуализаторов есть метод `getChanges()`, который позволяет получить список изменений. Каждое изменение характеризуется своим началом и концом для документа с левой и с правой стороны (также и для документа посередине в случае трехстороннего сравнения). Благодаря этой информации возможно понять, в каких ячейках нужно отключать рендеринг.

Модуль `notebooks.visualization` позволяет позволяет по номеру строки определить, в какой ячейке она находится. С каждой ячейкой ассоциирован интервал, который показывает, какие номера строк из ноутбучного документа занимает ячейка. Модуль визуализации позволяет получить такой интервал. Также модуль визуализации позволяет отключить обработку ячейки с кодом Markdown по интервалу.

В итоге был использован следующий подход. Из компонента визуализации была получена информация об измененных строках, а на их основе были найдены соответствующие интервалы. Далее через модуль визуализации был отключен рендеринг необходимых ячеек.

Был создан класс `JupyterMarkdownRenderDisabler`. Внутри него есть метод `disableRender(DiffViewer viewer)`, который в зависимости от переданного ему компонента визуализации отключает рендеринг в одностороннем, двухстороннем или трехстороннем сравнении.

Объект класса `JupyterMarkdownRenderDisabler` создается только один раз, поэтому было решено спроектировать его при помощи паттерна "Одиночка". Это решение в данном случае не дает каких-либо архитектурных преимуществ или недостатков, однако улучшает читаемость кода. В итоге, `DiffViewer` делает вызов `disableRender(this)`. Это вызов производится в методе `onAfterRediff()`.

На рисунке 11 можно наблюдать, как теперь работает сравнение ячеек с кодом на языке Markdown:

