Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий

**К ЗАЩИТЕ ДОПУСТИТЬ**

Заведующий кафедрой ИИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.Н. Обабков

« 06 » июня 2022 г.

**РАЗРАБОТКА СИНХРОНИЗАЦИИ МОДЕЛИ ИЗ REVIT С ОНЛАЙН-СЕРВИСОМ ALTECINSOLATIONS.**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Пояснительная записка

Руководитель к.х.н., доцент \_\_\_\_\_\_\_\_\_ И.С. Стародубцев

Нормоконтролер: ст. преподаватель \_\_\_\_\_\_\_\_\_ В. Н. Васина

Обучающийся: группа РИ-480022 \_\_\_\_\_\_\_\_\_ Д. С. Соловьев

Екатеринбург – 2022**РЕФЕРАТ**

Выпускная квалификационная работа бакалавра 44 с., 6 рис., 12 источника, 2 прил.

Разработка синхронизации модели из Revit с онлайн-сервисом AltecInsolations.

Цель работы – реализация моментальной синхронизации модели из Revit с онлайн-сервисом AltecInsolations.

Объект разработки – онлайн сервис по расчету Инсоляции и KEO.

Результаты работы: реализован локальный сервер синхронизации модели из Revit с веб-сервисом AltecInsolations. Разработана служба обновления сервера. Подготовленно ядро для отправки данных из плагина Revit и приема их на клиенте в веб-сервисе. Все компоненты упакованы в инсталлятор.

Выпускная квалификационная работа выполнена в текстовом редакторе Microsoft Word.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc104751384)

[1 Проектирование системы 6](#_Toc104751385)

[1.1 Архитектура сервиса 6](#_Toc104751386)

[1.2 Сервер синхронизации 10](#_Toc104751387)

[1.3 Служба обновления 14](#_Toc104751388)

[1.4 Веб-сервис 15](#_Toc104751389)

[1.5 Экспортер – плагин Revit 16](#_Toc104751390)

[2 Реализация системы 18](#_Toc104751391)

[2.1 Разработка сервера синхронизации 18](#_Toc104751392)

[2.2 Разработка службы обновления 19](#_Toc104751393)

[2.3 Подготовка ядра для приема данных в сервисе 21](#_Toc104751394)

[2.4 Подготовка ядра для отправки данных из экспортера 22](#_Toc104751395)

[2.5 Создание инсталлятора 23](#_Toc104751396)

[ЗАКЛЮЧЕНИЕ 25](#_Toc104751397)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 26](#_Toc104751398)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 28](#_Toc104751399)

[ПРИЛОЖЕНИЕ А 30](#_Toc104751400)

[ПРИЛОЖЕНИЕ Б 37](#_Toc104751401)

# ВВЕДЕНИЕ

Строительство является одной из наиболее важных отраслей отечественной экономики, связанных как непосредственно с возведением зданий, так и с различной вспомогательной деятельностью.

Данная выпускная квалификационная работа является частью разработки сервиса AltecInsolations, разрабатываемого компанией Altec Systems, нацеленного на отображение информационных моделей зданий, расчет и визуализации норм инсоляции и КЕО.

[AltecInsolations](https://insolations.ru/) – онлайн сервис по расчету Инсоляции и KEO из BIM-моделей на отечественном ПО.

Инсоляция – достаточность попадания прямого солнечного света внутрь помещений или на участки местности для обеспечения комфортного проживания людей.

Коэффициент естественной освещенности (КЕО) — величина отношения световой энергии, попадающей в расчетную точку внутри помещения в конкретных условиях застройки, к величине световой энергии, которая попадала бы в расчетную точку в условиях, что та находилась бы под открытым небом. [1]

Программный комплекс предназначен для выполнения расчетов продолжительности инсоляции, уровня естественной освещенности, проверки выполнения рассчитанных значений нормативным требованиям и анализа затемняющих расчетную точку объектов.

Выгрузка модели из Revit в облачный сервис происходит с помощью плагина за 2–3 минуты. После чего по нажатию одной кнопки производятся расчеты и создается отчет.

Программный комплекс ориентирован на проектные компании. Расчет можно выполнять на любой стадии проектирования.

Кто использует решение:

* проектные компании;
* государственные экспертизы;
* коммерческие экспертизы.

Актуальность темы выпускной квалификационной работы заключается в том, что в реалиях активной и крупной застройки многие компании нуждаются в средствах автоматизации расчетов строительных норм для увеличения скорости и качества расчетов, сокращения рабочего времени сотрудников, уходящего на рутинные задачи.

Проблема заключается в отсутствие частичного обновления модели в сервисе. Например, пользователю при малейшем изменении своей модели приходилось полностью перезагружать её в сервис, частой практикой является то, что в модели присутствует сразу несколько объектов – зданий, и при изменении одного дома хотелось синхронизировать роботу в Revit и веб-сервисе.

Целью работы является реализация моментальной синхронизации модели из Revit с онлайн-сервисом AltecInsolations. Для достижения данной цели необходимо выполнить следующие задачи:

* реализовать отправку измененной части модели в плагине для Revit;
* разработать сервер синхронизации между плагином-экспортером и веб-сервисом AltecInsolations;
* реализовать прием данных в веб-сервис;
* реализовать службу обновления сервера синхронизации;
* создать инсталлятор.

# Проектирование системы

## Архитектура сервиса

Архитектура сервиса AltecInsolations представляет из себя сервер построенный по микросервисной архитектуре с файловой системой и двумя базами данных MongoDB и PostgreSQL для хранения информации о настройках, пользователях, информации о моделях и проектах проектировщиков и экспертов и т. д. Веб-сервис для расчета инсоляции и КЕО. И плагин – экспортер для Revit обрабатывающий данные и отправляющий их на сервер (рисунок 1).

