# Реферат

# Введение

# Постановка задачи

## Обзор и анализ прототипов

## Актуальность решаемой задачи

# 3. Архитектура проекта

## 3.1 Общая структура

## 3.2 Модуль LfkSharedResources

Реализация : NetworkPackageController, Huffman Encoding

## **3.3 Модуль LfkGUI**

**Предназначение модуля**

Данный программный модуль предназначен для визуализации рабочего процесса системы контроля версий Lfk. LfkGUI частично реализует фильтрацию и валидацию вводимых пользователем данных. Также он предоставляет возможность ведения диалога с пользователем, и визуализацию сообщений об ошибках, возникающих в других модулях.

**Структура модуля LfkGUI**

Данный модуль реализован на основе популярного шаблона проектирования MVVM(Model-View-ViewModel). В его состав входят представления, модели представлений, а также валидационные правила и дополнительные сервисы, отвечающие за диалоговые окна и навигацию: WindowService – отвечает за навигацию между окнами, DialogService – отвечает за вывод сообщений пользователю, FolderOpenDialogService – отвечает за диалоговое окно для выбора директорий. LfkGUI также содержит ряд ресурсов, которые содержат стили, словари локализации. На рисунке 3.3.2.1 представлена схема взаимодействия компонентов модуля LfkGUI.



Рисунок 3.3.1 – Схема взаимодействия компонентов модуля LfkGUI

**Диаграммы, описание и реализация классов модуля**

При разработке данного проекта использовался паттерн проектирования MVVM, позволяющий отделить представление данных от их обработки. На рисунке 3.3.2 представлена диаграмма классов, отвечающих за представление моделей(View). Связанные с ними классы моделей представлений показаны далее на рисунке 3.3.5.

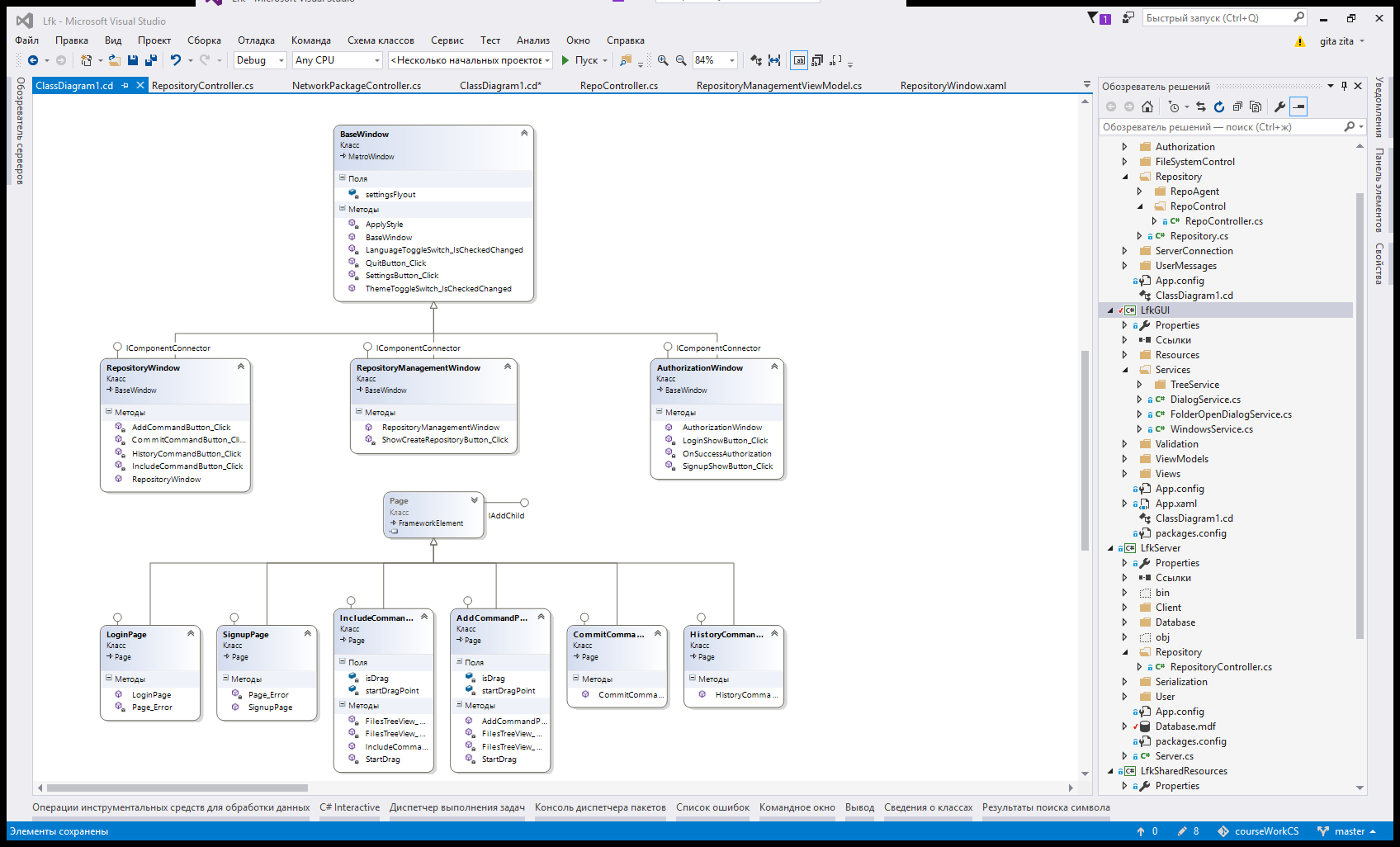


Рисунок 3.3.2 – Диаграмма классов представлений

Класс BaseWindow является базовым классом для используемых в проекте окон, данный класс описывает основной стиль графического интерфейса, также в нем реализованы функции меню общие для всех окон программы, такие как смена стиля программы, смена языка программы и возможность сменить пользователя в любом окне программы. Необходимо отметить также, что данный класс в силу особенностей наследования классов WPF, не содержит в себе XAML разметки, и все базовые стили и функционал прописаны программно. На диаграмме классов видно, что от класса BaseWindow наследуются три класса, которые и являются главными окнами взаимодействия с сущностью LfkClient.

Класс AuthorizationWindow отвечает за авторизацию пользователя, и содержит в себе две страницы LoginPage и SignupPage. LoginPage отвечает за авторизацию пользователя и двусторонне связан с моделью представления LoginViewModel, которая в свою очередь отвечает за вход и идентификацию пользователя. В свою очередь SignupPage визуализирует процесс регистрации пользователя и связан с SignupViewModel. Класс RepositoryManagementWindow представляет пользователю возможность управления пользовательскими репозиториями. Класс RepositoryWindow отвечает за визуализацию действий производимых с пользовательскими файлами, и основных комманд системы контроля версий. Каждой комманде соответствует свой класс Page, который связан с моделью представлений.

Хотелось бы отметить что в визуальном представлении рабочей директории используется элемент TreeView, который позволяет представлять элементы иерархически, в виде раскрывающихся списков. Специально для этого была создана модель являющаяся узлом дерева, а для формирования дерева был написан класс TreeBuilder, который из списка файлов формирует дерево, также в процессе управления файлами пользователь может динамически добавлять и удалять узлы дерева. На рисунке 3.3.3 представлена структура этих классов.