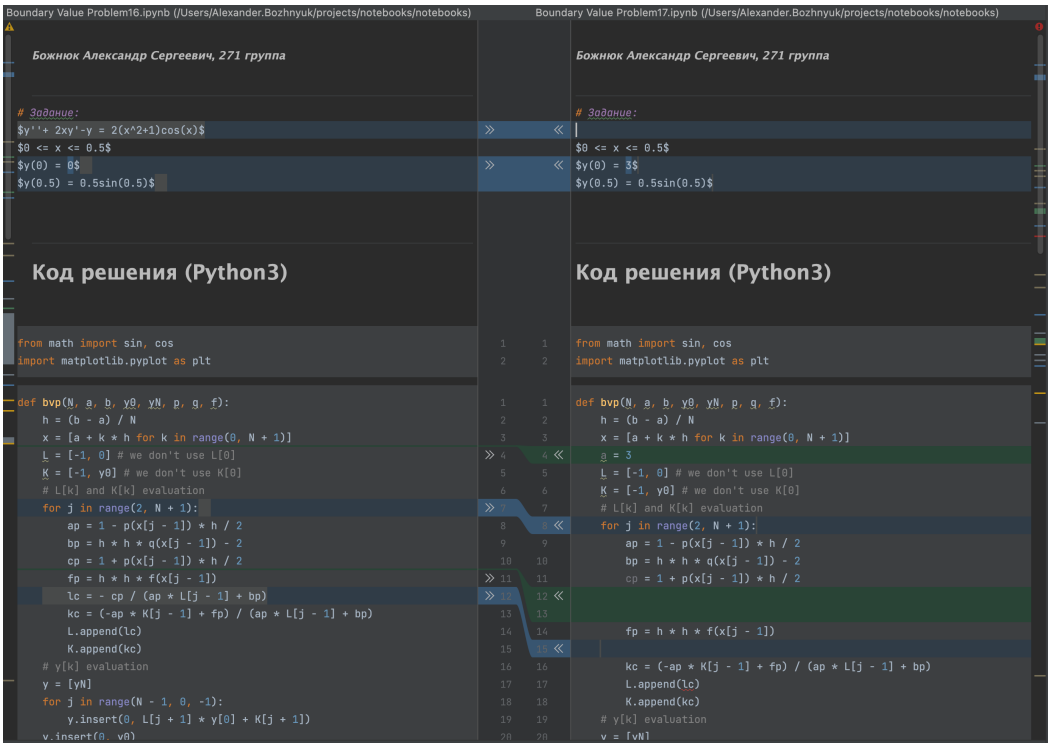

Рис. 11: Сравнение ячеек с кодом на языке Markdown

3.6. Сравнение выводов ячеек

Одной из наиболее важных составляющих в работе инструмента сравнения ноутбуков Jupyter является сравнение выводов ячеек. Вывод ячейки содержит результат вычислений, произведенных в этой ячейке. Это может быть текст, картинка, диаграмма и так далее. В рамках разбора конкурентов было решено, что лучше всего сравнивать выводы ячеек при помощи принципа было/стало: если изменение произошло, показывать полностью вывод до изменения и после, при этом выделять соответствующими цветами.

Сами выводы ячеек уже отображаются в ноутбуках и в режиме сравнения: за их отображение отвечает модуль визуализации. Однако в случае обнаружения изменения требуется перекрасить компоненты, которые отвечают за их визуализацию.

Можно выделить следующие этапы выполнения задачи:

- Извлечение информации о выводах ячеек в виде, удобном для сравнения.
- Сравнение информации о выводах ячеек и получение данных о том, какие компоненты были изменены/удалены/вставлены.
- Перекраска необходимых компонентов в соответствующие цвета.

3.6.1. Извлечение информации о выводах ячеек

Для того, чтобы хранить информацию о выводах ячеек, в модуле `notebooks.vizualization` используется интерфейс `NotebookOutputDataKey`. Это ключи данных, инкапсулирующие в себе информацию, которую можно отображать посредством компонента Swing. Также внутри этого модуля есть интерфейс `NotebookOutputDataKeyExtractor`. Он берет на себя ответственность по поиску ключей данных для конкретного интервала ячейки. У него есть метод `extract(EditorImpl editor, Interval interval)`, возвращающий `List<NotebookOutputDataKey>`, если для интервала удалось найти ключи (то есть, у ячейки были выводы), и `null` в противном случае.

В конечном итоге был создан класс `DataExtractor`, который как раз отвечает за извлечение ключей с данными. Внутри него содержатся следующие методы:

- `List<NotebookOutputDataKey> getOutputDataKeys(Editor editor)`: позволяет получить список всех ключей данных для конкретного редактора. Из этого редактора извлекаются интервалы ячеек при помощи метода `extract(...)`, а из них получают необходимые ключи (интервалы, на которых метод вернул `null`, отбрасываются).

- `List<Interval> getOutputIntervals(Editor editor)`: позволяет получить интервалы ячеек, у которых есть выводы. Этот метод понадобится позже.

Однако этого все еще недостаточно. Проблема в том, что для сравнения данных нужно уметь извлекать из ключей информацию. Интерфейс `NotebookOutputDataKey` переопределен множеством классов, которые позволяют извлекать данные из ключа текста, картинки и так далее. Но нет возможности получить данные для сравнения через интерфейс `NotebookOutputDataKey`, не прибегая к использованию классов-наследников. Более того, в случае появления нового вида выводов ячеек могут быть проблемы с поддержкой.

Поэтому было решено добавить в интерфейс `NotebookOutputDataKey` метод `Object getContentForDiffing()`, который отвечал бы за извлечение данных для сравнения из ключа. Этот метод должны будут переопределять все классы, реализующие данный интерфейс, что избавит от необходимости обращаться к классам-наследникам и избавит от проблем в случае добавления новых типов выводов ячеек.

3.6.2. Сравнение информации о выводах ячеек

В модуле `diff.impl` есть следующий метод:
`public static <T> FairDiffIterable diff(T [] data1, T [] data2)`.
Он производит сравнение двух наборов элементов, базируясь на их `equals()` и `hashCode()` методах. В результате он создает итератор, который позволяет перебирать список объектов, формирующих разницу между наборами.

