РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОЙ СИСТЕМЫ МОНИТОРИНГА КУРСОВ ВАЛЮТ

Никитин Д.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Рябычина О.П. – канд.техн.наук, доцент, доцент кафедры ИРТ

В статье представлена информация о процессе разработки клиент-серверной системы, с применением практик написания качественного кода. Описаны варианты решения задач по реализации механизма кэширования данных и механизма взаимодействия с окнами в клиентском приложении. Спроектирована и разработана система мониторинга курсов валют на языке программирования C#.

При разработке программных продуктов много внимания уделяется распределению ресурсов вычислительной машины и функционала системы между подпрограммами. Одним из способов разделения функционала выступает архитектура клиент-сервер, такие архитектуры разделяют на виды за счет количества звеньев системы и распределения задач между ними.

В качестве модели для проектирования выбрана двухзвенная модель с тонким клиентом, однако если рассматривать систему как комплекс, а не как разрабатываемый продукт, то система состоит из трех звеньев – третье звено – это сервер банка, с которого поступает информация о курсе валют.

Такое распределение задач позволит повысить эффективность работы системы на клиентской стороне, что в свою очередь является одним из наиболее значимых критериев для конечного пользователя.

Для разработки программного средства выбран язык программирования C#, в качестве реализации модуля сервера выбрана форма «Web API», для клиентского приложения выбрана технология WPF.

Для получения курсов валют используется API Национального банка Республики Беларусь. Выбранное API позволяет получать информацию о курсе валют на конкретный день. Таким образом для того, чтобы получить данные за определенный период, нужно циклично выполнять запросы.

Одним из важнейших механизмов в таком типе задач выступает кэширование информации. На высоком уровне абстракции кэширование рассматривается как сохранение информации внутри определенной структуры данных, с возможностью дальнейшего обращения к ним. Также стоит учесть сохранение кэша при остановке работы программы, в качестве места хранения данных выбран текстовый файл в формате JSON.

При реализации механизма кэширования стоит уделить внимание сравнению двух возможных способов реализации кэширования:

1. Добавление в кэш только новой информации. Такой подход позволит избежать перезаписи данных в файл. Недостатком такого подхода могут быть хаотичные запросы пользователя относительно временно периода, т.е. пользователь может сделать поиск с интервалом в неделю и в таком случае чем больше будет дробление на части, тем больше придется сделать запросов к серверу или проверок внутри программы.

2. Добавление в кэш данных, если хотя бы одного объекта из запроса нет в кэше. Выбор такого подхода позволит упростить реализацию механизма сохранения информации, однако появится необходимость перезаписи файла.

В какой-то степени второй вариант может показаться более качественным, он позволяет упростить код и не зависит от имеющегося набора данных. Однако благодаря технологии LINQ появляется возможность упростить проверку данных для загрузки в кэш до вызова одного метода расширения Where. Такой подход позволяет избежать перебора данных с применением вложенных циклов.

Для реализации кэширования на программном уровне выбран шаблон проектирования «Заместитель» («Proxy»), он позволит объединить запросы к API банка и сам механизм кэширования, а на слое использования объекта прокси-класса не будет необходимости задумываться о кэшировании.

Серверная часть выступает в роли Web-API для клиентского приложения и содержит два GET запроса: запрос с определением начала и конца периода; запрос с заданием только начальной даты, в данном случае текущая дата выступает в роли конечного периода.

При разработке клиентского приложения выбрана технология WPF, одним из критериев выбора было наличие подхода модель-представление-модель представления при работе с данными (рисунок 1).

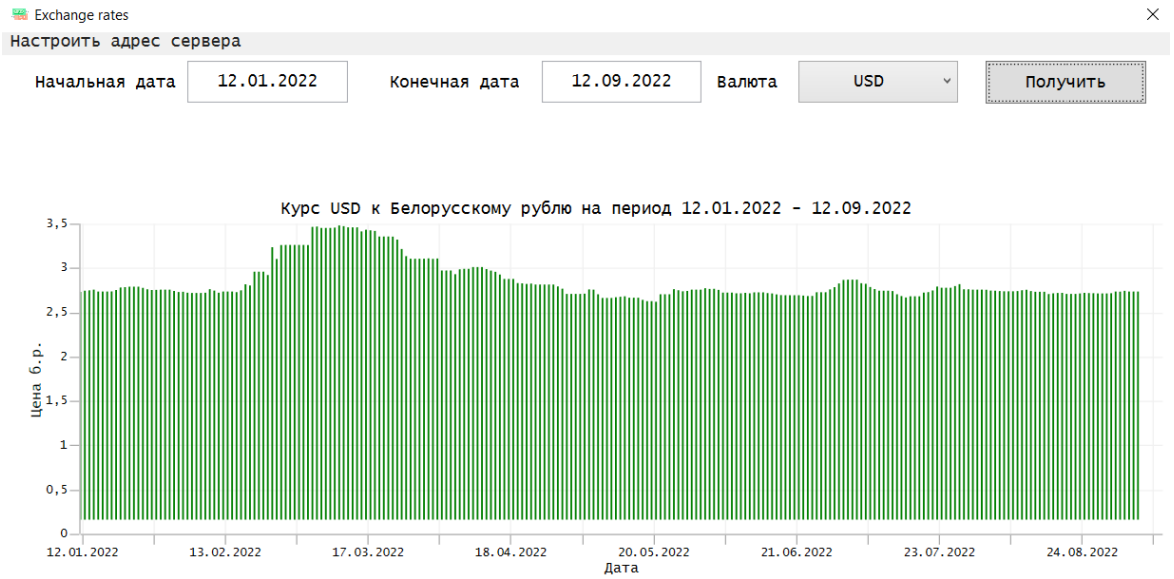

Рисунок 1 – Графический интерфейс клиентского модуля

За визуальное представление сведений о курсе валют отвечает бесплатная версия фреймворка «Syncfusion», она позволяет просто и гибко настроить вид диаграммы, включать и выключать необходимые функции.

Хорошей практикой будет вынесение процесса обращения к серверу в асинхронный метод, такое действие позволит предотвратить зависание пользовательского интерфейса, в совокупности с этим действием нужно блокировать элемент управления, который отвечает за отправку запроса, чтобы избежать наслаивания запросов. Важным является и предоставление информации пользователю о ходе формирования запроса, местом хранения этой информации выбран заголовок диаграммы.

Клиент-серверная архитектура подразумевает хранение и использование на стороне клиента адреса, на котором расположен серверный модуль в сети. Более гибким будет подход, заключающийся в добавлении нового окна с полем для ввода этого адреса.

В некоторых случаях открытие нескольких экземпляров одного окна нецелесообразно, а иногда и вовсе может сломать логику работы программы, например когда окно представляет собой изменение определенного набора данных, который должен быть изменен единожды. Решить эту проблему можно с помощью паттерна «Одиночка». Он позволит избежать повторного создания объекта, заменив его создание на получение уже существующего экземпляра.

В ходе исследования была рассмотрена важность разделения ресурсов в клиент-серверных системах и разработана система мониторинга курсов валют с применением такой архитектуры. Ознакомиться с детальной реализацией программы можно в репозитории GitHub «denden1s/Exchange-rates». Подход разделения системы на клиентскую и серверную часть позволил реализовать механизм кэширования без затрат ресурсов клиентского модуля.

Список использованных источников:

1. Клиент – сервер [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент_—_сервер. – Дата доступа: 27.02.2023.

2. Паттерн MVVM [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/22.1.php. – Дата доступа: 27.02.2023.