СРАВНИТЕЛЬНЫЙ АНАЛИЗ И ОЦЕНКА МЕТОДОЛОГИЙ ПОСТРОЕНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ В ОБЛАСТИ САПР БОРТОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОСМИЧЕСКОГО АППАРАТА

М.А. Кагарманов
Научный руководитель: С.Г. Цапко
Томский политехнический университет
mak55@tpu.ru

Введение

Существующая система автоматизированного проектирования бортового программного обеспечения космического аппарата (далее САПР БПО) выполняет роль информационной системы, хранящей совокупность данных об изделии, требуемых проектанту на этапах определения требований, архитектурного проектирования, тестирования и сопровождения.

Однако, из-за архитектуры клиента возникают трудности в плане обновлений или отката к предыдущим версиям. Так же существующее решение можно развернуть только на платформе с ОС Windows.

Изучение существующего решения на базе WPF-клиент и WCF-сервер

На данный момент проект САПР БПО представляет собой клиент-серверное приложение. Сервером в этой системе является набор WCF-сервисов, хостящихся на IIS. Клиент – это WPF-приложение. В качестве СУБД выбрана Microsoft SQL Server. (Рисунок 1)

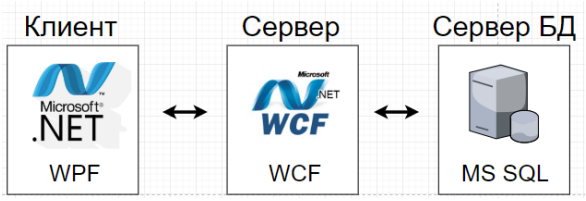

Рис. 1. Архитектура существующего проекта

Главным плюсом такого подхода является то, что он уже написан и исправно функционирует. Также, в очень крупной перспективе, толстый клиент будет мощнее своего браузерного аналога.

Но в такой архитектуре имеется и ряд проблем. Десктопный клиент сложен в обновлении: либо нужно писать систему обновлений, либо производить обновление на каждом клиенте вручную.

Сравнительный анализ наиболее часто используемых подходов при построении клиентской части информационных систем.

Сегодня существует большое количество веб-фреймворков для построения масштабируемых и гибких веб-приложений. Согласно исследованиям CUBA Platform, веб-фреймворки можно разделить группы по степени скорости разработки, специализации и масштабируемости [1]. Для данного проекта подойдут узконаправленные Enterprise фреймворки такие, как Java EE, Spring, ASP.NET. Все они предназначены для оптимизации разработки в определенной достаточно небольшой области за счет повышения уровня абстракции и предоставления понятного API для своего домена. Связывание ORM, Middleware, UI, Messaging не является тривиальной задачей, и, останавливаясь в этом классе, мы имеем сочетание удовлетворительной скорости разработки с высоким уровнем гибкости.

Варианты реализации клиента: только Angular, только React, ASP.NET Core Web API + JS-framework (Angular/React).

Чистый Angular:

Плюсы:

1. Есть опыт работы
2. Angular Material – язык дизайна для приложений, который был разработан Google.
3. TypeScript.
4. Angular включает в себя dependency injection, паттерн, в котором один объект(сервис) предоставляет зависимости другому объекту(клиенту). Это дает большую гибкость и более чистый код.
5. Больше подходит для Enterprise-решений.
6. У Angular, как у MVC-фреймворка, имеется поддержка MVC из коробки.
7. Angular производительнее React`a, хотя и незначительно
8. Двустороннее связывание в Angular чище в коде и проще для реализации разработчиком.
9. Angular – это фреймворк, “цельное решение”. Не нужно проводить анализ библиотек, решать вопрос с роутингом и т.п. – можно просто начать работать

Минусы:

1. У Angular, определенно, крутая кривая обучения. Это можно рассматривать, как минус из-за большой, сразу существующей экосистемы, которую вам нужно со временем изучить. С другой стороны, это может и хорошо в определенных ситуациях, поскольку многие решения уже приняты
2. Angular постоянно меняется
3. Нет прямой установки js-библиотек, однако Angular имеет возможность установки пакетов через npm.
4. TypeScript-разработчиков гораздо меньше, чем JavaScript-разработчиков

Чистый React:

Плюсы:

1. С React`ом можно работать, просто добавляя JavaScript-библиотеки к исходникам. Это невозможно с Angular, потому что он использует TypeScript. React предоставляет полный контроль над размером приложения, позволяя включить только те вещи, которые действительно нужны.
2. Одностороннее связывание React позволяет получить лучшее понимание о том, что происходит с данными, потому что поток данных течет лишь в одном направлении (это делает отладку проще).
3. React более популярен, и число JavaScript разработчиков очень велико.