Микросервисная архитектура — это метод построения легковесных приложений. При таком подходе приложение делится на множество независимых и слабосвязанных модулей (сервисов). Микросервисы поддерживают независимое развертывание и могут быть созданы на разных языках программирования и с применением разных технологий хранения данных. Микросервисы являются альтернативой монолитам. [7]

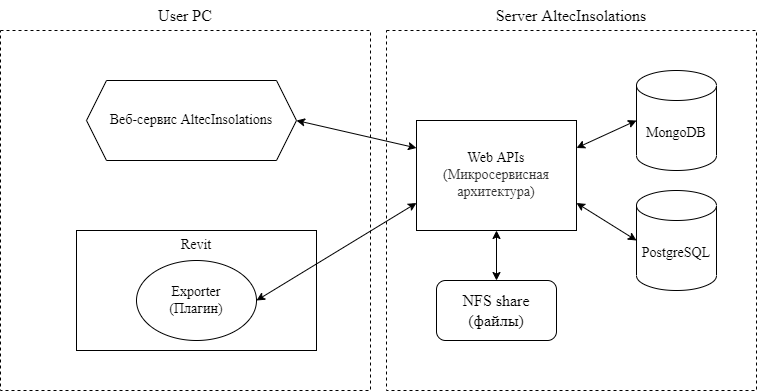


Рисунок 1 – Архитектура до использования сервера синхронизации

В результате анализа и рассмотрения множества вариантов стало ясно что у пользователя (проектировщика) может быть открыто несколько приложений Revit и несколько сцен нашего сервиса в браузере. Поэтому было принято решение о создании «моста» - сервера между плагином Revit и браузером пользователя.

Первоначально было решено развернуть ещё один web интерфейс, который принимал изменённые данные модели, и отправлял их обратно пользователю на сцену в браузере. Такой подход уже решал проблему, но у него были существенные недостатки. Во-первых, этот интерфейс был бы постоянно нагружен. Во-вторых, данные делали бы лишний «крюк» проходя через сервер и возвращаясь назад к пользователю. В-третьих, данные могли превышать 100 МБ, а это уже много в рамках нашего сервиса. Поэтому было решено поднять локальный сервер. Находясь на одном устройстве, данные передавались между экспортером и сервисом с огромной скоростью и не нагружали бы удалённый сервер.

Возникало сразу несколько вопросов. Как этот сервер обновлять? Какие права у него будут? Сможет ли он контролировать много подключений, как из плагина, так и из веб-сервиса?

Для обновления локального сервера было принято решение развернуть Windows службу. Так как для установки Revit требуются права администратора, то мы уверены в том, что у нашего пользователя они имеются, а значит и установка нашей службы и сервера будут с правами администратора (рисунок 2).

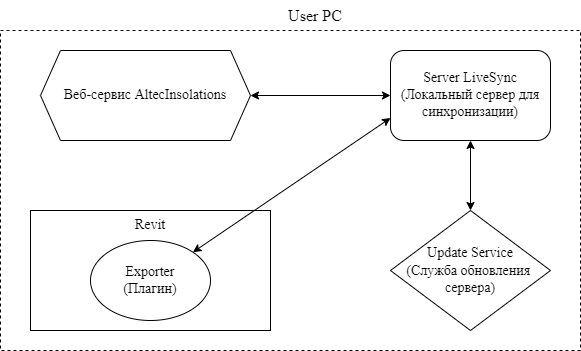


Рисунок 2 – Схема взаимодействия сервера синхронизации

В итоге получилось два развития пути, когда пользователь первый раз загружает модель и когда её изменят. В первом случае используется старый подход через прогон модели через удалённый сервер для полного сохранения её. Во втором случае измененная часть модели обрабатывается и через локальный сервер загружается прямо в браузер (рисунок 3).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Дорожная карта пользователя

Таким образом была использована комбинация этих двух подходов для достижения максимального результата, а также многократного увеличении скорости доставки данных.

## Сервер синхронизации

Задача заключалась в том, что данные необходимо было отправлять в реальном времени в браузер пользователю.

Для знакомых с web людей не является секретом, что протокол HTTP предполагает клиент-серверную архитектуру, в которой клиент всегда инициирует передачу данных, а сервер только отвечает на запросы клиентов. В режиме реального времени необходимо, чтобы сервер инициировал отправку данных.

Рассмотрим, какие есть решения этой проблемы.

* Polling;
* Long Polling;
* Server Sent Events (SSE);
* WebSocket.

Polling - клиент по таймеру опрашивает сервер: «А не появилось ли чего-нибудь новенького?» Это самый старый и прямолинейный способ организации real-time. Минусов у этого подхода больше, чем плюсов: нагрузка на сеть и сервер; данные приходят не в реальном времени, а с задержкой между наступлением события и отправкой данных.

Long Polling - клиент открывает соединение, а сервер держит его до наступления события, потом отправляет данные, после чего клиент переоткрывает соединение. Это уже настоящий real-time — нагрузка на сеть и сервер снижается. Но остается необходимость самостоятельно организовывать непрерывное соединение, следить за его обрывами и тем, чтобы передаваемые данные не потерялись в этот момент.

Server Sent Events (SSE) - поддерживаемая браузерами технология непрерывного HTTP-соединения, в котором данные передаются потоком от сервера к клиенту.

WebSocket - независимый протокол поверх TCP. Это самое современное и популярное решение задачи организации передачи данных между клиентом и сервером в реальном времени. [2]

Исходя из технологий выше было решено выбрать фреймворк SignalR.

ASP.NET Core SignalR — это библиотека с открытым исходным кодом, которая упрощает добавление веб-функций в режиме реального времени в приложения. Веб-функции в режиме реального времени позволяют коду на стороне сервера мгновенно отправлять содержимое на клиенты.

Хорошие кандидаты для SignalR:

* приложения, требующие частых обновлений с сервера. Примерами являются игры, социальные сети, платформы для голосования, аукционы, карты и приложения GPS;
* панели и приложения мониторинга. Примеры включают в себя панели мониторинга компании, мгновенные обновления продаж или оповещения о поездках;
* приложения для совместной работы. Примерами таких приложений служат демонстрационные приложения и программное обеспечение для конференций;
* приложения, которым требуются уведомления. Уведомления используют социальные сети, электронная почта, чат, игры, оповещения о поездках и многие другие приложения.