Сами изменения описываются посредством класса `Range`. Он содержит четыре поля - начало и конец изменения в первом наборе, а также начало и конец изменения во втором наборе.

Используя этот метод, сравниваются наборы текстовой информации, полученной из `NotebookOutputDataKey` посредством метода `getContentForDiffing()`.

Был создан класс `OutputKeysComparator`, который отвечает за реализацию логики, описанной выше. Данный класс имеет две вариации метода `compareKeys(...)` (для двухстороннего и трехстороннего сравнения). Он принимает списки `NotebookOutputDataKey`, а далее производит сравнение данных, полученных из ключей при помощи `getContentForDiffing()`.

3.6.3. Визуализация сравнения выводов ячеек

Для того, чтобы провести визуализацию, потребовалось сделать следующее:

- Извлечь компоненты Swing, которые требуется перекрасить.
- Осуществить перекраску самих компонентов и всех их подкомпонентов.

Как было указано ранее, для внесения графических элементов в документ используются декораторы (Inlays). Визуализация выводов ячеек также сделана при помощи декораторов. В редакторе для визуализации документа ноутбука используется большое количество декораторов, а поэтому среди них требовалось найти те, которые относятся к выводам ячеек. Для этого необходимо проверить тип данного декоратора.

После того как были найдены декораторы, относящиеся к выводам ячеек, из них было необходимо вытащить сами компоненты для перекраски. Для этого есть специальный метод, который возвращает список объектов `JComponent`. Это и есть компоненты Swing, требующие покраски.

Описанная выше логика была реализована в классе `ComponentExtractor`. У него есть следующие методы:

- `List<Inlay> getOutputInlays(Editor editor)`: находит все инкрустации выводов ячеек в данном редакторе.
- `List<JComponent> getOutputComponents(Editor editor)`: находит все компоненты Swing, отвечающие за выводы ячеек в данном редакторе. Внутри себя он использует предыдущий метод.

Далее требуется поменять цвета у извлеченных компонентов. Нужно выбрать цвет, а потом перебрать все элементы дерева, которые образуются подкомпонентами элементов, требующие покраски. Сама смена цвета осуществляется за счет записи нового значения в поле `background` у компонента.

Класс `DiffPainter` реализует данную логику. В него входит следующий метод:
`void paintDiffComponent(JComponent component, Editor editor, Side side)`.
В зависимости от стороны, на которой находится редактор, он выбирает нужный цвет. Далее он осуществляет перекраску компонента.

3.6.4. Итоговый алгоритм сравнения выводов ячеек

Результаты, описанные выше, можно объединить в следующий алгоритм:

1. Произвести загрузку необходимых ключей данных посредством `DataExtractor` (используя метод `getOutputDataKeys(...)`) и провести их сравнение при помощи `OutputKeysComparator`. Все эти действия лучше не производить в методе `onAfterRediff()`, так как он выполняется в Event Dispatch Thread. Поэтому они выполняются в методе `computeDifferences(...)`. Именно в нем происходит сравнение документов из запроса.

2. Извлечение компонентов. Основываясь на списке изменений, извлечь необходимые компоненты через `ComponentExtractor` (при помощи метода `getOutputComponents(...)`). Эти действия также исполняются в методе `computeDifferences(...)`.

3. Смена цвета извлеченных компонентов. Это делается через метод `paintDiffComponent(...)` класса `DiffPainter`. Эти действия следует выполнить в методе `onAfterRediff()`, так как он выполняется в Event Dispatch Thread.

