ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ

Информационная система, которую я намерен реализовать, является совокупностью нескольких типов архитектур информационных систем. В силу своего опыта в коммерческой разработке, а также большого опыта проектирования высоконагруженных информационных систем в крупных компаниях на рынке, могу сказать, что мой подход является оптимальным решением для реализации информационной системы типа CMDB.

В моем проекте необходимо реализовать множество программных компонентов, минимизировав связанность и повысив их связность.

Первым программным компонентом, является консольное приложение, запущенное на хостинге. Данный компонент должен обеспечивать автоматизированную запись в локальное хранилище. Период записи должен задаваться в конфигурации приложения. Данное приложение является источником и сборщиком данных, с которыми мы будем работать в дальнейшем.

Вторым по счету, но не по важности является сервис, расположенный локально на одной машине с консольным приложением. Данный сервис является связующим звеном с центральным сервером. По запросу из центрального сервера, данный сервис должен считывать необходимые данные из локального хранилища и возвращать их.

Локальное хранилище в моей информационной системе является инструментом, благодаря которому я добиваюсь необходимой слабой связанности программных компонентов. Каждое из решений на локальной машине, не знает о существовании другой программы, в счет того, что взаимодействует только с локальным хранилищем. Данный подход позволяет сделать информационную систему легко расширяемой. Мы добились одного из принципа SOLID «открыт для расширения, закрыт для изменения».

Следующим наиболее важным компонентом является реализация сервиса на централизованном сервере. Данный сервис автоматически в каждый заданный период, запускает синхронизацию по всем зарегистрированным в системе машинам (персональным компьютерам). Получая ответ от каждого компьютера данный сервис должен записать результаты в свою реляционную базу данных, что важно, идентифицируя полученные данные с адресом компьютера откуда они пришли. Также данный сервис обеспечивает работу части пользовательского интерфейса, возвращает данные по запросу, фильтрует их, выполняет работу авторизации и регистрации, а также предоставления доступа пользователям (настройка ролевого доступа).

На централизованном сервере есть своя база данных, где хранятся данные по всем пользователям, а также их доступам и ролям. И данные о системных характеристиках персональных компьютеров и информации о запущенных приложениях на этих устройствах.

Заключительным компонентом информационной системы является пользовательский интерфейс в виде сайта. Это веб-приложение, позволяющее работать с данными.

В данной информационной системе, в силу необычной структуры, комбинируется локальная архитектура и дважды клиент-серверная.

Для реализации первого компонента используется современная технология .net 6, данная технология позволяет осуществлять сбор системных характеристик персонального компьютера, и также является кроссплатформенной, что делает нас независимыми от операционной системы на персональном компьютере. Для реализации автоматического сбора информации используется технология Quartz.

В качестве локального хранилища используется технология Lite DB, ориентированная под работу на .net является документоориентированной, нереляционной, бессерверной и быстродействующей.

Для создания локального сервиса используется .net 6 версии, данный сервис общается с централизованным сервером через протокол http.

Сервис на централизованном сервере создан на технологии .net 6. Периодическая синхронизация реализована с помощью технологии Quartz. В качестве ORM используется EntityFramework. Протокол для общения с удаленными машинами и пользовательским интерфейсом http.

В качестве реляционной базы данных на централизованном сервере используется PostgreSQL.

Для отображения данных на пользовательском интерфейсе используется vue 3.