SignalR предоставляет API для создания [удаленных вызовов процедур (RPC) "](https://wikipedia.org/wiki/Remote_procedure_call)сервер-клиент". Вызов RPC вызывает функции на клиентах из кода .NET Core на стороне сервера. Существует несколько [поддерживаемых платформ](https://docs.microsoft.com/ru-ru/aspnet/core/signalr/supported-platforms?view=aspnetcore-6.0), каждая из которых имеет пакет SDK для клиента. Из-за этого вызов RPC вызывает различные языки программирования.

Ниже приведены некоторые функции SignalR для ASP.NET Core.

* управляет автоматическим управлением соединениями;
* отправляет сообщения всем подключенным клиентам одновременно. Например, комната чатов;
* отправляет сообщения конкретным клиентам или группам клиентов;
* масштабируется для управления увеличением трафика;
* источник размещается в SignalR репозитории на GitHub.

SignalR поддерживает следующие методы обработки обмена данными в режиме реального времени (в порядке правильного резерва):

* WebSockets;
* события, посылаемые сервером;
* длительный опрос.

SignalR автоматически выбирает лучший транспортный метод, который находится в пределах возможностей сервера и клиента.

SignalR использует концентраторы для взаимодействия между клиентами и серверами.

Концентратор — это высокоуровневый конвейер, который позволяет клиенту и серверу вызывать методы друг друга. SignalR автоматически обрабатывает диспетчеризации между компьютерами, позволяя клиентам вызывать методы на сервере и наоборот. Можно передать строго типизированные параметры методам, которые обеспечивают привязку модели. SignalR предоставляет два встроенных протокола концентратора: текстовый протокол на основе JSON и двоичный протокол на основе [MessagePack](https://msgpack.org/). MessagePack обычно создает меньшие сообщения по сравнению с JSON. Более старые браузеры должны поддерживать [XHR уровня 2](https://caniuse.com/#feat=xhr2) , чтобы обеспечить поддержку протокола MessagePack.

Концентраторы вызывают код на стороне клиента, отправляя сообщения, содержащие имя и параметры клиентского метода. Объекты, отправляемые как параметры метода, десериализованы с помощью настроенного протокола. Клиент пытается сопоставить имя с методом в коде на стороне клиента. Когда клиент находит совпадение, он вызывает метод и передает ему данные десериализованного параметра. [3]

ASP.NET Core является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету.

ASP.NET Core позволяет выполнять следующие задачи:

* создавать веб-приложения и службы, приложения Интернета вещей (IoT) и серверные части для мобильных приложений;
* использовать избранные средства разработки в Windows, macOS и Linux;
* выполнять развертывания в облаке или локальной среде;
* запускать в .NET Core. [4]

Таким образом было решено реализовать сервер синхронизации на языке программирования C# в среде разработки ASP.NET Core с использованием фреймворка SignalR. Что в итоге привело к снижению времени разработки, а также избавило от большого количества потенциальных багов и ошибок приложения.

## Служба обновления

Служба Windows — приложение, автоматически (если настроено) исполняемое системой при запуске операционной системы Windows и выполняющиеся вне зависимости от статуса пользователя. Имеет общие черты с концепцией демонов в Unix. [5]

Для взаимодействия со службой из экспортера, был выбран протокол TCP. С его помощью служба принимает команды и в зависимости от них управляет сервером.

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных [протоколов передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) интернета. Предназначен для управления [передачей данных интернета](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Пакеты в TCP называются сегментами.

В [стеке протоколов TCP/IP](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D0%BE%D0%B2_%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) выполняет функции [транспортного уровня](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D1%8B%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) [модели OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI).

Механизм TCP предоставляет [поток данных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым (в отличие от [UDP](https://ru.wikipedia.org/wiki/UDP)) целостность передаваемых данных и уведомление отправителя о результатах передачи.

Реализации TCP обычно встроены в [ядра](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Существуют реализации TCP, работающие в [пространстве пользователя](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F).

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, [браузером](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) и веб-сервером. TCP осуществляет надёжную передачу потока байтов от одного процесса к другому. [6]

В итоге служба обновления сервера синхронизации, благодаря TCP «слушателю», имеет потенциал к расширению, а также многофункциональному управлению из любого десктопного приложения. Что дает огромные возможности в будущем, и гибкое управление сервером в настоящем.

## Веб-сервис

В качестве языка для разработки веб-интерфейса AltecInsolations был выбран Typescript. В отличии от Javascript он обладает возможностью статической типизации, что позволяет отлавливать часть ошибок на этапе компиляции типов. Также в функционал языка входят модификаторы доступа, интерфейсы, дженерики, абстрактные классы и другие структуры, отсутствующие в Javascript, нацеленные на улучшение инкапсуляции, структурирование кода и упрощение рефакторинга и расширения функционала.

React — это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек JavaScript, предоставляющих полноценную платформу приложений, React ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса. Вы можете разместить отдельный компонент на веб-странице или вложить иерархии компонентов для создания сложного пользовательского интерфейса. [12]

SignalR использован в качестве создателя соединения между сервисом AltecInsolations и сервером синхронизации.

## Экспортер – плагин Revit

Autodesk Revit, или просто Revit — программный комплекс, реализующий принцип информационного моделирования зданий (Building Information Modeling, BIM). Предназначен для архитекторов, проектировщиков несущих конструкций и инженерных систем. Предоставляет возможности трехмерного моделирования элементов здания и плоского черчения элементов оформления, создания пользовательских объектов, организации совместной работы над проектом, начиная от концепции и заканчивая выпуском рабочих чертежей и спецификаций.

База данных Revit может содержать информацию о проекте на различных этапах жизненного цикла здания, от разработки концепции до строительства и снятия с эксплуатации.