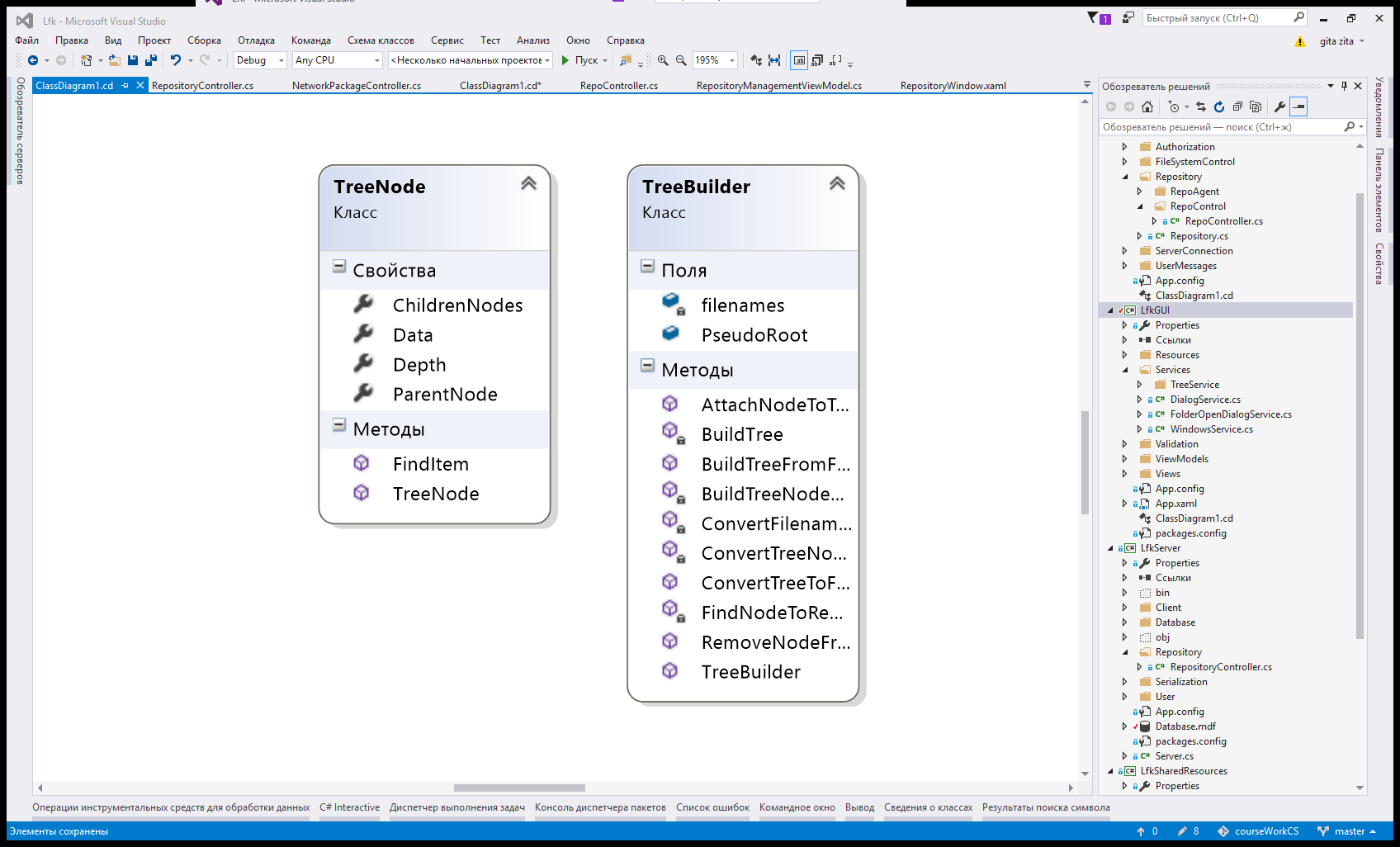


Рисунок 3.3.3 – Диаграмма классов для представления пользовательской директории

Механизмы построения, добавления, и удаления элементов, реализованы на основе рекурсивных алгоритмов обхода дерева. На рисунке 3.3.4 представлен код рекурсивной функции, которая позволяет преобразовать узел в список путей файлов.

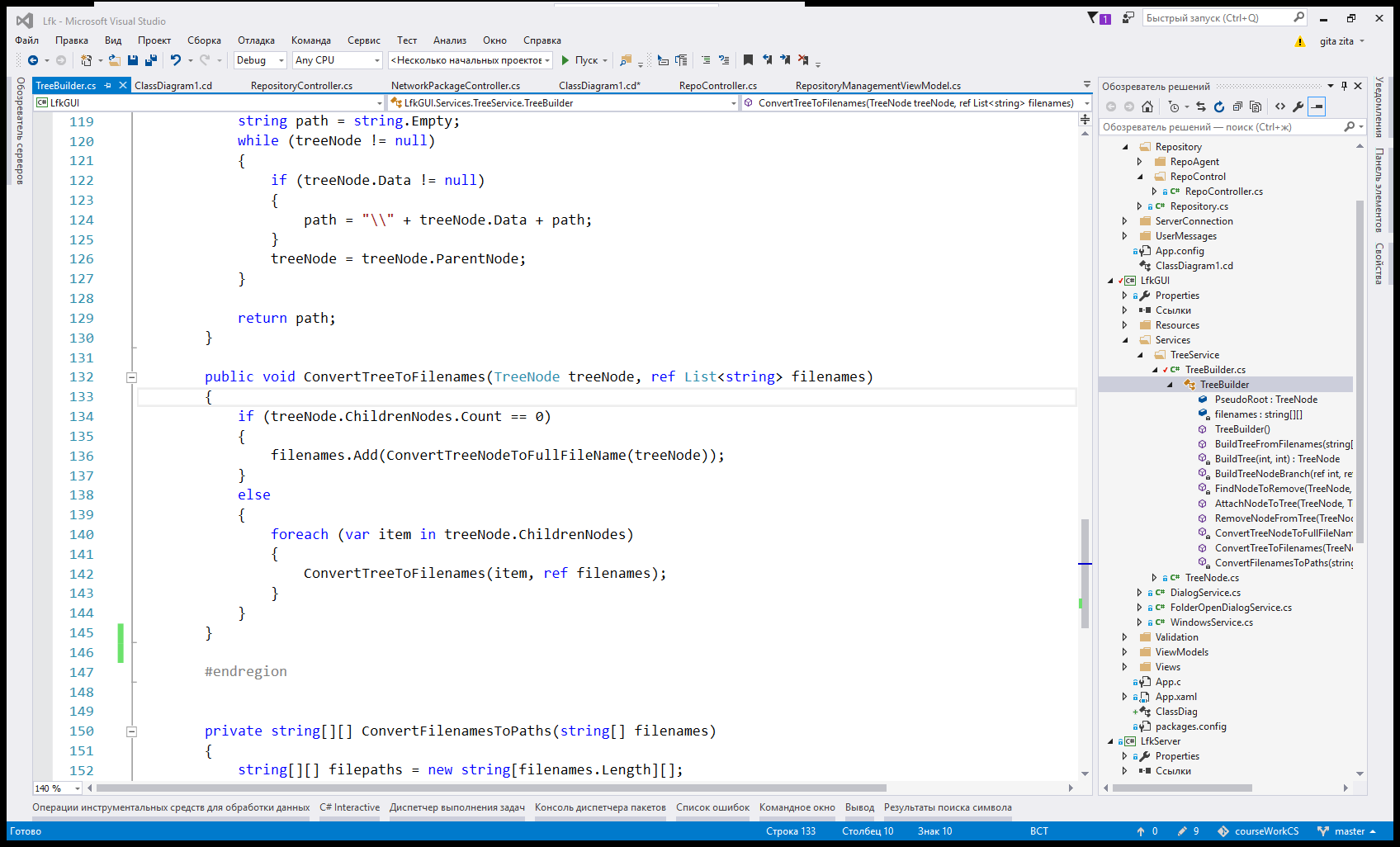


Рисунок 3.3.4 – Функция построения путей файлов из узла дерева

Ниже представлена диаграмма классов моделей представления, тоесть классов, которые отвечают за обработку пользовательских действий, и последующий вызов функций остальных модулей. Необходим отметить что базовый для всех моделей представления класс реализует интерфейс INotifyPropertyChanged, что позволяет оповещать привязанные к свойствам контролы, об изменении свойств. Также активно используется специальный класс ObservableCollection. Он по функциональности похож на список List за тем исключением, что позволяет известить внешние объекты о том, что коллекция была изменена.

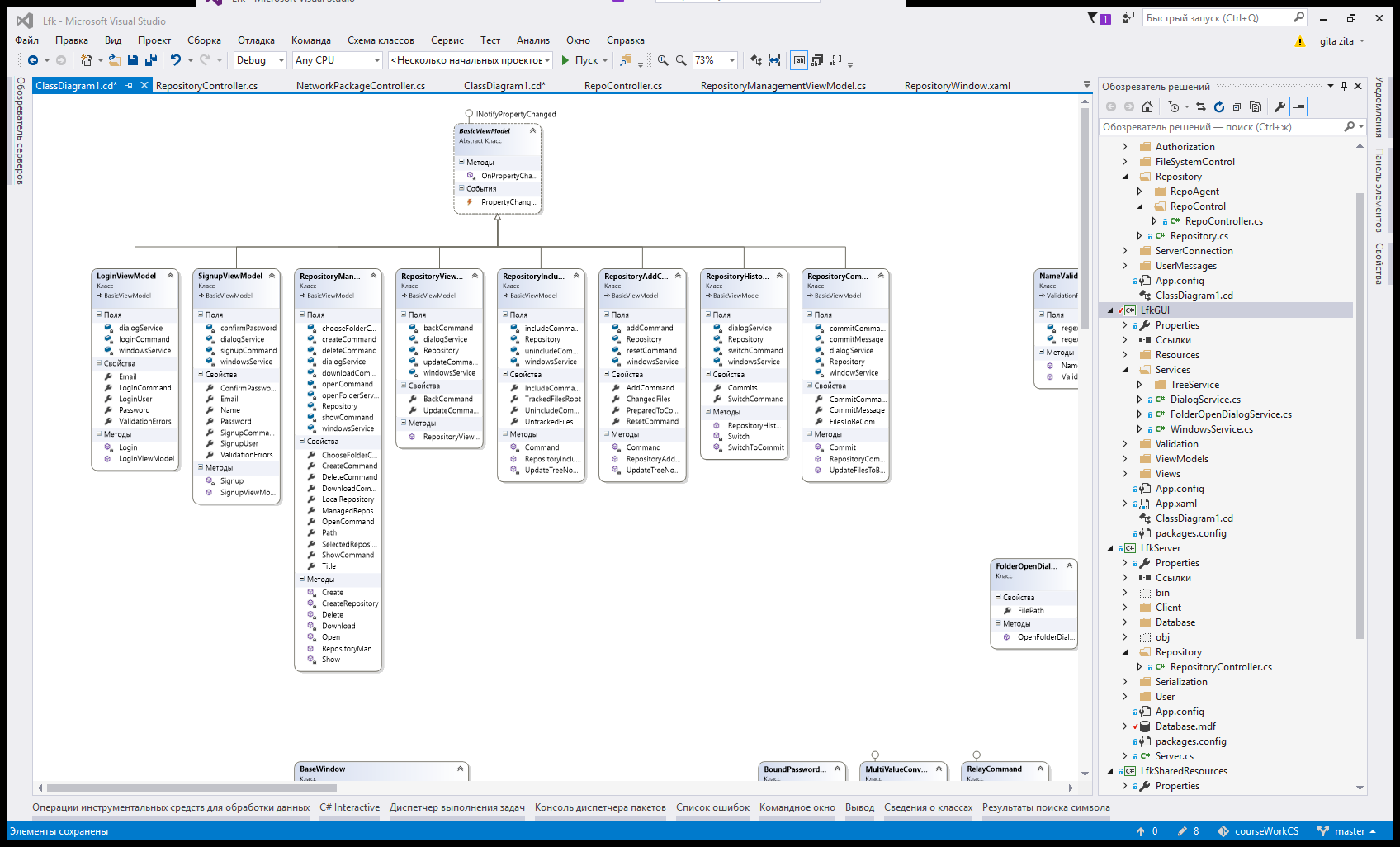


Рисунок 3.3.5 – Диаграмма классов моделей представления

На рисунке 3.3.6 представлен код реализации модели представления LoginViewModel, который отражает основные принципы построения ViewModel в проекте LfkGUI.

