

Санкт-Петербургский государственный университет

Математическое обеспечение и
администрирование информационных систем

Кафедра системного программирования

Коекин Ярослав Алексеевич

F# Interactive в IDE Rider

Курсовая работа

Научный руководитель:
ст. преп. Я. А. Кириленко

Консультант:
JetBrains, программист-инженер
Е. П. Аудучинок

Санкт-Петербург

2020

Содержание

Введение	3
1 Постановка задачи	5
2 Обзор	7
2.1 Существующие решения	7
2.2 Вывод	8
3 Решение	9
3.1 Анализ существующей реализации	9
3.2 Добавление функциональных возможностей для редактора кода	11
3.3 Реализация подсветки синтаксиса в истории вывода	12
3.4 Решения по изменению интерфейса	13
3.5 Внедрение в существующее решение	13
4 Заключение	16
5 Благодарности	17
Список используемой литературы	18

Введение

Интерактивная среда языка программирования – это программное средство, позволяющее использовать возможности конкретного языка программирования в рамках интерфейса командной строки, в котором инструкции компьютеру подаются путём ввода текстовых строк (команд) с клавиатуры. В основе таких программных средств лежит принцип REPL (Read-Eval-Print-Loop), который задаёт правила взаимодействия с интерфейсом: среда выполняет выражения и осуществляет вывод результатов сразу после их отправки пользователем в систему.

Различные интерактивные среды языков программирования встраивают в интегрированные среды разработки программного обеспечения для увеличения эффективности создания продуктов. Такие инструменты позволяют тестировать небольшие фрагменты кода, проверять утверждения, не отвлекаясь от процесса разработки, а также предоставляют инструменты для удобной интеграции проверенного кода с основной частью проекта.

JetBrains Rider¹ – кросс-платформенная интегрированная среда разработки программного обеспечения. Поддерживает проекты, основанные на платформах .Net Framework, .Net Core и Mono. Предоставляет инструменты для эффективной разработки программных продуктов. Так, например, к функциональным возможностям Rider можно отнести: анализ кода, подсветку синтаксиса кода, перепроектирование программного продукта, систему навигации по проекту и много другое. Продукт был анонсирован в январе 2016 года, активно развивается и по сей день.

JetBrains Rider поддерживает языки программирования C#, VB.NET. В 2017 году выпускником математико-механического факультета была добавлена поддержка функционального языка про-

¹jetbrains.com/rider

граммирования F# в Rider [3].

JetBrains Rider основан на двух продуктах JetBrains: IntelliJ Platform² и ReSharper³.

IntelliJ Platform – open-source платформа, предназначенная для создания интегрированных сред разработки. Предоставляет инструменты для конструирования инфраструктуры среды: компоненты для объявления редакторов кода, компоненты для работы с моделью проекта, система виртуальных файлов, компоненты для построения интерфейса пользователя и другие.

Resharper изначально был создан как расширение для Microsoft Visual Studio⁴. Предоставляет продвинутые инструменты для инспекции, рефакторинга и навигации по коду. Поддерживает языки C# и VB.NET.

На данный момент Rider предоставляет интерактивную среду языка программирования F#, которая позволяет исполнять исходный код и выводить результаты. Инструменты, упрощающие разработку приложений в основном редакторе IDE, также упрощают работу и в интерактивных средах языков программирования. Подобные рассуждения являются мотивацией для расширения функциональных возможностей интерактивной консоли. F#

²github.com/JetBrains/intellij-community

³jetbrains.com/resharper

⁴<https://visualstudio.microsoft.com>

1 Постановка задачи

Такие инструменты как подсветка синтаксиса, система подсказок при выделении синтаксической сущности, система перехода от сущности к участку кода, где она объявлена (система навигации) и другие упрощают процесс написания кода и в интерактивных средах языков программирования.