Конечный результат можно видеть на рисунке 12:

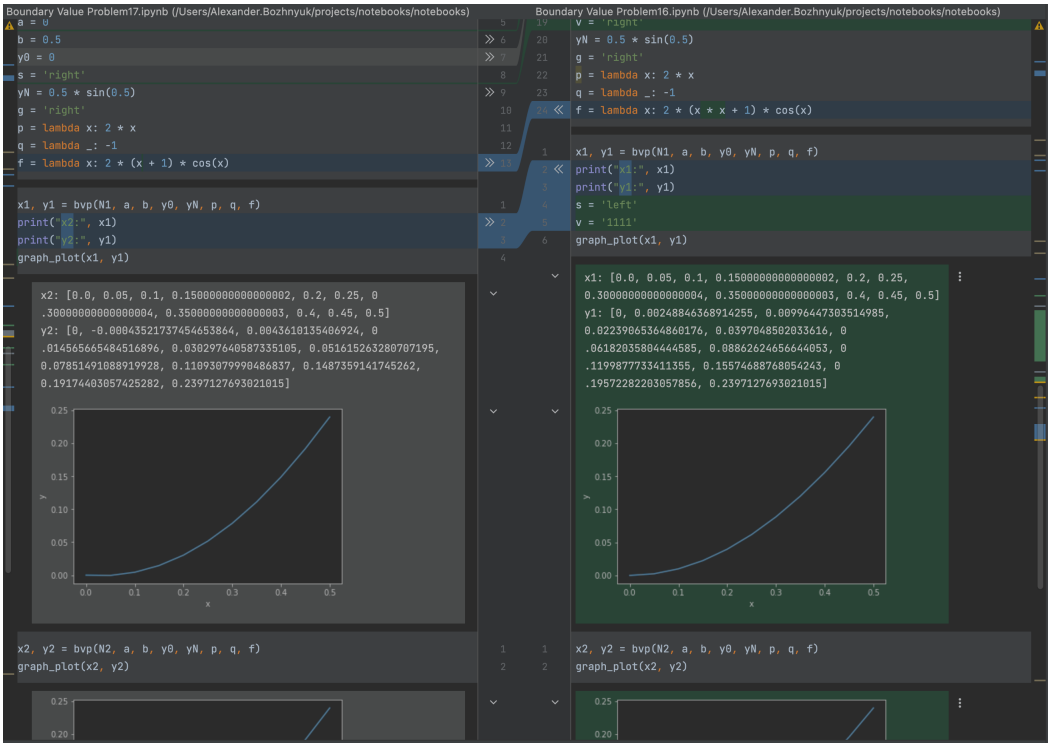

Рис. 12: Сравнение выводов ячеек

3.7. Инструмент слияния

Еще одной важной частью работы является создания инструмента слияния (Merge Tool). Он используется для того, чтобы разрешать конфликты, например, при слиянии веток или ребазировании.

Основная идея остается точно такой же. Она заключается в подмене JSON документа на ноутбучный. Отличия заключаются лишь в интерфейсах, которые необходимо было реализовывать. Однако даже эти отличия в архитектурном плане не очень сильны.

Для реализации своего инструмента слияния необходимо реализовать интерфейс `MergeTool`. Этот интерфейс имеет похожие методы, что и `DiffTool`. Метод `canShow(...)` ничем не отличается от предыдущей версии. Внутри метода `createComponent(...)` производится подмена `MergeRequest` на новый, с подмененными документами. На основе `MergeRequest` и `MergeContext` строится `MergeViewer`.

`MergeViewer` необходим для визуализации конфликтов и отображения необходимого интерфейса для их разрешения. Однако стоит отметить, что внутри себя он использует `DiffViewer` для визуализации сравнения.

При подмене также появлялось достаточно большое количество визуальных проблем, а поэтому потребовалось переопределить внутренний `DiffViewer` на свой. Здесь пришлось столкнуться с проблемой: этот `DiffViewer` являлся внутренним классом `MergeViewer`, что затрудняло выполнение задачи. Однако она решилась выделением внутреннего `DiffViewer` в отдельный класс в отдельном файле внутри модуля `diff.impl`. Далее был создан новый класс `JupyterMergeThreeSideViewer`, который является наследником внутреннего `DiffViewer`. Также, для правильного определения нового класса, внутри изначального `MergeViewer` потребовалось создать метод `loadViewer()`, который отвечает за создание внутреннего `DiffViewer`.