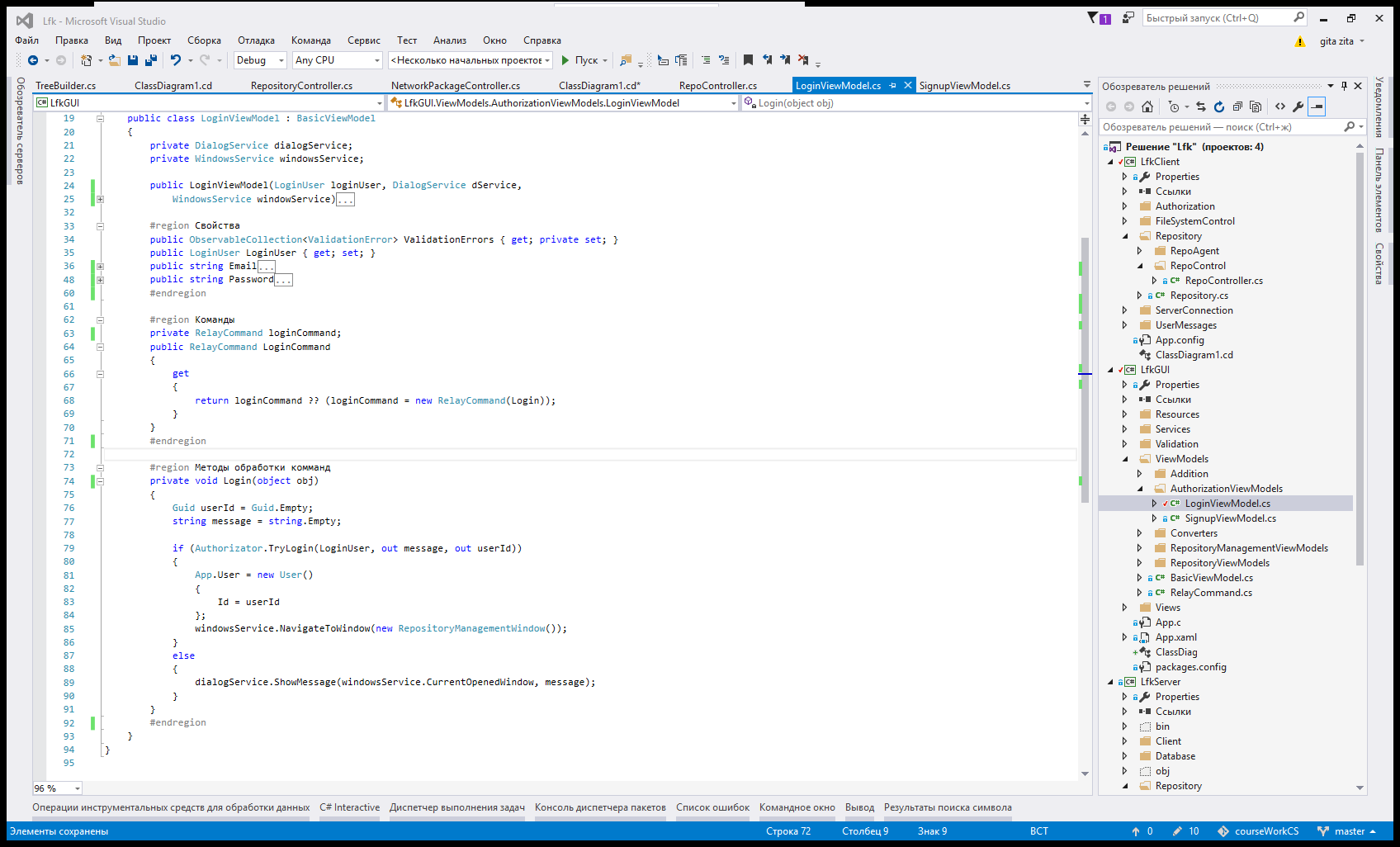


Рисунок 3.3.6 – Код реализации модели представления

Каждая модель представления содержит в себе свойства, привязанные к элементам интерфейса, также, существует система валидации вводимых пользователем данных, которая основана на том, что пользователю не позволяется выполнять команды, пока введенные данные не удовлетворят необходимым условиям. Данная система основана на валидационных правилах и командах. Ошибки отлавливаются на уровне представления, и при наличии хоть одной, во избежание непредвиденных последствий выполнение основных команд блокируется, и элемент управления отвечающий за исполнение команды становится неактивным. На рисунке 3.3.4 представлена диаграмма классов, отвечающих за валидацию, которая в основана на механизме регулярных выражений. Также ниже отображен класс, представляющий собой команду.

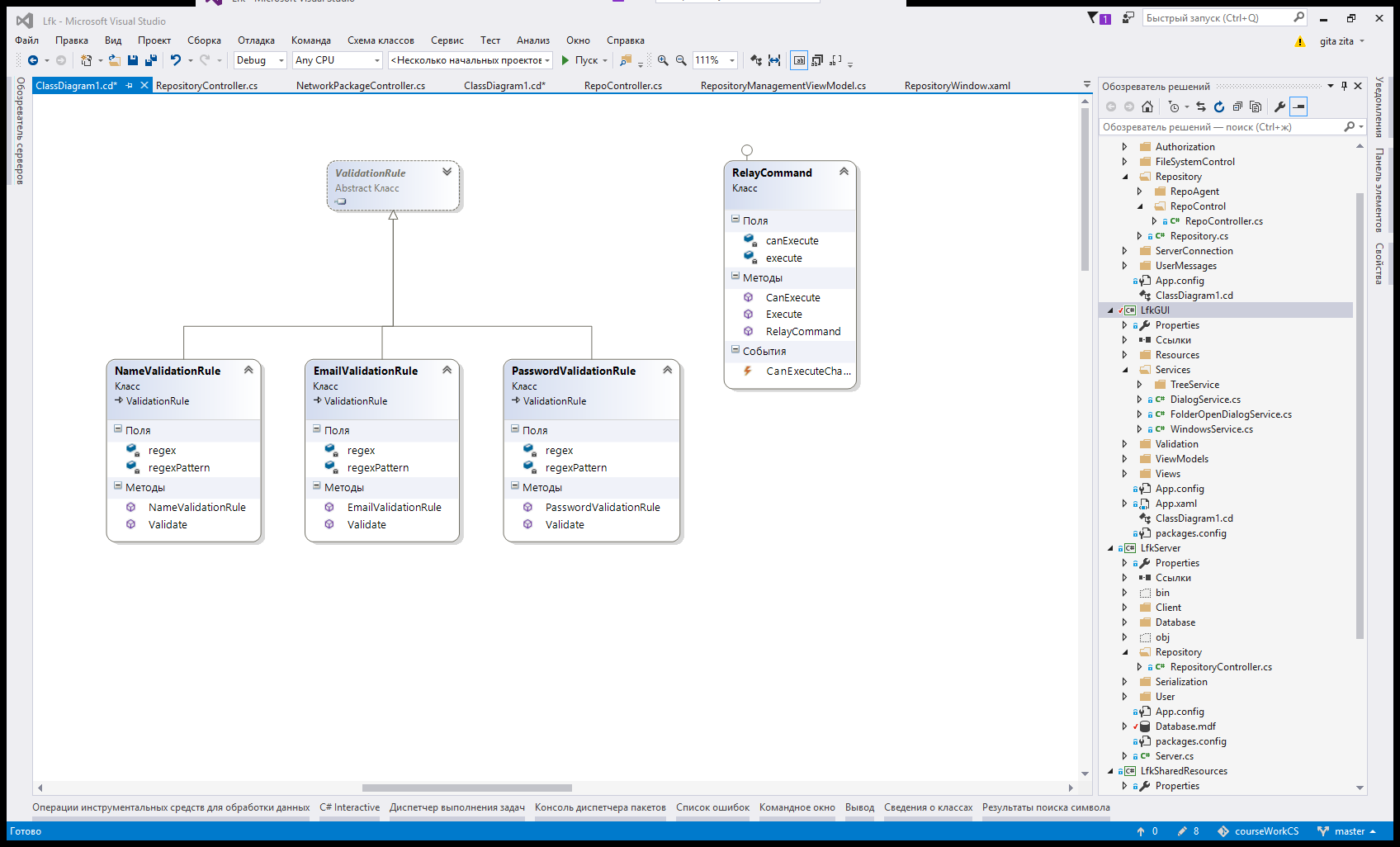


Рисунок 3.3.7 – Диаграмма классов валидации и класса команды

Программная реализация одного из валидационного правила представлена на рисунке 3.3.8