Что такое API? API – это сокращение от Application Programming Interface (интерфейс программирования приложений). API является средством взаимодействия программиста с программным продуктом. Revit API, например, предоставляет программистам доступ к функциональным возможностям Revit и позволяет писать инструкции, которые Revit будет выполнять по порядку. Взглянем на процесс немного с другой стороны. Коммерческие компании-разработчики программного обеспечения, такие как Autodesk, часто предоставляют наборы библиотек, которые можно применять в пользовательских приложениях для взаимодействия с программными продуктами (Autodesk Revit и др.), расширяя таким образом их функциональность. Такие наборы библиотек называются API программных продуктов. Техника написания приложений, создаваемых для расширения функциональности программного продукта, зависит от структуры API и от перечня функций, доступ к которым он предоставляет. Что такое плагины?

Плагины – это программные модули или файлы, которые добавляют дополнительные функциональные возможности в целевой продукт (обычно в виде команд, автоматизирующих некоторые действия). Когда мы говорим о плагинах для Revit (для них применяются также термины «надстройка», «дополнение» и «подключаемый модуль»), мы имеем в виду модули, содержащие код, который взаимодействует с Revit API. Revit загружает плагины и использует их в определенных ситуациях – например, когда пользователь вызывает ту или иную команду. [9]

Плагины для Revit в основном разрабатываются на языке программирования C# на платформе .NET Framework. Так как это самый мощный и гибкий способ, позволяющий выходить за рамки родительского приложения (Autodesk Revit).

# Реализация системы

## Разработка сервера синхронизации

Сервер синхронизации состоит из трех компонентов:

* контроллера для получения сведений о соединении и отправки данных (ConnectorController);
* концентратора соединений (ConnectorHub);
* сервиса хранения соединений (ConnectorStorageService).

Когда открывается сцена с моделью в сервисе AltecInsolations, с помощью SignalR улетает запрос на соединение в концентратор сервера. Все нужные данные об этом соединении сохраняются в сервис хранения соединений.

Контроллер содержит два get запроса на получение всех соединений и какого-то одного конкретного соединения по имени проекта и один post запрос для отправки данных на открытую сцену веб-сервиса (рисунок 4).

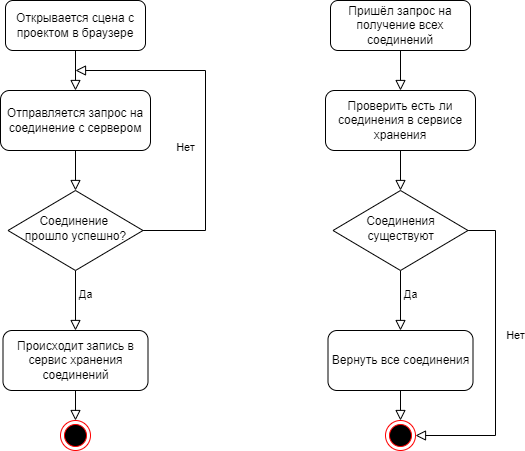
Когда приходит post запрос с данными и именем проекта, метод получает из сервиса соединений нужное, путём простого перебора всех элементов словаря, далее отправляет потоковой передачей данные прямиком в браузер. (Приложение А) 

Рисунок 4 – Диаграмма событий сервера синхронизации

## Разработка службы обновления

Служба обновления состоит из трех компонентов:

* сервиса загрузки, выгрузки сборки с сервером (AssemblyService);
* сервера TCP – «слушателя» TCP – клиентов (TcpServer);
* самой службы (Worker).

Было принято решение что служба будет сама запускать сервер подгружая его сборку себе в домен, таким образом мы избавляемся от лишних действий с правами на запуск и уменьшаем количество запущенных процессов. Так как сервис обновления плагина находится в самом себе, и уже был готов, на момент реализации сервера синхронизации, то было принято использовать его для замены dll файла (сборки). Так как Revit обладает ограниченными правами, а также его экземпляров может быть запущенно несколько, то запуск сервера полностью предоставлялся службе.

При запуске компьютера пользователя автоматически запускается служба, следующим шагом она подгружает и запускает сервер. Когда появляется новое обновление сервера, с экспортера приходит TCP сообщение о том, что требуется удалить из домена сборку, чтобы получить доступ на её изменение. После обновления сервера синхронизации он обратно подгружается в домен и запускается (рисунок 5).

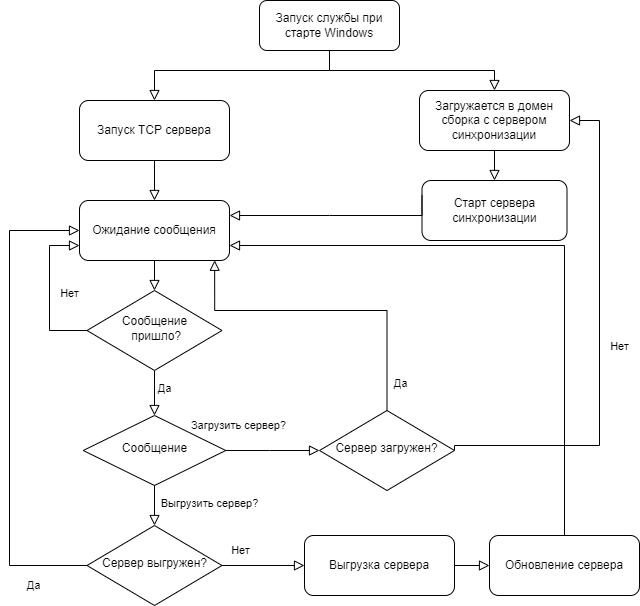
Запуск осуществляется с помощью рефлексии, а именно находится тип «Program», далее находится метод «StartServer», создаётся экземпляр класса этого типа и вызывается вышеупомянутый метод. (Приложение Б)

Рисунок 5 – Диаграмма событий службы обновления сервера

## Подготовка ядра для приема данных в сервисе

В веб-сервисе был реализован функционал подключения к серверу при открытии сцены, а также прием данных.