Такой прием позволяет сделать следующее: создать класс `JupyterMergeViewer`, унаследовав его от изначального `MergeViewer`, и внутри него переопределить метод `loadViewer()`, чтобы в нем создавался объект класса `JupyterMergeThreeSideViewer`.

Таким образом, настройку всего интерфейса, которая проводилась ранее, можно производить через настройку `JupyterMergeThreeSideViewer`. Там же было добавлено отключение рендеринга ячеек с кодом Markdown и другие улучшения визуализации. На рисунке 13 виден результат проделанной работы:

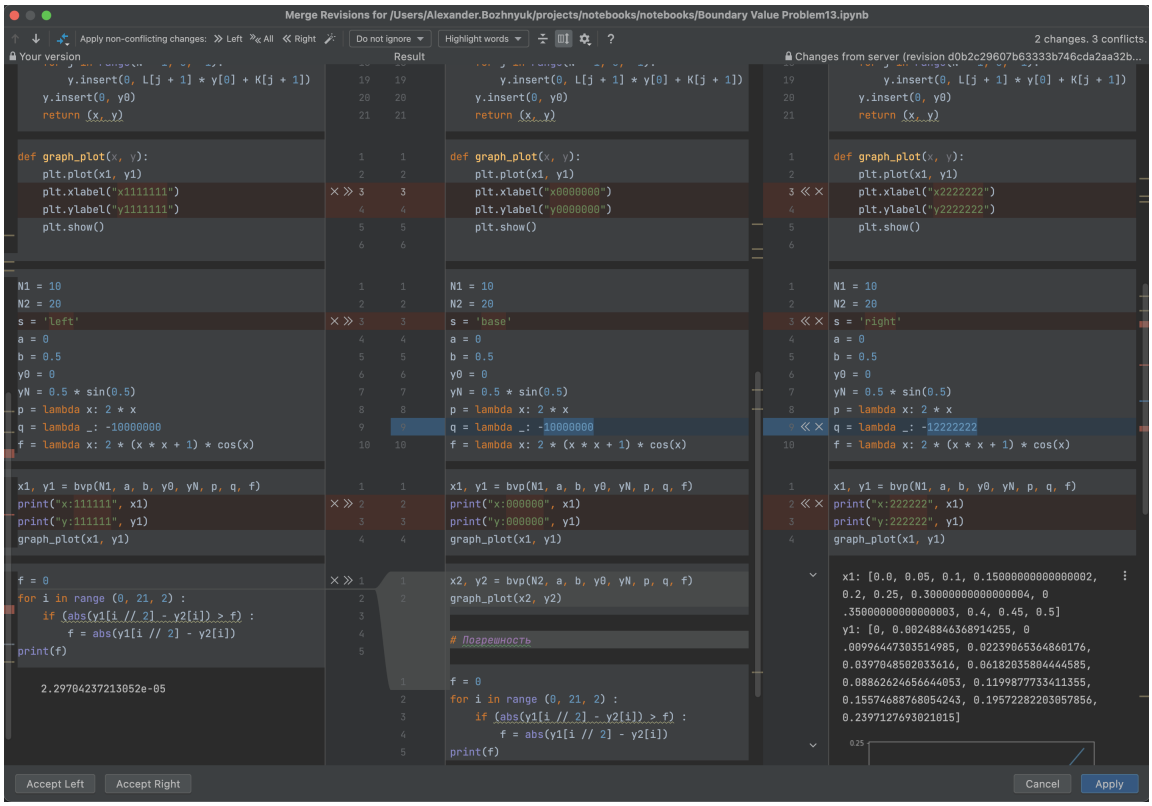

Рис. 13: Инструмент слияния

4. Результаты

Благодаря работе над производственной практикой были достигнуты следующие результаты:

- Проведен обзор проекта Jupyter.
- Проведен обзор алгоритмов для сравнения файлов.
- Проведен обзор конкурентов.
- Проведен обзор платформы IntelliJ.
- Реализован инструмент сравнения ноутбуков.
- Исправлены многочисленные проблемы, связанные с визуализацией ноутбуков Jupyter в режиме сравнения.
- Реализована система сравнения ячеек с кодом на языке Markdown.
- Реализована система сравнения выводов ячеек.
- Реализован инструмент слияния веток.