Помимо этого, существуют специфичные для таких сред инструменты (например, возможность передать код из некоторого участка проекта в интерактивную консоль). К специфике интерактивных консолей также можно отнести наличие двух частей, из которых они состоят: редактор, позволяющий ввести команду на языке программирования и часть, отображающая историю исполнения команд.

Расширение для поддержки языка F# в JetBrains Rider имеет реализацию интерактивной консоли, однако её возможности на данный момент ограничиваются вводом команд, их исполнением и выводом результатов.

Таким образом, целью данной работы является расширение функциональных возможностей F# Interactive для IDE JetBrains Rider.

Для достижения данной цели были поставлены следующие задачи:

- Проанализировать существующую реализацию интерактивной консоли F# в расширении для IDE Rider.
- Реализовать в F# Interactive на уровне редактора:
 - систему автодополнений;
 - подсветку кода;
 - систему подсказок tooltips;
 - систему навигации по коду.

- Реализовать подсветку синтаксиса для вводимых команд и выводимых конструкций на уровне вывода истории команд.
- Внести изменения в интерфейс с целью сделать его более удобным для пользователя.
- Внедрить разработанные возможности в существующее решение.

2 Обзор

Инструменты, на которых основывается JetBrains Rider – IntelliJ Platform и ReSharper – исполняются независимо, в разных процессах, коммуникация происходит с помощью Reactive Distributed Communication⁵. И IntelliJ Platform, и ReSharper предоставляют инструменты для добавления функциональных возможностей к уже имеющимся с помощью системы расширений.

Таким образом осуществляется и поддержка языка F# в среде Rider. При этом расширение происходит и на уровне IntelliJ Platform, и на уровне ReSharper. Поддержка языка F# реализована с использованием библиотеки FSharp.Compiler.Service⁶, которая предоставляет дополнительные возможности при работе с F# на уровне компилятора и рефакторинга. Эти знания необходимы при разработке новых функциональных возможностей интерактивной консоли.

2.1 Существующие решения

F# Interactive в Microsoft Visual Studio

Интерактивная консоль F# в Visual Studio имеет возможность ссылаться на проекты и библиотеки, что позволяет использовать классы, переменные и процедуры в интерактивном режиме, не дублируя их. К недостаткам этого инструмента можно отнести то, что он не имеет подсветки синтаксиса, системы подсказок и системы навигации. Также, такое решение не может поддерживаться средствами IDE Rider.

Ionide F# Interactive

⁵<https://github.com/JetBrains/rd/tree/master/rd-net>

⁶<https://github.com/fsharp/FSharp.Compiler.Service>

Ionide⁷ – кросс-платформенное open-source расширение для Microsoft Visual Studio Code⁸ и редактора Atom⁹. Ionide предоставляет инструменты для поддержки языка F#, например: автодополнение, рефакторинг, систему навигации, систему подсказок, интеграцию с MsBuild и другие. Для реализации таких инструментов Ionide использует библиотеку FSharp.Compiler.Service. Для эффективной разработки приложений на языке F# Ionide предоставляет интерактивную консоль. Её преимущества: наличие автодополнения кода, подсветка синтаксиса, генерация скриптовых файлов. Недостатки: отсутствие системы подсказок, отсутствие системы навигации, недоработанная подсветка синтаксиса.

2.2 Вывод

В связи с тем, что F# Interactive для Microsoft Visual Studio имеет закрытый код, а Ionide является расширением для платформы, не совместимой с платформами, на которых основывается IDE Rider, их использование для достижения поставленной цели является нецелесообразным.