Схема работы довольно проста. Сначала происходит соединение с сервером, за счет концентратора SignalR, по нему отправляется информация о открытом проекте, которая в последующем потребуется для того, чтобы отправить данные в нужную вкладку или браузер. Далее это соединение остаётся открытым до тех пор, пока не придет измененная часть модели. В последующем и нужно будет обработать и загрузить в 3D движок Babylon.js.

## Подготовка ядра для отправки данных из экспортера

В плагине – экспортере был разработан и внедрён сервис, отправляющий данные на сервер синхронизации, а также команды в службу обновления сервера, такие как загрузка, выгрузка сборки, и запуск, остановка сервера.

Запрос с данными отправляется по HTTP протоколу.

HTTP (HyperText Transfer Protocol — протокол передачи гипертекста) — символьно-ориентированный клиент-серверный протокол прикладного уровня без сохранения состояния, используемый сервисом World Wide Web.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier – уникальный идентификатор ресурса) в запросе клиента. Основными ресурсами являются хранящиеся на сервере файлы, но ими могут быть и другие логические (напр. каталог на сервере) или абстрактные объекты (напр. ISBN). Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: mime-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым. [11]

Для того чтобы измененная модель была доставлена по нужному адресу требуется всего лишь знать имя проекта, которое экспортер изначально получает с удаленного сервера AltecInsolations.

На рисунке 6 можно видеть полную схему взаимодействия компонентов синхронизации.

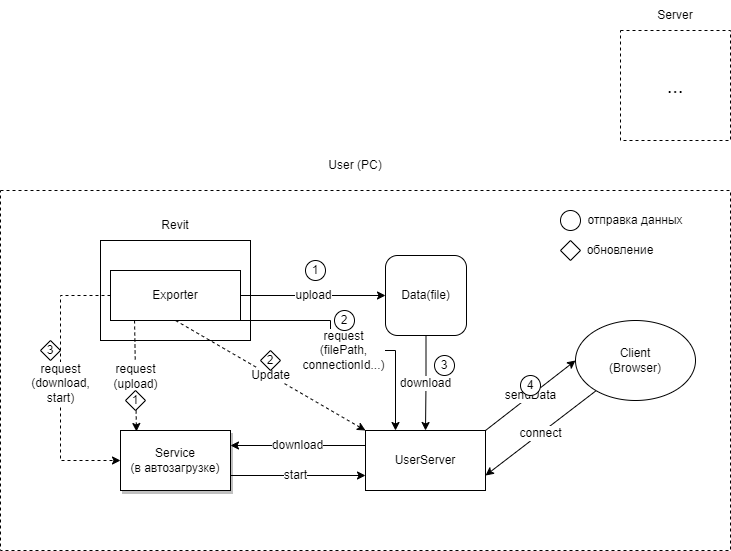


Рисунок 6 – Полная схема взаимодействия с сервером синхронизации

## Создание инсталлятора

После завершения разработки всех компонентов синхронизации, требовалось их упаковать в инсталлятор под операционную систему Windows. Выбор упал на Advanced Installer.

Advanced Installer — это мощное и простое в использовании средство для создания программ установок в формате MSI / EXE / CAB за считанные минуты. Решение Advanced Installer основывается на использовании стандартных открытых форматов. Файлы проектов сохраняются в формате XML, что упрощает их интеграцию с системами управления исходными текстами. Advanced Installer - профессиональный инструмент для создания MSI-установщиков приложений под операционные системы Windows. [8]

Существует такая практика, что у проектировщика на компьютере установлено несколько версий Revit. Чтобы установить плагин требуется положить файл с расширением addin в папку с соответствующей версией «C:\Users\[User]\AppData\Roaming\Autodesk\Revit\Addins\[Version]».

Появилась задача при установке предоставить пользователю выбор в какие версии Revit установить экспортер. Для этого был написан скрипт на языке VBS, который обнаруживал установленные версии Revit.

VBScript (VBS, развернуто Microsoft Visual Basic Script Edition, иногда Visual Basic Script) — язык сценариев, созданный компанией Microsoft на основе языка Visual Basic, предназначенный для применения в приложениях, использующих технологию Active Scripting. [10]

# ЗАКЛЮЧЕНИЕ

В процессе работы цель была достигнута. Реализован локальный сервер синхронизации модели из Revit с веб-сервисом AltecInsolations. Разработана служба обновления сервера, способная полностью управлять им, а именно подгружать его в домен, запускать, останавливать и удалять. Подготовленно ядро для отправки данных из плагина Revit и приема их на клиенте в веб-сервисе. Все компоненты упакованы в инсталлятор.

В результате было получено многократное ускорение при отправке модели Revit после её изменения. И заложен фундамент под формирование новых улучшение за счёт поднятого изменяемого локального сервера.

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

[AltecInsolations](https://insolations.ru/) – онлайн сервис по расчету Инсоляции и KEO.

КЕО – коэффициент естественной освещенности.

BIM – сокращение от Building Information Model или Modeling – информационная модель (или моделирование) [зданий](https://ru.wikipedia.org/wiki/%D0%97%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5) и сооружений

ПО – программное обеспечение.

Revit – программный комплекс для автоматизированного проектирования.

MongoDB – это документоориентированная база данных типа NoSQL. PostgreSQL – свободная объектно-реляционная система управления базами данных (СУБД).

HTTP (HyperText Transfer Protocol – «протокол передачи [гипертекста](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82)») — [протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [прикладного уровня](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) передачи данных.

TCP – сокращение от Transmission Control Protocol (протокол управления передачей), один из основных [протоколов передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) интернета.

ASP.NET (Active Server Pages для .NET) – платформа разработки [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5).

SignalR – это библиотека для ASP.NET разработчиков, которая упрощает процесс добавления веб-функций в режиме реального времени к приложениям.

Концентратор – это высокоуровневый конвейер, который позволяет клиенту и серверу вызывать методы друг друга.

URI – сокращение от Uniform Resource Identifier (уникальный идентификатор ресурса).

