ADR Разворачивание функционала sAccept в локале и глобале

In Progress

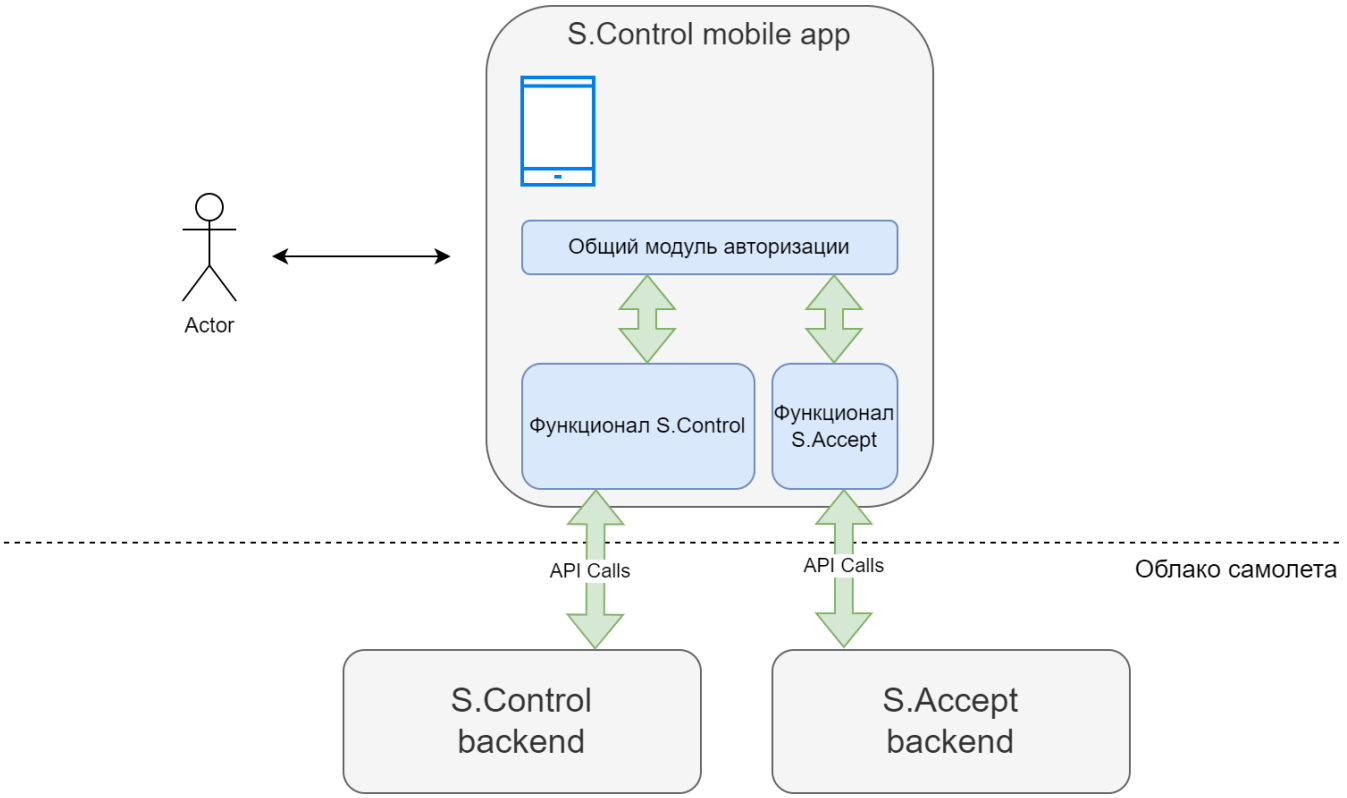
Заказчик (PDM или PO)	Минеева Александра Капшуль Антон
Исполнитель	Хисамов Илья Барган Алексей
Бизнес-инициатива	
Архитектурная задача	~~SPROCUR-15728~~ - ADR на S.Accept Mobile и его выходу в облако ЗАКРЫТО
Статус	ADR_DRAFT
Зона влияния ADR	10D-ADR