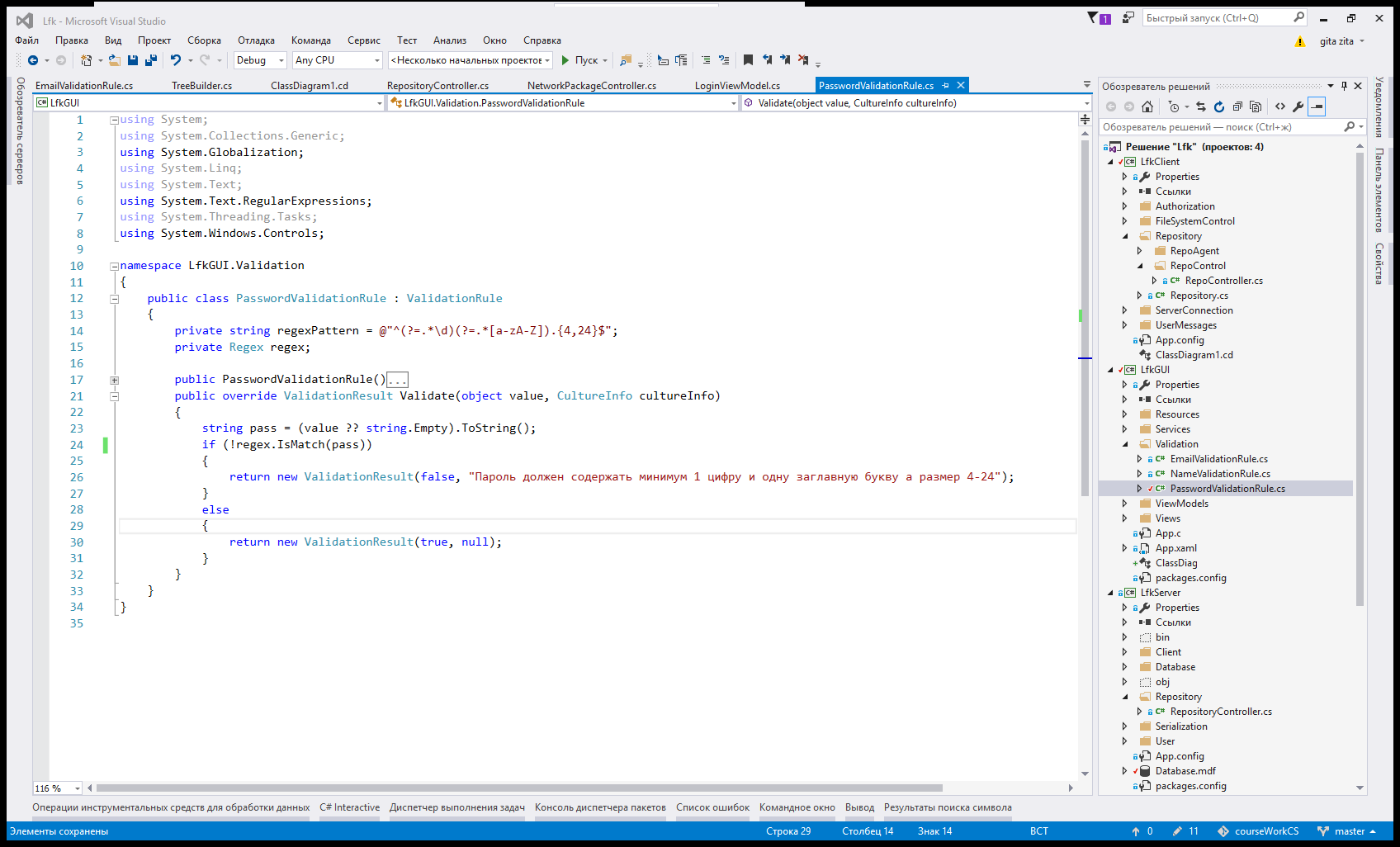


Рисунок 3.3.8 – Код реализации класса отвечающего за валидацию

Необходимо отметить, что для стилизации графического интерфейса использовалась библиотека MaterialDesign, которая включает в себя огромный набор стилей, для всех видов элементов управления. Также для расширения функционала стандартного класса Window была подключена библиотека MahApps.Metro, которая позволила добавить меню, общее для всех окон, и систему диалоговых окон.

## **3.4 Модуль LfkClient**

**Предназначение модуля**

Данный программный модуль предназначен для управления авторизацией, работы с основным функционалом системы контроля версий, и взаимодействия с серверной частью приложения.

**Структура модуля LfkClient**

LfkClient можно разделить на пять компонент, три основных и две вспомогательных. Основные компоненты:

1. Authorizator – компонента отвечающая за отправку идентификационных данных пользователя на сервер, и возврат ответа графическому интерфейсу.
2. RepoController – компонента отвечающая за управление пользовательскими репозиториями. Содержит функционал инициализации, создания, скачивания и удаления репозиториев.
3. RepoAgent - компонента, отвечающая за основной функционал работы с локальным репозиторием. Отвечает за такие команды как, включение пользовательских файлов в список отслеживаемых, фиксация изменений, переключение между версиями, загрузка изменений на удаленный сервер.

Дополнительные компоненты:

1. FileSystemControl – класс для работы с файловой системой.
2. ServerConnector – класс для взаимодействия с сервером.

При разработке данного модуля активно использовался шаблон проектирования фасад - структурный шаблон, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. На рисунке 3.4.1 представлена схема взаимодействия модуля LfkClient.



Рисунок 3.4.1 – Схема взаимодействия компонент модуля LfkClient

**Диаграммы, описание и реализация классов модуля**

При разработке данного модуля использовался паттерн проектирования «Фасад». Фасадными объектами являются классы: Repository, FileSystem. Класс Repository помимо функций RepoAgent и RepoController также содержит некоторые дополнительные функции управления репозиторием, что обеспечивает использование всего функционала системы контроля версий через один класс. На рисунке 3.4.2 изображена диаграмма основных классов.

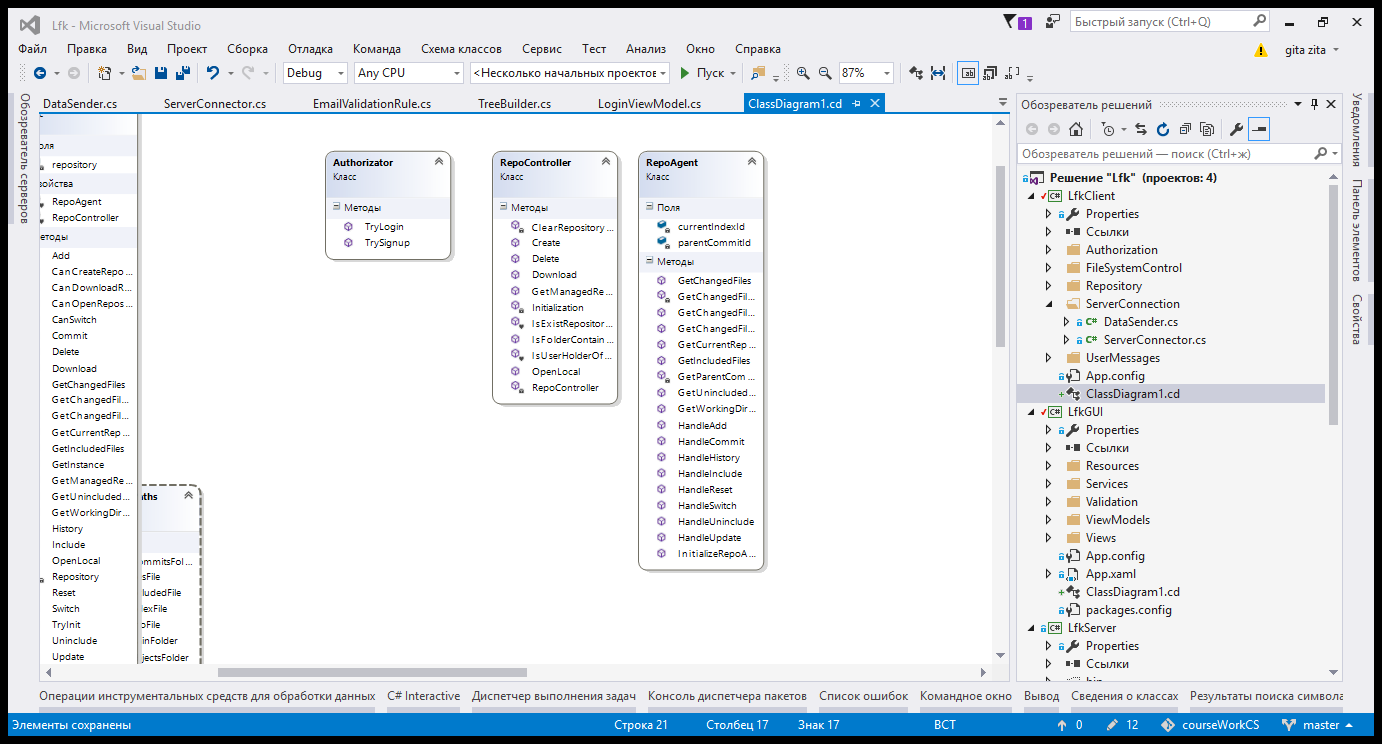


Рисунок 3.4.2 – Диаграмма основных классов модуля

Класс RepoController предоставляет функции работы с пользовательскими репозиториями, помимо этого у него есть ряд методов, проверяющих возможность выполнения той или иной команды. Результатом работы этих функция являются специально определенные перечисления, позволяющие точнее выявить причину невозможности выполнения действия. На рисунке 3.4.3 приведен пример функции создания пользовательского репозитория.