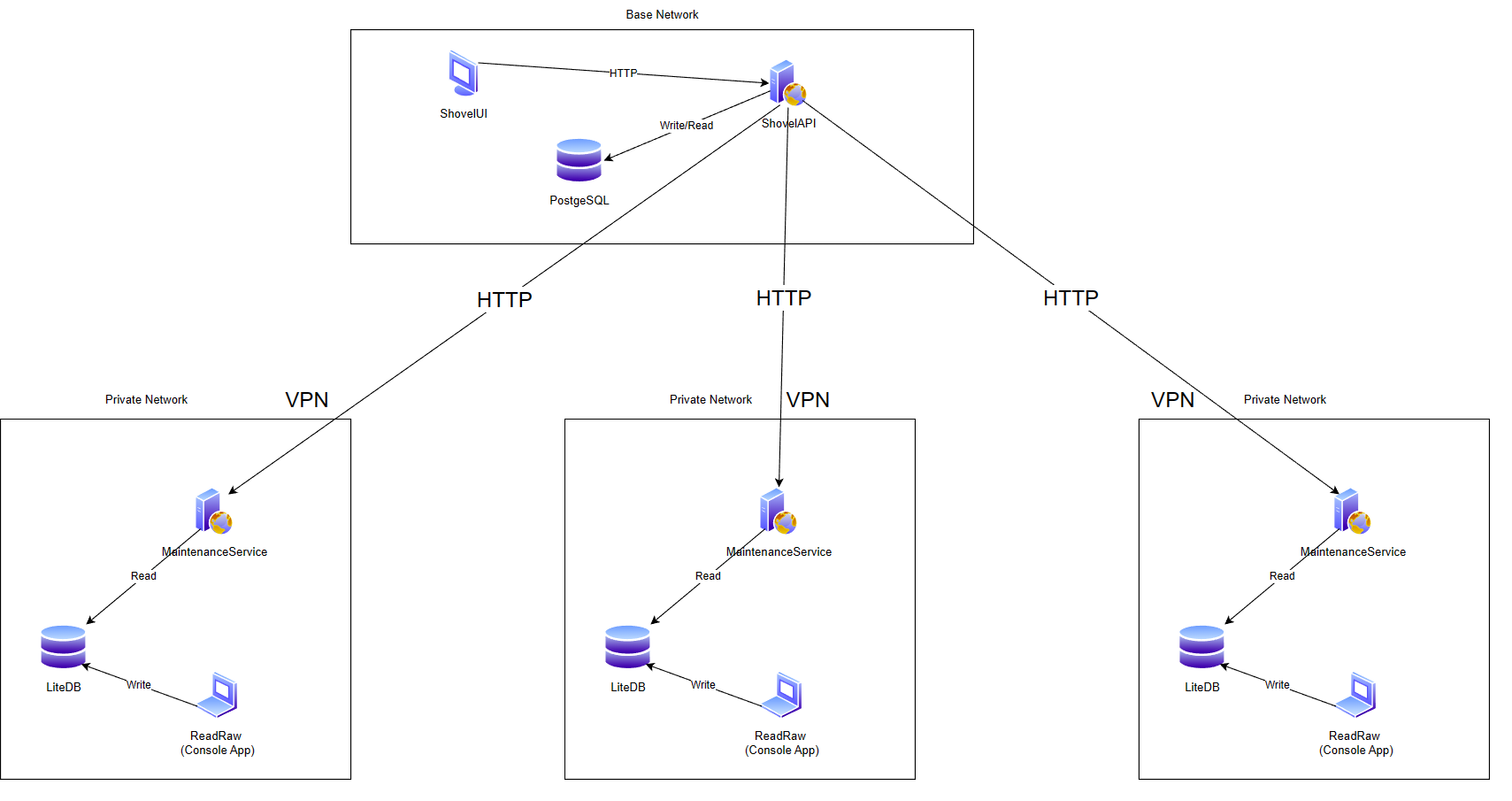


Рисунок 1 – Архитектура информационной системы

В данной диаграмме представлено три локальной сети, и одна централизованная сеть. В каждой локальной сети есть одинаковый набор компонентов. Сервис на централизованном сервере проводит синхронизация по всем локальным сетям.

Консольное приложение под названием «ReadRaw» - что в переводе считать сырое, записывает с заданной периодичностью данные в свое локальное хранилище. Затем «MaintenanceService» - сервис посредник, по запросы считывает данные из базы и возвращает на централизованный сервер. ShovelAPI запускает синхронизацию и обращается к каждому устройству по протоколу http и далее записывает полученные данные в своё реляционное хранилище. На shovel-ui мы уже можем увидеть данные запросив их у ShovelAPI, а он в свою очередь возьмет данные из базы.

На рисунке 1 внутри «Private network» указана локальная архитектура, для примера она продублирована трижды. В качестве клиент-серверной архитектуры мы представляем общение сервиса ShovelAPI и Maintenance сервиса. В виде абстракции клиентом, запрашивающим данные, является ShovelAPI, а Maintenance стороной возвращающей данные. И вторым представлением клиент-серверной архитектуры является общение shovel-ui и ShovelAPI, веб-приложение запрашивает данные у сервиса, который имеет доступ к базе.