API – сокращение от Application Programming Interface (интерфейс программирования приложений).

React – это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов.

JSON – текстовый формат обмена данными, основанный на JavaScript.

VBScript (VBS, развернуто Microsoft Visual Basic Script Edition, иногда Visual Basic Script) — язык сценариев, созданный компанией Microsoft на основе языка Visual Basic.

Babylon.js – простой в использовании движок трехмерной (3D) графики на базе WebGL.

Advanced Installer — это мощное и простое в использовании средство для создания программ установок.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Расчет инсоляции и КЕО (коэффициента естественной освещенности) [Электронный ресурс] // iD группа https://idgruppa.ru/services/insolyacia-keo/ (Дата обращения – 21.05.2022)
2. Вам посылка, или как мы доставляем сообщения с сервера на клиент в реальном времени [Электронный ресурс] // Хабр https://habr.com/ru/company/superjob/blog/649053/ (Дата обращения – 20.05.2022)
3. Общие сведения об ASP.NET Core SignalR [Электронный ресурс] // Microsoft Docs https://docs.microsoft.com/ru-ru/aspnet/core/signalr/introduction?view=aspnetcore-6.0 (Дата обращения – 20.05.2022)
4. Общие сведения об ASP.NET Core [Электронный ресурс] // Microsoft Docs https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0 (Дата обращения – 20.05.2022)
5. Служба Windows [Электронный ресурс] // Википедия https://ru.wikipedia.org/w/index.phptitle=Служба\_Windows&stable=1 (Дата обращения – 20.05.2022)
6. Transmission Control Protocol [Электронный ресурс] // Википедия https://ru.wikipedia.org/wiki/Transmission\_Control\_Protocol (Дата обращения – 20.05.2022)
7. Микросервисная архитектура от А до Я [Электронный ресурс] // BellSoft https://libericajdk.ru/announcements/2021/12/02/microservices-101/ (Дата обращения – 20.05.2022)
8. Advanced Installer [Электронный ресурс] // 4PDA https://4pda.to/forum/index.php?showtopic=184907&st=20 (Дата обращения – 20.05.2022)
9. Урок 1: Общее представление о программе [Электронный ресурс] // Autodesk https://damassets.autodesk.net/content/dam/autodesk/images/ru-adn/lesson\_1.pdf (Дата обращения – 20.05.2022)
10. VBScript [Электронный ресурс] // Википедия https://ru.wikipedia.org/wiki/VBScript (Дата обращения – 21.05.2022)
11. Протокол HTTP — Документация по Веб-программированию 0.0.0 [Электронный ресурс] // Информационные системы и технологии УрФУ ИРИТ-РТФ https://iit-web-lectures.readthedocs.io/ru/latest/www/http.html (Дата обращения – 21.05.2022)
12. React в Windows [Электронный ресурс] // Microsoft Docs https://docs.microsoft.com/ru-ru/windows/dev-environment/javascript/react-overview (Дата обращения – 21.05.2022)

# ПРИЛОЖЕНИЕ А

ConnectorController.cs

**using** **Exporter.LiveSync.Interfaces**;

**using** **Exporter.LiveSync.Models**;

**using** **Microsoft.AspNetCore.Mvc**;

**using** **Microsoft.AspNetCore.SignalR**;

**namespace** **Exporter.LiveSync.Controllers**;

/// <summary>

/// Контроллер для получения сведений о соединении и отправки данных.

/// </summary>

[Route("api/[controller]")]

[ApiController]

**public** **class** **ConnectorController** : ControllerBase

{

**private** **readonly** IConnectionStorageService \_connectionStorage;

**private** **readonly** IHubContext<ConnectorHub> \_hubContext;

**private** **readonly** ILogger<ConnectorController> \_logger;

**public** **ConnectorController**(ILogger<ConnectorController> logger,

IConnectionStorageService connectionStorage,

IHubContext<ConnectorHub> hubContext)

{

\_logger = logger;

\_connectionStorage = connectionStorage;

\_hubContext = hubContext;

}

[HttpGet("GetConnections")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

**public** **async** Task<IActionResult> Get()

{

**try**

{

**var** connections = \_connectionStorage.GetConnections();

\_logger.LogInformation("Connections successfully received");

**return** **Ok**(connections);

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**return** **BadRequest**(ex.Message);

}

}

[HttpGet("{projectName}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

**public** **async** Task<IActionResult> Get(**string** projectName)

{

**try**

{

**if** (**string**.IsNullOrEmpty(projectName))

{

\_logger.LogError("Invalid project name");

**return** **BadRequest**("Invalid project name");

}

**var** connection = \_connectionStorage.GetConnection(projectName);

\_logger.LogInformation("Connections successfully received");

**return** **Ok**(connection);

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**return** **BadRequest**(ex.Message);

}

}

[HttpGet("connect")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

**public** **async** Task<IActionResult> Connect(Connection connection)

{

**try**

{

**if** (!\_connectionStorage.IsConnectionExist(connection.ConnectionId))

{

\_logger.LogInformation("Project {ProjectName} not found", connection.ProjectName);

**return** **NotFound**(connection);

}

\_logger.LogInformation("Project {ProjectName} exist", connection.ProjectName);

**await** \_hubContext.Clients.Client(connection.ConnectionId).SendAsync("Connect");

**return** **Ok**();

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**return** **BadRequest**(ex.Message);

}

}

[HttpPost]

[RequestSizeLimit(180L \* 1024L \* 1024L)]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

**public** **async** Task<IActionResult> Post([FromForm] **string** projectName, [FromForm] IFormFile file)

{

**try**

{

**if** (**string**.IsNullOrEmpty(projectName))

{

\_logger.LogError("Invalid project name");

**return** **BadRequest**("Invalid project name");

}

**var** connection = \_connectionStorage.GetConnection(projectName);

**if** (connection **is** **null**)

{

\_logger.LogError("Connection was null");

**return** **BadRequest**("Connection was null");

}

**if** (file.Length == **0**)

{

\_logger.LogWarning("File length was null");