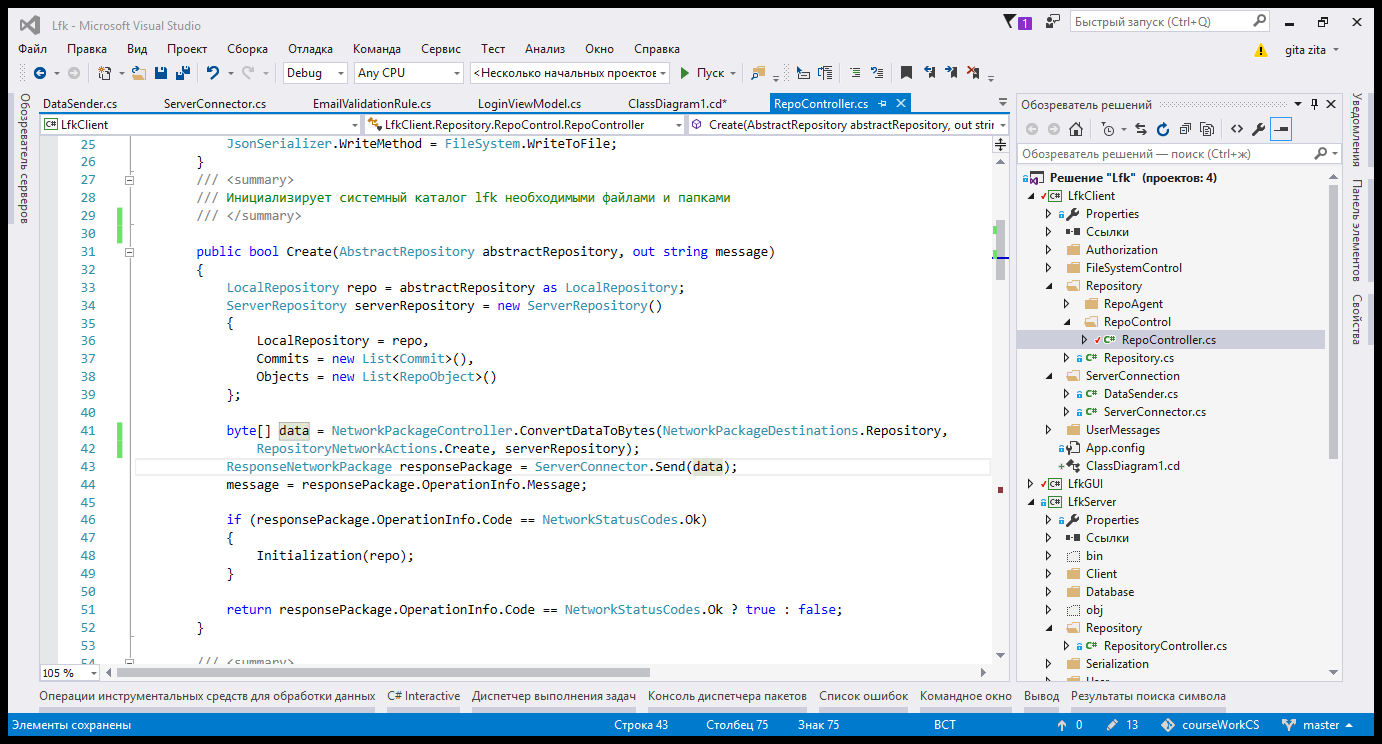


Рисунок 3.4.3 – Код реализации функции создания репозитория

Класс RepoAgent содержит в себе основной функционал системы контроля версий, также в нем определены методы, для получения информации о статусе пользовательских файлов и самого репозитория. На рисунке 3.4.4 приведен код реализации функции switch.

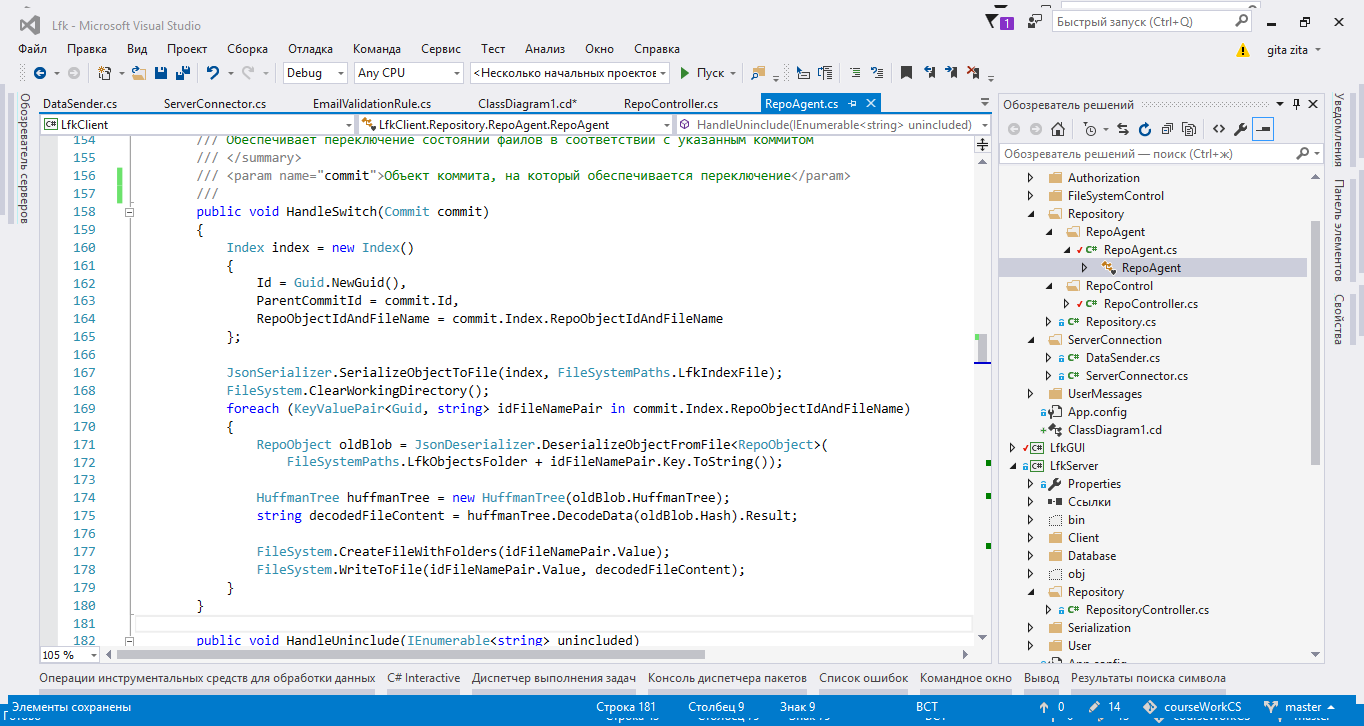


Рисунок 3.4.4 – Программная реализация команды switch.

LfkClient также включает в себя ряд вспомогательных классов, диаграмма которых представлена на рисунке 3.4.5.

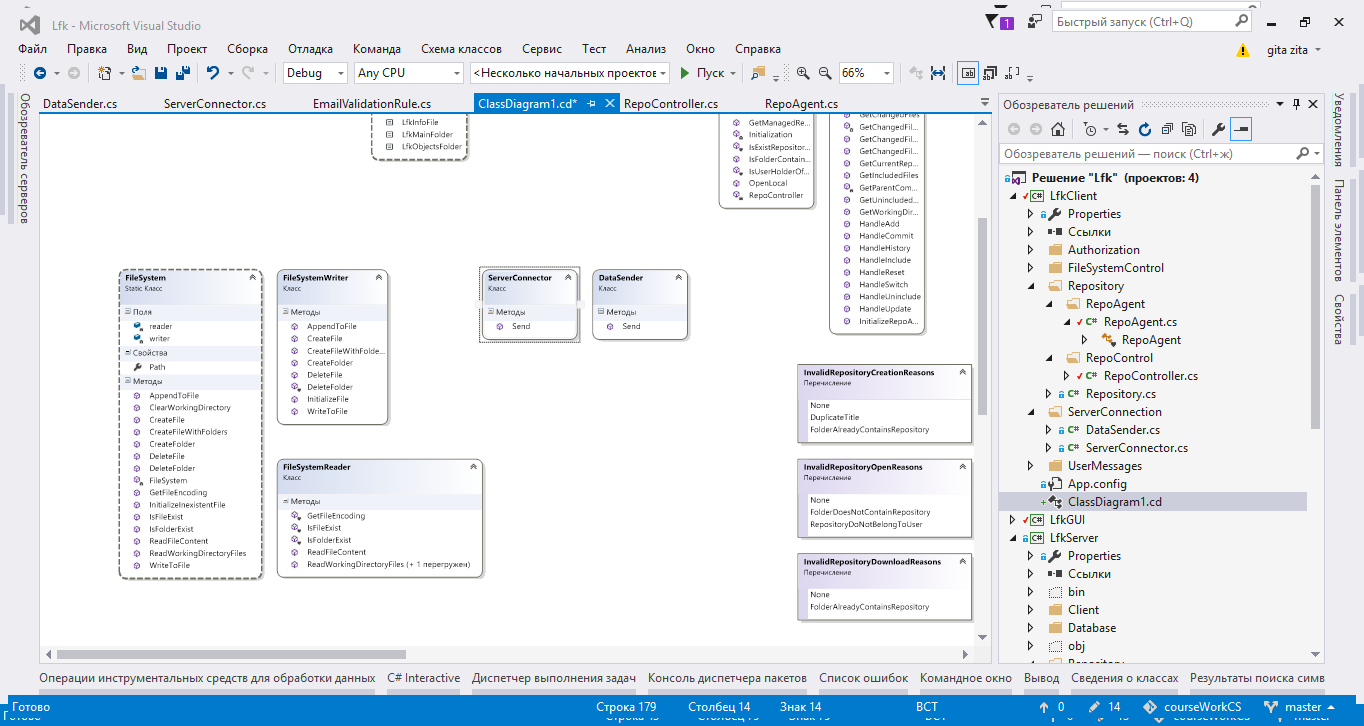


Рисунок 3.4.5 – Диаграмма вспомогательных классов.