5. Список литературы

[1] Jupyter Notebooks:
https://jupyter.org/
Дата обращения: 6 Декабря 2021 г.

[2] JetBrains:
https://www.jetbrains.com/
Дата обращения: 6 Декабря 2021 г.

[3] JetBrains DataSpell: IDE for data science:
https://www.jetbrains.com/ru-ru/dataspell/
Дата обращения: 6 Декабря 2021 г.

[4] Diff tool for Linux example:
https://man7.org/linux/man-pages/man1/diff.1.html
Дата обращения: 6 Декабря 2021 г.

[5] Fernando Pérez:
https://bids.berkeley.edu/people/fernando-p%C3%A9rez
Дата обращения: 7 Декабря 2021 г.

[6] Brian Granger:
https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/speaker/146215.html
Дата обращения: 7 Декабря 2021 г.

[7] Jupyter Kernels:
https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
Дата обращения: 8 Декабря 2021 г.

[8] *J. W. Hunt, M. D. McIlroy*, An Algorithm for Differential File Comparison:
https://www.cs.dartmouth.edu/~doug/diff.pdf
Computing Science Technical Report, Bell Laboratories, 1976.

[9] *Eugene W. Myers*, An O(ND) Difference Algorithm and Its Variations:
http://www.xmailserver.org/diff2.pdf
Algorithmica journal, Vol. 1, no. 2. — P. 251—266, 1986.

[10] nbdime – diffing and merging of Jupyter Notebooks:
https://nbdime.readthedocs.io/en/latest/
Дата обращения: 11 Декабря 2021 г.

[11] Visual Studio Code:
https://code.visualstudio.com/
Дата обращения: 11 Декабря 2021 г.

[12] Jupyter Notebooks in VS Code:
https://code.visualstudio.com/docs/datascience/jupyter-notebooks
Дата обращения: 11 Декабря 2021 г.

[13] DagsHub: Open Source Data Science Collaboration:
https://dagshub.com/
Дата обращения: 11 Декабря 2021 г.

[14] GitHub:
https://github.com/
Дата обращения: 11 Декабря 2021 г.

[15] Curvenote: Writing platform for science
https://curvenote.com/
Дата обращения: 11 Декабря 2021 г.

[16] ReviewNB: Rich Diffs Commenting for Jupyter Notebooks
https://www.reviewnb.com/
Дата обращения: 11 Декабря 2021 г.

[17] Jupyter Notebooks Metadata:
https://jupyterbook.org/content/metadata.html
Дата обращения: 11 Декабря 2021 г.

[18] IntelliJ Platform:
https://plugins.jetbrains.com/docs/intellij/welcome.html
Дата обращения: 24 Марта 2022 г.

[19] IntelliJ Document:
https://plugins.jetbrains.com/docs/intellij/documents.html
Дата обращения: 24 Марта 2022 г.

[20] IntelliJ Editor:
https://plugins.jetbrains.com/docs/intellij/editors.html
Дата обращения: 24 Марта 2022 г.

[21] IntelliJ Extension points:
https://plugins.jetbrains.com/docs/intellij/plugin-extension-points.html
Дата обращения: 24 Марта 2022 г.

[22] IntelliJ Extension:
https://plugins.jetbrains.com/docs/intellij/plugin-extensions.html
Дата обращения: 24 Марта 2022 г.