**return** **BadRequest**("Incorrect file size");

}

**var** data = **new** MemoryStream();

\_logger.LogInformation("Start copying data");

**await** file.CopyToAsync(data);

\_logger.LogInformation("Copied successfully, start transfer");

**await** \_hubContext.Clients.Client(connection.ConnectionId).SendAsync("Data", data.ToArray());

\_logger.LogInformation("File transferred successfully");

**return** **Ok**();

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**return** **BadRequest**(ex.Message);

}

}

}

ConnectorStorageService.cs

**using** **Exporter.LiveSync.Interfaces**;

**using** **Exporter.LiveSync.Models**;

**namespace** **Exporter.LiveSync.Services**;

**public** **class** **ConnectionStorageService** : IConnectionStorageService

{

**private** **readonly** Dictionary<**string**, Connection> \_connections;

**private** **readonly** ILogger<ConnectionStorageService> \_logger;

**public** **ConnectionStorageService**(ILogger<ConnectionStorageService> logger)

{

\_connections = **new** Dictionary<**string**, Connection>();

\_logger = logger;

}

**public** **void** **DeleteConnection**(**string** connectionId)

{

**try**

{

**if** (\_connections.Remove(connectionId))

{

\_logger.LogInformation("Connection {ConnectionId} deleted successfully", connectionId);

**return**;

}

\_logger.LogInformation("Connection {ConnectionId} not found", connectionId);

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**throw**;

}

}

**public** IEnumerable<Connection> GetConnections()

{

**try**

{

**return** \_connections.Values;

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**throw**;

}

}

**public** Connection? GetConnection(**string** projectName)

{

**try**

{

**var** connection = \_connections.Values.FirstOrDefault(x => x.ProjectName == projectName);

**if** (connection **is** not **null**)

{

\_logger.LogInformation("Connection with project name {ProjectName} is successfully received.", projectName);

}

**return** connection;

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**throw**;

}

}

**public** **bool** **IsConnectionExist**(**string** connectionId)

{

**return** \_connections.ContainsKey(connectionId);

}

**public** **void** **AddConnection**(Connection connection)

{

**try**

{

**var** connectionId = connection.ConnectionId;

**if** (!\_connections.ContainsKey(connectionId))

{

\_connections[connectionId] = connection;

\_logger.LogInformation("Connection with project name {ProjectName} successfully added",

connection.ProjectName);

**return**;

}

\_logger.LogInformation("Connection with project name {ProjectName} already exists",

connection.ProjectName);

}

**catch** (Exception ex)

{

\_logger.LogError(ex, ex.Message);

**throw**;

}

}

}

ConnectorHub.cs

**using** **Exporter.LiveSync.Interfaces**;

**using** **Exporter.LiveSync.Models**;

**using** **Microsoft.AspNetCore.SignalR**;

**namespace** **Exporter.LiveSync**;

**public** **class** **ConnectorHub** : Hub

{

**private** **readonly** IConnectionStorageService \_connectionStorage;

**private** **readonly** ILogger<ConnectorHub> \_logger;

**public** **ConnectorHub**(ILogger<ConnectorHub> logger, IConnectionStorageService connectionStorage)

{

\_connectionStorage = connectionStorage;

\_logger = logger;

}

**public** **async** Task **Add**(Connection connection)

{

\_connectionStorage.AddConnection(connection);

}

**public** **override** **async** Task **OnConnectedAsync**()

{

\_logger.LogInformation("Connected Id: {Id}", Context.ConnectionId);

**await** **base**.OnConnectedAsync();

}

**public** **override** **async** Task **OnDisconnectedAsync**(Exception exception)

{

\_connectionStorage.DeleteConnection(Context.ConnectionId);

**await** **base**.OnDisconnectedAsync(exception);

}

}

Program.cs

**using** **Exporter.LiveSync.Controllers**;

**using** **Exporter.LiveSync.Interfaces**;

**using** **Exporter.LiveSync.Services**;

**using** **Microsoft.AspNetCore.Http.Features**;

**namespace** **Exporter.LiveSync**;

**public** **class** **Program**

{

**public** **void** **StartServer**(CancellationToken stoppingToken)

{

**var** assembly = **typeof**(ConnectorController).Assembly;

**var** builder = WebApplication.CreateBuilder();

builder.Services.AddControllers().AddApplicationPart(assembly).AddControllersAsServices();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddSingleton<IConnectionStorageService, ConnectionStorageService>();

builder.Services.AddSignalR();

builder.Services.Configure<FormOptions>(formOptions =>

{

formOptions.ValueLengthLimit = **int**.MaxValue;

formOptions.MultipartBodyLengthLimit = **int**.MaxValue;

});

builder.Services.AddCors(config =>

{

config.AddPolicy("DefaultPolicy",

policyBuilder => policyBuilder.WithOrigins("https://localhost:3000")

.AllowAnyMethod()

.AllowAnyHeader()

.AllowCredentials());

});

**var** app = builder.Build();

**if** (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors("DefaultPolicy");

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.UseEndpoints(endpoints =>

{

endpoints.MapHub<ConnectorHub>("/hub");

});

app.RunAsync(stoppingToken);

}

}

Connection.cs

**namespace** **Exporter.LiveSync.Models**;

**public** **class** **Connection**

{

**public** **string** ProjectName { **get**; **set**; }

**public** **string** ConnectionId { **get**; **set**; }

}

IConnectionsStorageService.cs

**using** **Exporter.LiveSync.Models**;

**namespace** **Exporter.LiveSync.Interfaces**;

**public** **interface** IConnectionStorageService

{

**bool** **IsConnectionExist**(**string** connectionId);

**void** **AddConnection**(Connection connection);

IEnumerable<Connection> GetConnections();

Connection? GetConnection(**string** projectName);