Класс FileSystem инкапсулирует в себе всю работу с пользовательскими файлами и директориями, такой подход позволяет обрабатывать исключения ввода-вывода в одном месте и определить свою логику их обработки. Обязанности работы с файловой системой распределены между двумя классами FileSystemWriter и FileSystemReader, для записи и чтения соответственно. Класс DataSender содержит в себе один единственный метод, отправки пакета на сервер, и последующего считывания ответа. В случае возникновения неполадок на стороне сервера, вызывающим модулям будет отправлено соответствующее сообщение. На рисунке 3.4.6 представлена реализация этого класса.

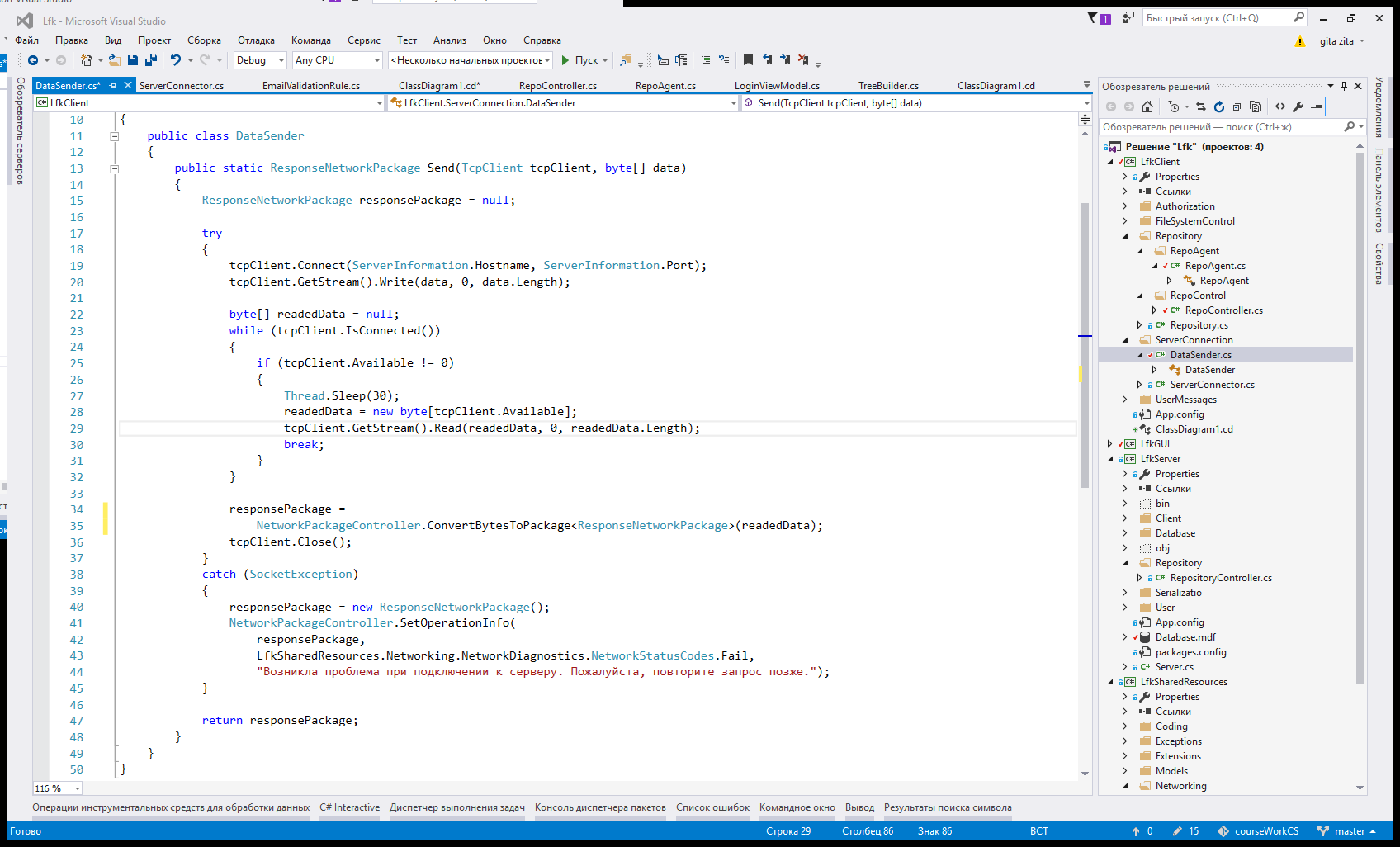


Рисунок 3.4.6 – Реализация метода соединения с сервером.

Взаимодействие с сервером осуществляется посредством использования класса NetworkPackage определенного в модуле LfkSharedResources. Соединение же устанавливается стандартным классом TcpClient.

## 3.5 Модуль LfkServer

Реализация : Контроллер

## 3.6 Проектирование базы данных

# **4. Руководство пользователя**

**Авторизация**

При запуске приложения пользователю предоставляется возможность войти с помощью e-mail и пароля, либо зарегистрировать новую учетную запись. На рисунке 4.1 представлено окно регистрации нового пользователя.

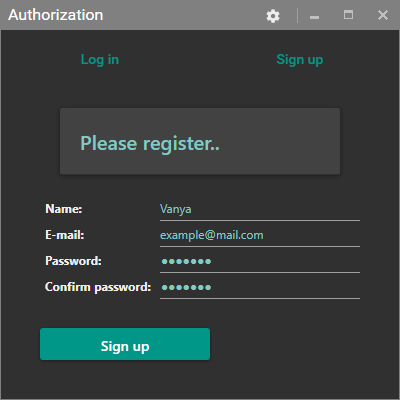


Рисунок 4.1 – Окно регистрации пользователя.

При регистрации новой учетной записи существует ряд правил, которым пользователь должен следовать:

1. Имя должно состоять только из латинских символов, включая верхний и нижний регистр. Минимальная длина имени 2 символа, максимальная 24.
2. E-mail адрес должен соответствовать принятым стандартам, что означает состоять из двух частей разделенных символом «@», причем в правой части в части доменного имени сервера должно быть минимум два символа.
3. Пароль должен содержать минимум один символ латиницы верхнего регистра, и одну или более цифр. Минимальная длина пароля 6 символа, максимальная - 24.

При отсутствии соединения с сервером, пользователь получит соответствующее сообщение, но уже не сможет использовать основной функционал приложения.

**Управление пользовательскими репозиториями**

В случае же успешной авторизации, пользователь перейдет к окну управления пользовательскими репозиториями. Все управляемые пользователем репозитории будут автоматически привязываться к текущему пользователю. На рисунке 4.2 изображено главное окно управления пользовательскими репозиториями.

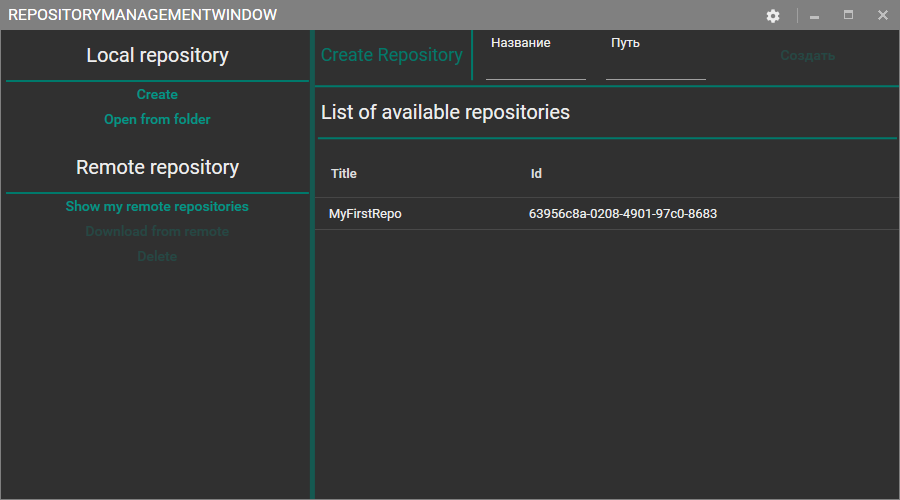


Рисунок 4.2 – Окно управления пользовательскими репозиториями.

При открытии пользователь видит в левой части приложения два меню, одно из которых предназначено для работы с локальными репозиториями, а другое позволяет управлять репозиториями которые находятся на сервере.

**Создание репозитория**

Для создания репозитория пользователь должен развернуть меню создания репозитория нажатием на пункт меню Создать. Далее необходимо ввести название и путь к существующей локальной директории, в которой будет располагаться репозиторий. Путь можно прописать в поле для ввода вручную, либо двойным щелчком левой клавиши мыши открыть диалоговое окно выбора директории.

При попытке создания репозитория в папке уже содержащей репозиторий, пользователю представиться выбор перезаписать репозиторий, либо выбрать другой каталог для создания репозитория.

**Открытие существующего репозитория**

Для открытия репозитория пользователь должен выбрать каталог, содержащий инициализационные файлы. В случае отсутствия в каталоге системных файлов приложение не может открыть каталог и возвращает управление окну управления пользовательскими репозиториями.

**Скачивание удаленного репозитория**

Для скачивания репозитория с сервера необходимо выбрать репозиторий из списка, и нажать кнопку скачать, после, приложение предложит пользователю выбрать каталог для размещения репозитория. В случае успешного скачивания, пользователь может отрыть данный репозиторий и начать с ним работать.

