**Информационная система для учёта и обработки заявок сервисного центра по ремонту компьютерного и мобильного оборудования**

Информационная система «антитрабл» представляет из себя клиент-серверное приложение на платформе .Net Core с микросервисной архитектурой. Микросервисы общаются между собой используя REST, формат json. Пользователи взаимодействуют с системой посредством http протокола (Рис. 1). Всего существуют 3 микросервиса:

* Person —обеспечивает работу с пользователями системы и счетами клиентов;
* Equipment — взаимодействие с оборудованиями, предоставленными клиентами сервисного центра. (Добавление, удаление, поиск, получение списка);
* Repair — основная часть бизнес-фунций приходится на него. Здесь и оформление заявок, и изменение статуса ремонта, составление отчётов и оплата «заказов»

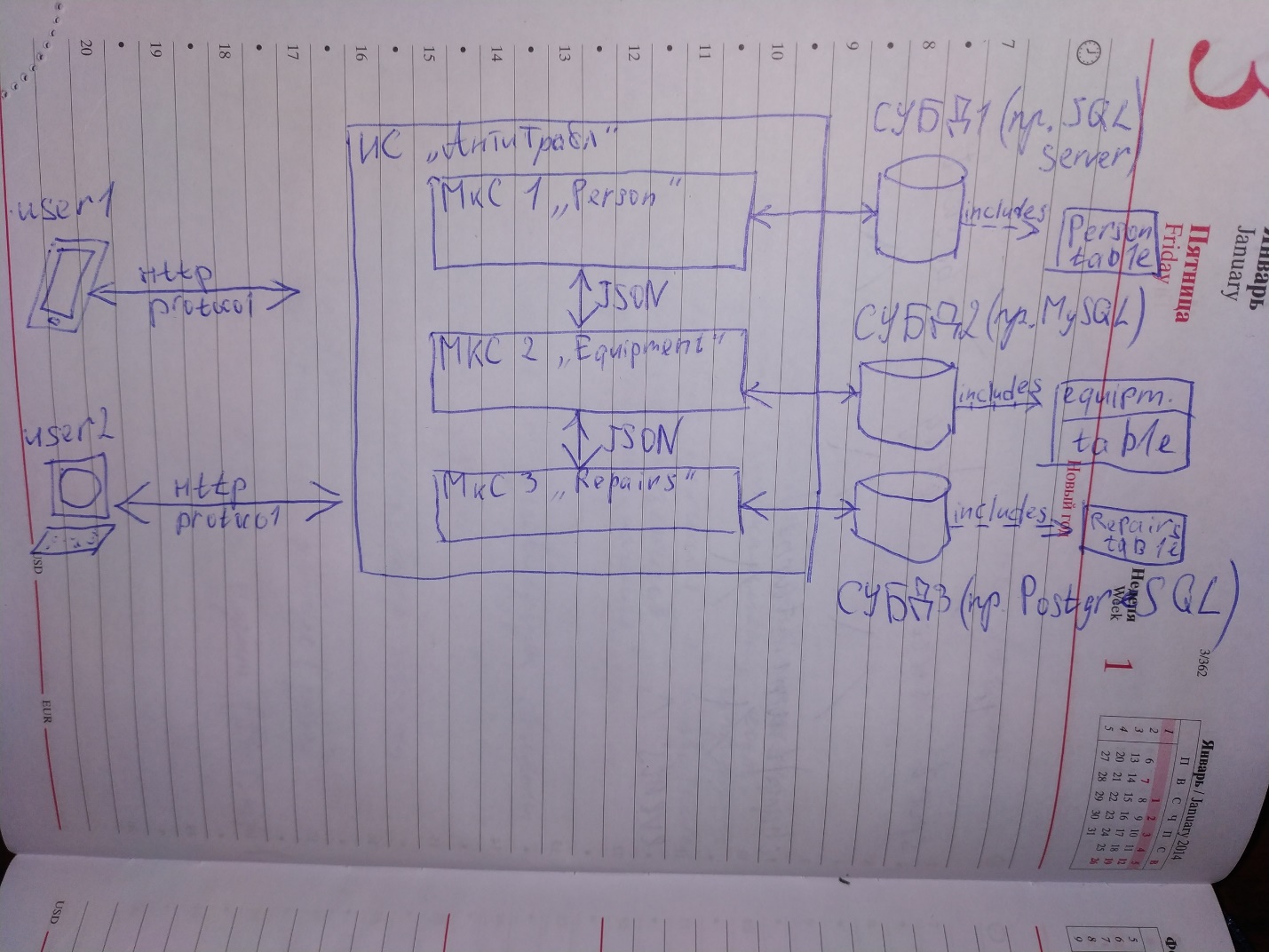


Рис. 1 «Deployment diagram»

Чем же так хороша микросервисная архитектура?

* Писать и поддерживать небольшие сервисы всегда проще, чем большие. Чем меньше кода, тем легче уместить его в голове;
* Поскольку каждый микросервис представляет собой отдельный проект, можно строить иерархии из сервисов. То есть, некие сервисы будут использованы только парой сервисов, предоставляющих внешний API для других сервисов
* бизнес-процессы хорошо ложатся на MSA;
* Для каждого сервиса можно выбрать язык и библиотеки, подходящие конкретно для решаемой этим сервисом задачи.
* Практически из коробки получаем горизонтально масштабируемый и отказоустойчивый код, да и к тому же с конвейерным параллелизмом. Горизонтальное масштабирование, как известно, приводит к экономии денег, так как система может работать на множестве сравнительно недорогих машин. Более того, под каждый конкретный микросервис можно подобрать железо подходящей конфигурации;
* Если в программе всего лишь несколько сотен строк кода, в ней не так-то просто сделать ошибку. Необходимость [юниттестов](https://eax.me/unit-testing/) становится сомнительной, а использовать языки с [динамической типизацией](https://eax.me/dynamic-typing/) уже вроде как и не страшно.
* Модульность.
* Сравнительно нетрудно переписать легаси систему в соответствии с MSA. Ставим перед системой зонтик/прокси. Затем потихоньку выносим компоненты системы в сервисы, а прокси следит за тем, чтобы клиенты ходили туда, куда нужно;
* Даже если ты пилишь тупо сайтик, с MSA чувствуешь себя крутым разработчиком распределенной системы