**void** **DeleteConnection**(**string** connectionId);

}

# ПРИЛОЖЕНИЕ Б

TcpServer.cs

**using** **System.Net**;

**using** **System.Net.Sockets**;

**using** **System.Text**;

**using** **Exporter.Service.Interfaces**;

**namespace** **Exporter.Service.Services**;

**public** **class** **TcpServer** : ITcpServer

{

**private** **const** **int** Port = **8888**;

**private** **readonly** IAssemblyService \_assemblyService;

**private** **readonly** ILogger<TcpServer> \_logger;

**public** **TcpServer**(ILogger<TcpServer> logger, IAssemblyService assemblyService)

{

\_logger = logger;

\_assemblyService = assemblyService;

}

**public** **void** **Start**()

{

TcpListener? server = **null**;

**try**

{

**var** localAddress = IPAddress.Parse("127.0.0.1");

server = **new** TcpListener(localAddress, Port);

server.Start();

**while** (**true**)

{

\_logger.LogInformation("Ожидание подключений... ");

**var** client = server.AcceptTcpClient();

\_logger.LogInformation("Подключен клиент. Выполнение запроса...");

**var** data = **new** **byte**[**256**];

**var** response = **new** StringBuilder();

**var** stream = client.GetStream();

**do**

{

**var** bytes = stream.Read(data, **0**, data.Length);

response.Append(Encoding.UTF8.GetString(data, **0**, bytes));

} **while** (stream.DataAvailable);

**var** command = response.ToString();

**try**

{

\_logger.LogInformation("Response (command) = {Command}", command);

**switch** (command)

{

**case** **nameof**(\_assemblyService.LoadAssembly):

\_assemblyService.LoadAssembly();

**break**;

**case** **nameof**(\_assemblyService.UnloadAssembly):

\_assemblyService.UnloadAssembly();

**break**;

**default**:

\_logger.LogInformation("Command:{Command} not found", command);

**break**;

}

}

**catch** (Exception ex)

{

\_logger.LogError(ex.Message);

}

stream.Close();

client.Close();

}

}

**catch** (Exception ex)

{

\_logger.LogError(ex.Message);

}

**finally**

{

server?.Stop();

}

}

}

AssemblyService.cs

**using** **Exporter.Service.Interfaces**;

**namespace** **Exporter.Service.Services**;

**public** **class** **AssemblyService** : IAssemblyService

{

**private** **const** **string** AssemblyFilePath = "Server/Exporter.LiveSync.dll";

**private** **readonly** ILogger<AssemblyService> \_logger;

**private** CancellationTokenSource \_cancelTokenSource = **new**();

**private** CustomAssemblyLoadContext \_context = **new**();

**public** **AssemblyService**(ILogger<AssemblyService> logger)

{

\_logger = logger;

}

**public** **void** **LoadAssembly**()

{

\_context = **new** CustomAssemblyLoadContext();

**var** assembly = \_context.LoadFromStream(**new** MemoryStream(File.ReadAllBytes(AssemblyFilePath)));

Console.WriteLine(assembly.FullName);

**var** type = assembly.GetType("Exporter.LiveSync.Program");

**var** greetMethod = type.GetMethod("StartServer");

**var** instance = Activator.CreateInstance(type);

**new** **Thread**(() =>

{

**try**

{

greetMethod?.Invoke(instance, **new** **object?**[] { \_cancelTokenSource.Token });

\_logger.LogInformation("StartServer Invoked");

}

**catch** (Exception ex)

{

\_logger.LogError(ex.Message);

}

}).Start();

\_logger.LogInformation("Thread started");

}

**public** **void** **UnloadAssembly**()

{

**try**

{

\_cancelTokenSource.Cancel();

\_cancelTokenSource = **new** CancellationTokenSource();

\_logger.LogInformation("Token cleaned");

GC.Collect();

GC.WaitForPendingFinalizers();

\_context.Unload();

}

**catch** (Exception ex)

{

\_logger.LogError(ex.Message);

}

}

}

Worker.cs

**using** **Exporter.Service.Interfaces**;

**namespace** **Exporter.Service**;

**public** **class** **Worker** : BackgroundService

{

**private** **readonly** IAssemblyService \_assemblyService;

**private** **readonly** ILogger<Worker> \_logger;

**private** **readonly** ITcpServer \_tcpServer;

**public** **Worker**(ILogger<Worker> logger, IAssemblyService assemblyService, ITcpServer tcpServer)

{

\_logger = logger;

\_assemblyService = assemblyService;

\_tcpServer = tcpServer;

}

**public** **override** Task **StartAsync**(CancellationToken cancellationToken)

{

\_assemblyService.LoadAssembly();

**return** **base**.StartAsync(cancellationToken);

}

**protected** **override** **async** Task **ExecuteAsync**(CancellationToken stoppingToken)

{

**new** **Thread**(() => \_tcpServer.Start()).Start();

**while** (!stoppingToken.IsCancellationRequested)

{

\_logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);

**await** Task.Delay(**1000** \* **60**, stoppingToken);

}

}

**public** **override** Task **StopAsync**(CancellationToken cancellationToken)

{

\_assemblyService.UnloadAssembly();

**return** **base**.StopAsync(cancellationToken);

}

}

Program.cs

**using** **Exporter.Service**;

**using** **Exporter.Service.Interfaces**;

**using** **Exporter.Service.Services**;

**var** host = Host.CreateDefaultBuilder(args)

.UseWindowsService()

.ConfigureServices(services =>

{

services.AddHostedService<Worker>();

services.AddSingleton<IAssemblyService, AssemblyService>();

services.AddSingleton<ITcpServer, TcpServer>();

})

.Build();

**await** host.RunAsync();

CustomAssemblyLoadContext.cs

**using** **System.Reflection**;

**using** **System.Runtime.Loader**;

**namespace** **Exporter.Service.Services**;

**public** **class** **CustomAssemblyLoadContext** : AssemblyLoadContext

{

**public** **CustomAssemblyLoadContext**() : **base**(**true**)

{

}

**protected** **override** Assembly? Load(AssemblyName assemblyName)

{

**return** **null**;

}

}