**Удаление репозитория**

Для удаления репозитория с сервера необходимо выбрать репозиторий из списка, и нажать кнопку удалить, после чего на сервере уничтожатся все данные связанные с этим репозиторием. Удалить локальный репозиторий можно просто удалив системную папку, созданную в рабочем каталоге.

**Работа с репозиторием**

При работе с репозиторием пользовательские файлы могу находится в одном из нескольких состояний:

* Не отслеживаемые файлы – файлы, которые являются новыми для системы контроля версий;
* Отслеживаемые файлы – файлы, изменения которых отслеживаются системой контроля версий;
* Добавленные для включения в следующий коммит файлы;

Изначально при создании репозитория, все пользовательские файлы являются не отслеживаемыми. Для включения файлов в список отслеживаемых необходимо использовать команду Include. Рабочая директория на странице команды Include представлена в виде иерархического дерева файлов и директорий. На рисунке 4.3 представлена страница включения файлов в список отслеживаемых.

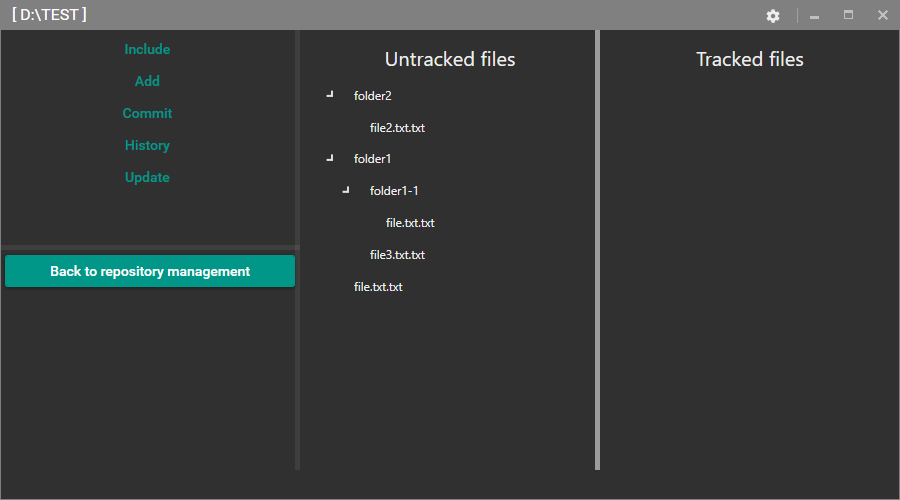


Рисунок 4.3 – Страница команды Include.

Добавление файлов в список отслеживаемых производится перетаскиванием любого узла в область страницы соответствующую отслеживаемым файлам. На рисунке 4.4 отображена страница команды Include после включения файла file.txt.txt.

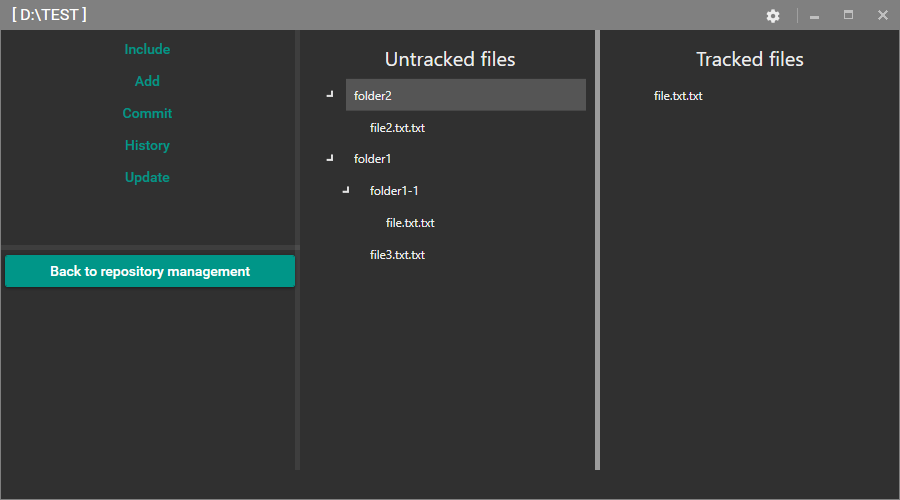


Рисунок 4.4 – Страница команды Include.

Для отмены отслеживания файла, необходимо перетащить его обратно в область не отслеживаемых файлов. Аналогичным образом производится выполнение команды Add.

Создание коммита происходит на вкладке Commit, которая изображена на рисунке 4.5. При создании коммита пользователю выводится список файлов, которые войдут в текущий коммит, также необходимо ввести сообщение коммита, отражающее суть произведенных изменений.

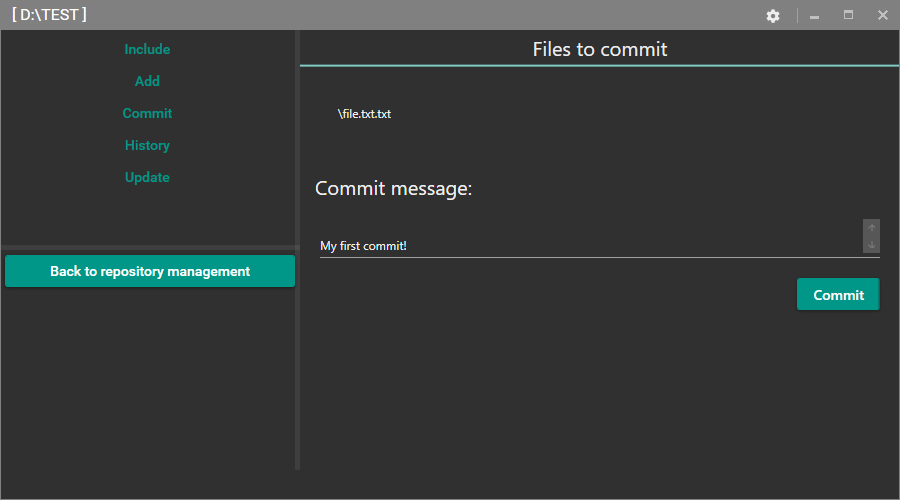


Рисунок 4.4 – Страница команды Commit.

Вкладка History позволяет пользователю просматривать историю коммитов. Также на вкладке History присутствует кнопка, позволяющая переключиться на любой коммит, в результате чего изменится состояние пользовательского рабочего каталога. На рисунке 4.5 изображена страница комманды History.

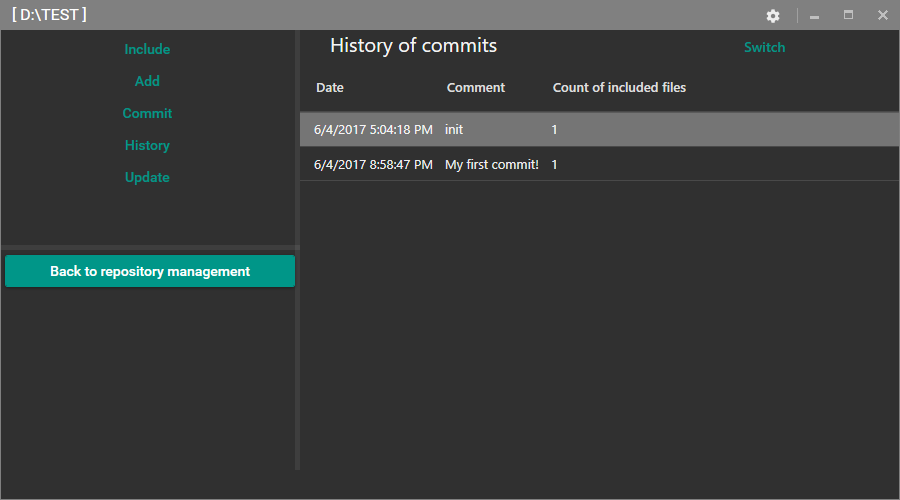


Рисунок 4.5 – Страница команды History.

Команда Update позволяет загрузить текущее состояние репозитория на сервер. Выполнение команды Update возможно лишь в случае если в репозитории создан хотя бы один коммит.

Также в приложении Lfk присутствует общее для всех окон меню настроек приложения, в котором можно переключить настройки стиля, и язык приложения. Данное меню продемонстрировано на рисунке 4.6.

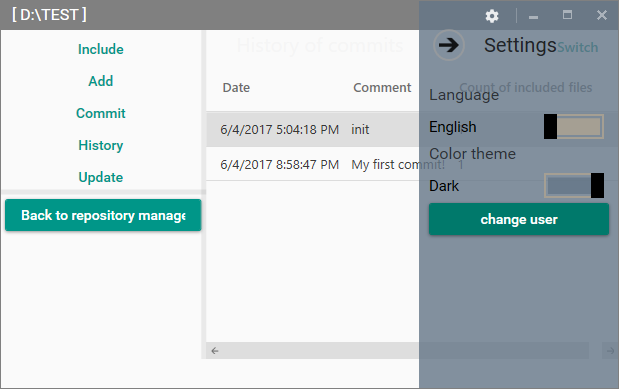


Рисунок 4.6 – Меню приложения Lfk.

Кнопка change user позволяет сменить пользователя, что реализовано путем перенаправления пользователя на страницу авторизации.

# **5. Тестирование**

## **5.1 Авторизация**

При запуске приложения Lfk, пользователю представляется окно авторизации, с возможностью входа, и регистрации. На этапе регистрации помимо некорректного заполнения полей данных о пользователе, возможна также ошибка дублирования пользователем с таким же e-mail. Данная ошибка будет обработана только на сервере, и ответ будет выведен в виде диалогового окна с сообщением, пример которого представлен на рисунке 5.1.1.