⁷<http://ionide.io>

⁸<https://code.visualstudio.com>

⁹<https://atom.io/>

3 Решение

3.1 Анализ существующей реализации

Для того, чтобы реализовывать дополнительные функциональные возможности F# Interactive, необходимо проанализировать архитектуру существующей реализации консоли на уровне IntelliJ Platform и ReSharper Platform. Представление такой архитектуры показано на рис. 1. На нём видны основные сущности реализации, а именно:

- FsiHost (на стороне IntelliJ Platform). Отвечает за создание и запуск интерактивной консоли. Предоставляет инструменты для отправки запроса извне интерактивной консоли (это используется, например, при отправке части кода напрямую из документа проекта в консоль).
- FsiConsoleRunner. Основная сущность интерактивной консоли F#. Именно она запускает новый процесс F# REPL, который принимает запросы пользователя, обрабатывает их и выводит информацию. Также она создаёт обработчик этого процесса (который имеет доступ к выводу) и вид консоли, который отвечает за отображение редактора кода и вывод отправленных запросов с ответами на них от F# REPL в окно истории.
- CommandHistory. Хранит историю введённых запросов.

Стоит отметить, что информация, необходимая для запуска F# REPL (путь до запускаемого файла, название, аргументы запуска), хранится на стороне ReSharper Platform. Таким образом, перед созданием и запуском интерактивной консоли, FsiHost на стороне IntelliJ Platform отправляет запрос FsiHost на стороне ReSharper Platform, который, в свою очередь, передаёт информацию обратно

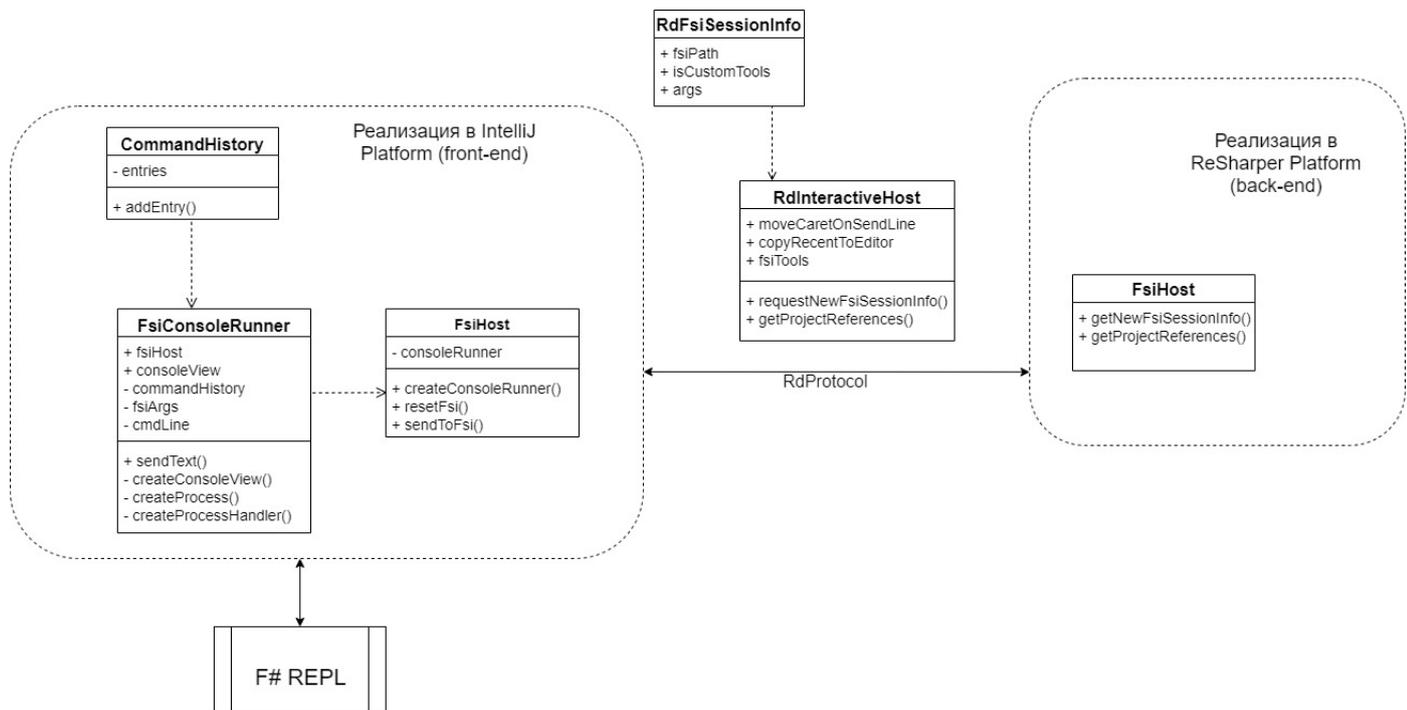


Рис. 1: Существующая реализация

в виде объекта класса `FsiSessionInfo`. Такое взаимодействие обеспечивается с помощью `RdProtocol`.