Минусы:

1. Из MVC в React есть только View; Model и Contoller нужно реализовывать самому.
2. React – это только UI-библиотека. При создании приложений на React`е нужно использовать другие библиотеки для управления различными частями приложения. Например, состояниями приложения.

Различные best-practice советуют разделять приложение на два отдельных проекта (в данном случае это ASP.NET Core Web API (Backend) и Angular/React (Frontend). Использование .NET Core Web Api с frontend-фреймворком получает все плюсы и минусы конкретного фреймворка. Уже существующий код взаимодействия с сервисами легко может быть вынесен на backend. Однако, код контроллеров придется писать и в этом случае мы получаем дублирование кода.

Основными преимуществами такого подхода будет следующее:

1. Два независимых друг от друга проекта, что позволит в дальнейшем реализовать альтернативный интерфейс, не трогая проект с серверной частью.
2. ASP.NET Core Web API в качестве Backend`а из коробки имеет механизм внедрения зависимостей, JWT-аутентификацию, Code First на ORM Entity Framework, поддержку SQL Server. Angular на frontend`е позволит использовать Angular CLI и Angular Material в качестве дизайна.
3. Абстрагированность от рабочего окружения, в котором разрабатывается серверная часть. Visual Studio или Rider для backend`а, а VS Code, Sublime Text, Atom или другой удобный редактор для Frontend`а.

Выбор методологии построения архитектуры проекта и обоснование выбора

Скорее всего, взаимосвязь .NET и Angular является в данном случае лучшим решением. Во-первых, имеется значительный опыт разработки на .NET, Angular и конкретно с этой архитектурой. Во-вторых, текущий проект написан на .NET и в результате проще переносить код из текущего WPF-клиента (Например, для переноса модели или логики взаимодействия с WCF сервисами достаточно скопировать соответствующие модули и изменить пространства имен).

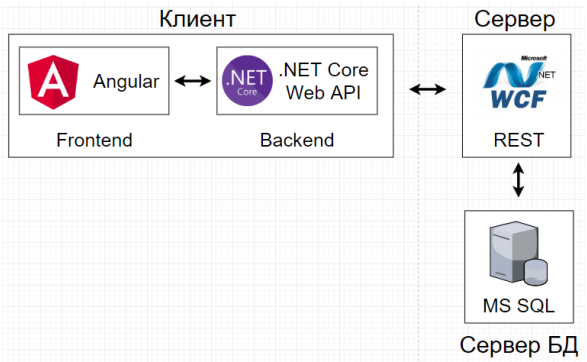

Рис. 2. – Выбранная архитектура приложения

В данной архитектуре Backend выполняет запросы на сервер, получает ответы, обрабатывает их и переводит в нужный формат клиенту. Также backend предоставляет API для клиента. Angular на frontend`е, в свою очередь, отправляет запросы на backend и получает необходимые данные для визуализации.

Заключение

Таким образом, в данной работе было проанализировано существующее решение на базе десктопного клиента и WCF-сервисов. Определены плюсы и минусы такого подхода. Принято решение о переносе клиентской части на веб-версию. Определена методика переноса серверной части (WCF-сервисы) на архитектурный стиль REST. На простейших приложениях методика была протестирована. Рассмотрены наиболее подходящие технологии реализации веб-клиента. Выявлены их сильные и слабые стороны. Для реализации клиента выбрана архитектура ASP.NET Core Web API (Backend) + Angular (Frontend). Построена архитектура веб-версии САПР БПО.

Список использованных источников

1. Classification of Development Frameworks for Enterprise Applications [Электронный ресурс]. - Режим доступа: https://www.cuba-platform.com/blog/classification-of-development-frameworks-for-enterprise-applications, свободный.
2. Angular vs. React vs. Vue: Comparison 2017 [Электронный ресурс]. - Режим доступа: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176, свободный.