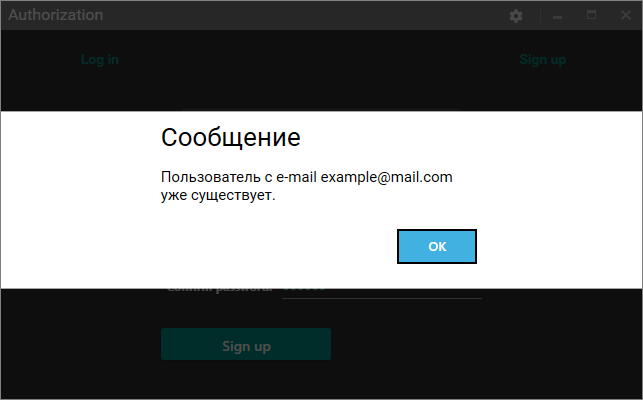


Рисунок 5.1.1 – Ошибка при регистрации.

В LfkGUI применяется валидация данных «на лету» что позволяет динамически сообщать пользователю о некорректно введенных данных. На рисунке 5.1.2 представлено окно регистрации в котором введены неверные данные, в силу этого блокируется возможность зарегистрировать пользователя.

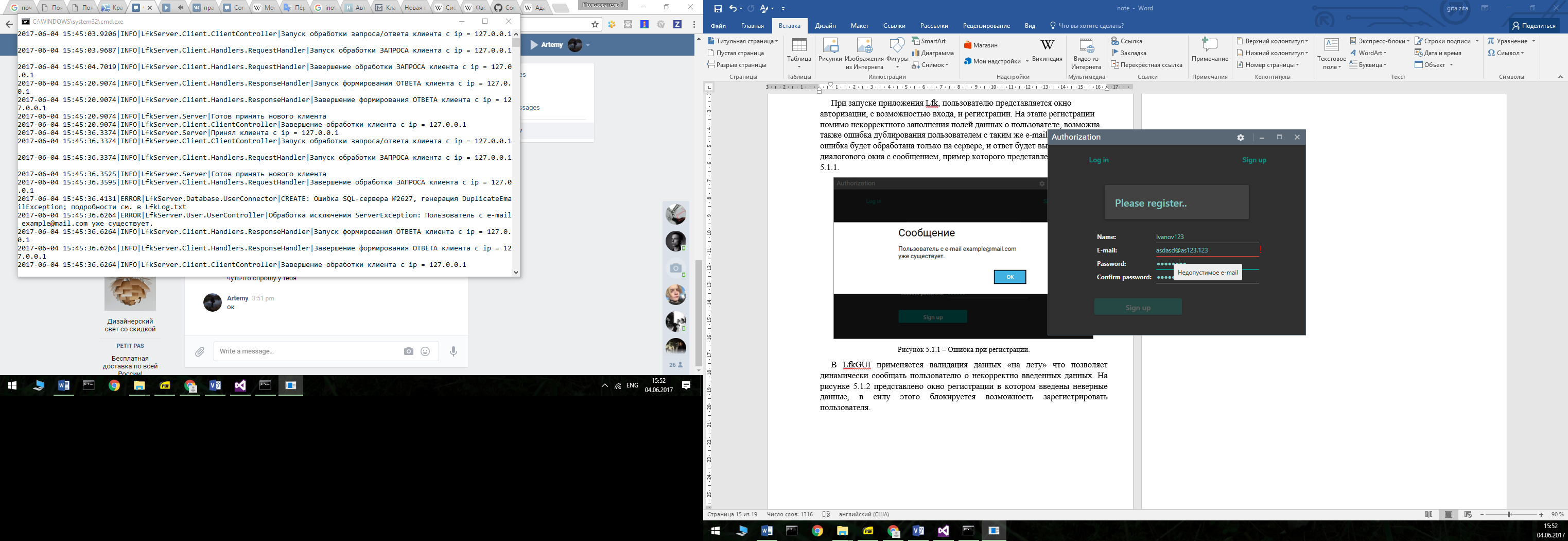


Рисунок 5.1.2 – Введение некорректных данных.

**5.2 Управление пользовательскими репозиториями**

Управление пользовательскими репозиториями осуществляется с помощью команд: открыть, скачать, показать все. Как видно на рисунке 5.2.1 возможность создать репозиторий будет блокирована, до тех пор, пока пользователь не введет корректные данные, также возможность удаления и скачивания блокируется пока пользователь не выберет репозиторий из списка.

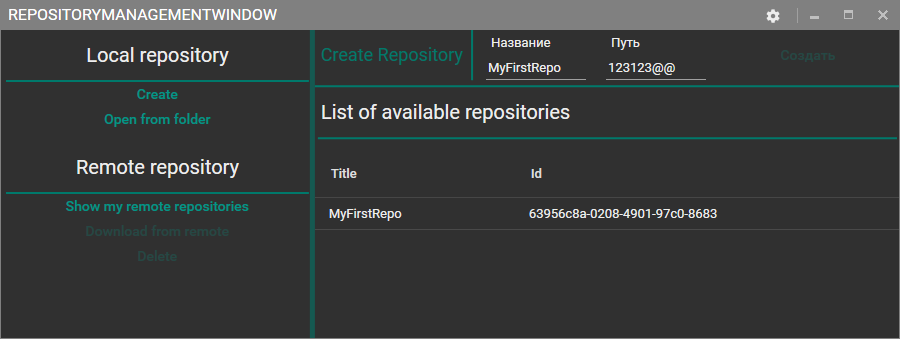


Рисунок 5.2.1 – Введение некорректных данных.

При попытке открыть репозиторий из папки которая не содержит скрытого каталога с системными и идентификационными файлами также будет выдано соответствующее сообщение, что показано на рисунке 5.2.2.

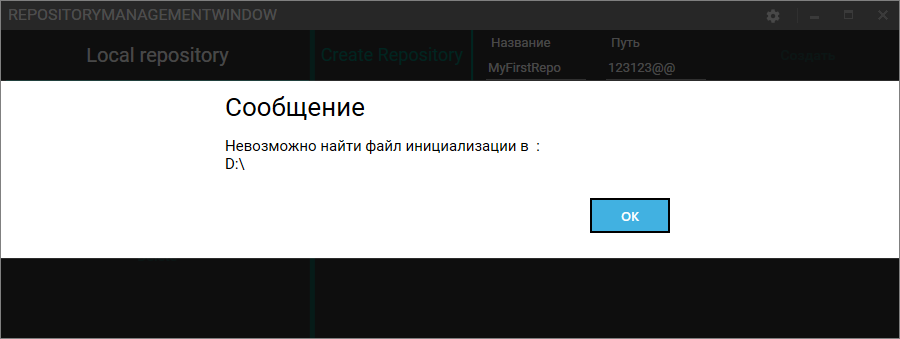


Рисунок 5.2.2 – Попытка открыть репозиторий из каталога, не содержащего репозиторий.

Также при попытке открыть репозиторий другого пользователя будет выведено сообщение о том, что репозиторий недоступен, данное сообщение представлено на рисунке 5.2.3.

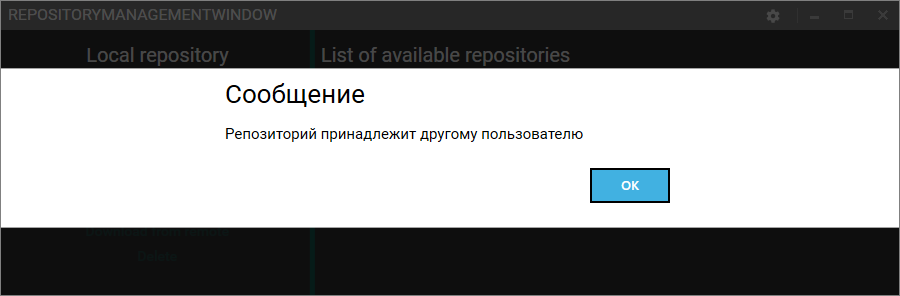


Рисунок 5.2.2 – Попытка открыть чужой репозиторий.

## **5.3 Контроль версий рабочей директории пользователя**

Пользователю запрещается делать коммит, если нет данных для фиксации, данный подход позволяет обезопасить приложение от непредвиденного поведения. Также при попытке загрузить репозиторий на сервер, при отсутствии каких-либо значимых данных пользователь увидит сообщение, представленное на рисунке 5.3.1.

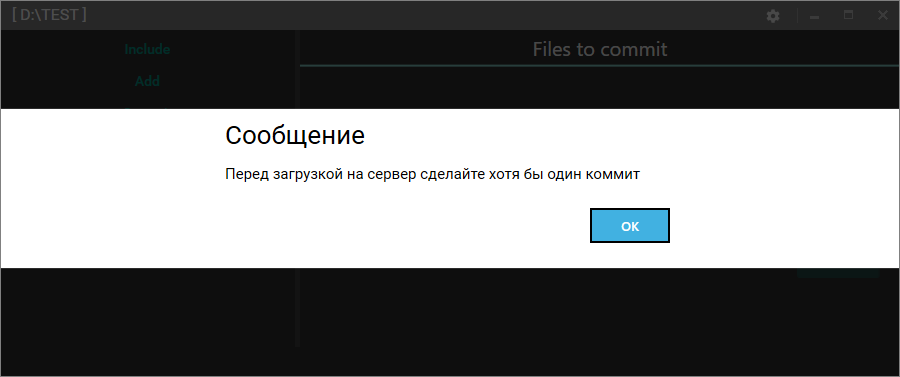


Рисунок 5.3.1 – Попытка загрузить пустой репозиторий.

## 5.4 Сервер

# 6. Заключение

# Список литераторы