При работе с интерактивной консолью **F#** пользователь взаимодействует с двумя редакторами: редактор кода и редактор вывода истории. Каждый редактор хранит документ, который необходим для представления содержимого текста в нём. В ходе анализа существующей реализации было выяснено, что оба они не находятся физически на диске. В этом случае `IntelliJ Platform Sdk` даёт возможность `ReSharper` анализировать контент документа редактора с помощью создания его клона на сервере. На сервере создаётся так называемый `сендбокс-документ`, связанный с документом на стороне клиента. Для этого на стороне `IntelliJ Platform` необходимо привязать к редактору объект класса `SandboxInfo`, который хранит информацию о языке, который используется в редакторе, ссылки на модули и библиотеки и другие опции, и предоставить обработчик создания нового `сендбокс-документа` на стороне сервера.

Воспользовавшись такой системой, можно иметь доступ к подсветке кода, а также возможность использовать систему подсказок и навигации в редакторе кода. Более того, для адекватного разрешения автодополнений необходимо предоставлять информацию объекту класса `SandboxInfo` о ранее успешно выполненных запросах.

3.2 Добавление функциональных возможностей для редактора кода

Для редактора кода необходимо добавить следующие функциональные возможности: подсветку, систему навигации и подсказок, автодополнение. Для дополнения первых трёх была разработана архитектура, использующая паттерн наблюдатель: объект, создающий сендбокс-документ для документа редактора кода подписывается на событие создания редактора.

Для адекватной работы системы автодополнений необходимо прибавлять к документу, который находится на стороне `ReSharper Platform` успешно выполненные `F# REPL` запросы. Для этого необходимо проанализировать ответ на запрос на наличие ошибок. Если таких не оказывается, необходимо пометить данный запрос как успешно выполненный. Все помеченные как успешно выполненные команды добавляются в новый объект класса `SandboxInfo`. Далее для этого объекта устанавливается связь с текущим документом редактора кода. Для создания нового сендбокс-документа на стороне `ReSharper Platform`, необходимо сообщить о внесённых изменениях в редакторе средствами `API IntelliJ Platform`.

Такой подход приводит к конфликтам состояний интерактивной консоли и сендбокс-документа. Например, если несколько раз корректно определить одну и ту же функцию, то интерактивная консоль переопределит все остальные последним, тогда как реализа-

ция, описанная выше, провоцирует анализировать F# код, в котором присутствует неоднократное определение одной и той же функции. Это приводит к ошибкам анализа.

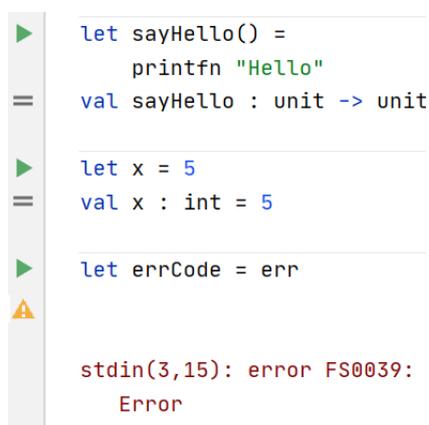
Для избежания такой ситуации было принято решение использовать особенности языка программирования F#, а именно атрибут `AutoOpen`. На помеченные таким атрибутом вложенные модули ссылаются автоматически при попытке сослаться на внешний модуль. Таким образом, для моделирования состояния интерактивной консоли существует возможность оборачивать каждый успешно выполненный запрос в новый модуль, помеченный атрибутом `AutoOpen`. В целях разграничения знаний о языке между клиентом и сервером и в целях возможного расширения в будущем было решено оборачивать запрос на стороне `ReSharper Platform`. Для этого была создана модель в рамках `RdProtocol`, которая позволяет отправлять ещё неподготовленные команды на сервер для обработки и принимать их в желаемом виде.

3.3 Реализация подсветки синтаксиса в истории вывода

Для подсветки синтаксиса в истории вывода можно воспользоваться средствами `API IntelliJ Platform`. Реализация языка F# в `Rider` позволяет использовать лексер этого языка на стороне `IntelliJ Platform`. Для подсветки синтаксиса вывода F# `REPL` необходимо было переопределить обработчик вывода на собственный. Новый обработчик учитывает тип вывода (обычный, системный и вывод с ошибками) и печатает текст в окне истории согласно соответствующим цветовым схемам.

3.4 Решения по изменению интерфейса

В данной работе были произведены некоторые действия по изменению интерфейса интерактивной консоли. К таким изменениям можно отнести: отделение различных запросов друг от друга, отделение запроса и вывода и добавление иконок для более интуитивного процесса взаимодействия с консолью. Текущий пользовательский интерфейс изображён на рис. 2



```
▶ let sayHello() =  
    printfn "Hello"  
=  
val sayHello : unit -> unit  
  
▶ let x = 5  
=  
val x : int = 5  
  
▶ let errCode = err  
⚠  
  
stdin(3,15): error FS0039:  
Error
```

Рис. 2: Интерфейс пользователя.

3.5 Внедрение в существующее решение

Таким образом, диаграмма классов реализации новых функциональных возможностей интерактивной консоли F# представлены на рис. 4 и рис. 3. Жёлтым цветом выделены те классы, которые были разработаны и реализованы в рамках данной работы. Синим – изменённые существовавшие классы. Белым – сущности, не изменённые в процессе работы.

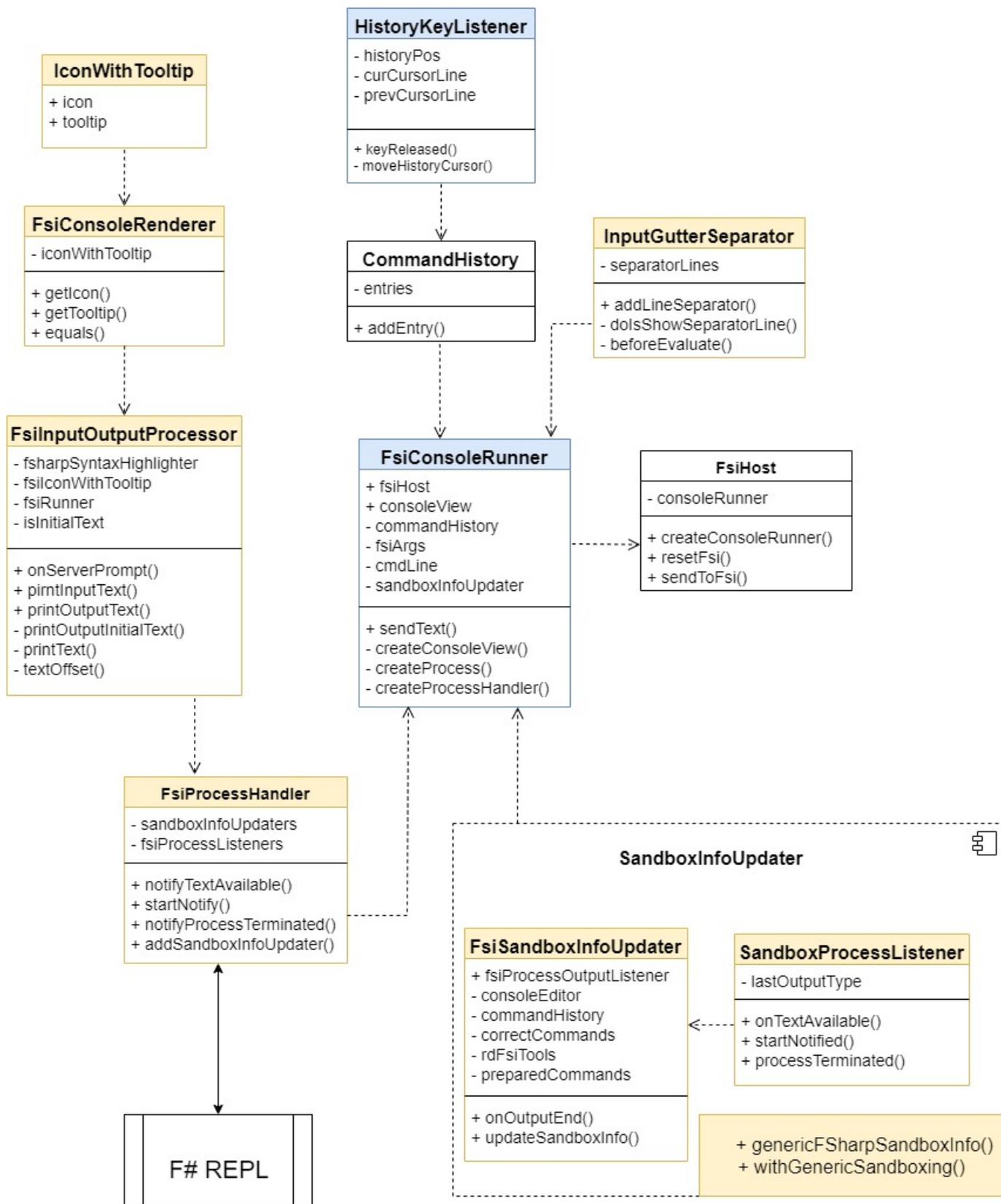


Рис. 4: Внедрение. Клиент.

4 Заключение

В рамках данной работы были получены следующие результаты.

- Был произведён разбор существующей реализации путём выделения модулей, имеющих непосредственное отношение к консоли.
- Реализованы в интерактивной консоли F# на уровне редактора:
 - система автодополнений;
 - подсветка кода;
 - система подсказок tooltips;
 - система навигации по коду.
- Реализована подсветка синтаксиса для вводимых команд и выводимых конструкций на уровне вывода истории команд.
- Изменён интерфейс интерактивной консоли на более дружелюбный с точки зрения пользователя.
- Реализация внедрена в существующее решение.

5 Благодарности

Огромная благодарность команде JetBrains Rider за ценный опыт работы в большой компании. Отдельно хочу выделить куратора проекта, Евгения Аудучинок, который помогал мне на протяжении всей работы. Также хочу поблагодарить Александра Кирсанова и Анну Громову, которые делились советами в процессе работы.

Список используемой литературы

- [1] IntelliJ Platform SDK DevGuide:
<http://www.jetbrains.org/intellij/sdk/docs/welcome.html>
(дата обращения: 05.20.2020)
- [2] ReSharper DevGuide:
<http://www.jetbrains.org/intellij/sdk/docs/welcome.html>
(дата обращения: 05.20.2020)
- [3] Е. П. Аудучинок. Реализация поддержки языка F# в интегрированной среде разработки JetBrains Rider. 2017.
<http://hdl.handle.net/11701/10571>
- [4] F# Compiler Services:
<https://fsharp.github.io/FSharp.Compiler.Service/>
(дата обращения: 05.20.2020)
- [5] F# Software Foundation:
<https://fsharp.org/>
(дата обращения: 05.20.2020)
- [6] F# Interactive Service:
<https://fsharp.github.io/FSharp.Compiler.Service/interactive.html>
(дата обращения: 05.20.2020)
- [7] F# Interactive Options:
<https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/fsharp-interactive-options>
(дата обращения: 05.20.2020)