Согласование / Участник экспертной группы	Должность/роль	Принял
Матаков Денис (опционально)	Руководитель архитектуры	☐
Беликов Вячеслав	PDL кластера "Снабжение"	☐
Минеева Александра	PO G.Control	☑
Симонов Вячеслав	PO S.Materials	☐
Майоров Виталий	PDL кластера "Стройка"	☑
Капшуль Антон	EM S.Materials + S.Accept	☑
Чернова Дарья	PO S.Accept	☑

Контекст

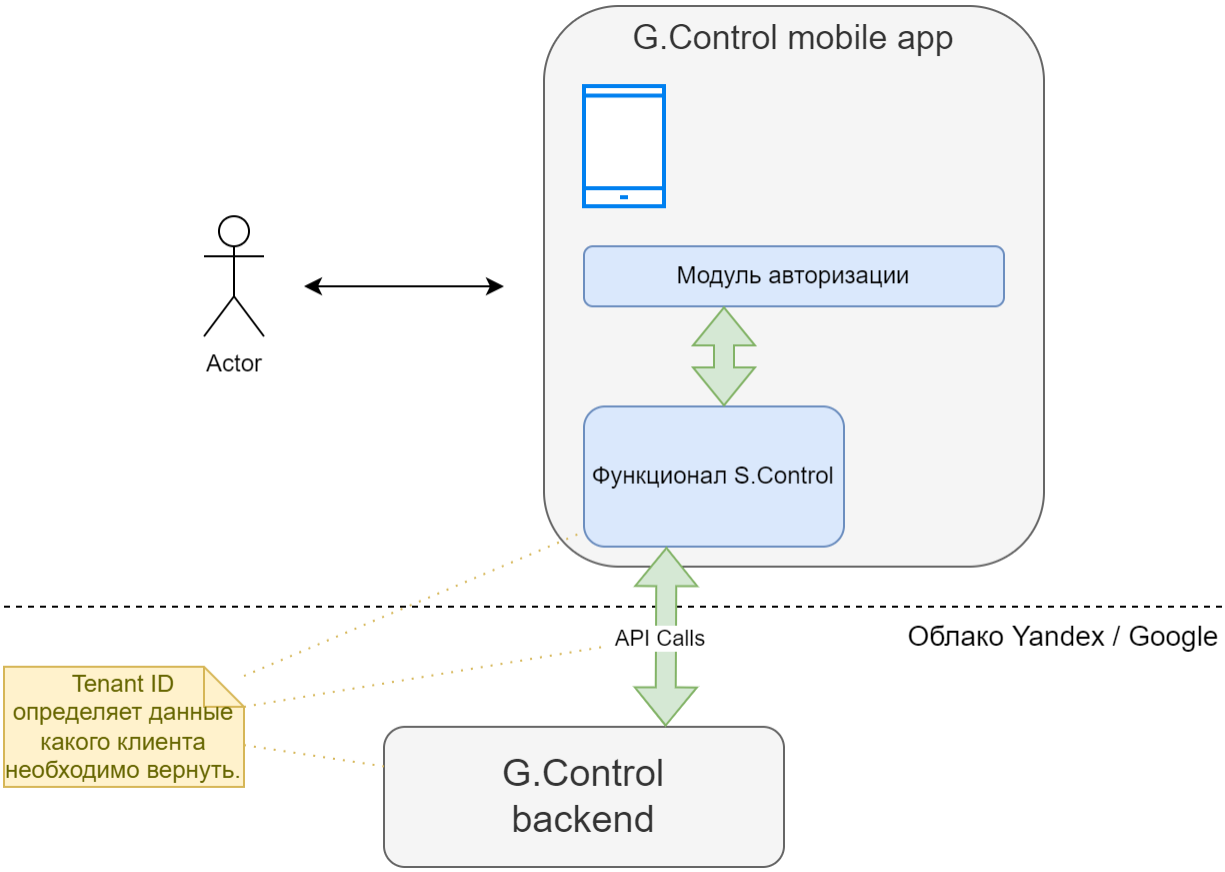
На текущий момент функционал sAccept реализован в мобильном приложении локального S.Control. Есть задача продублировать функционал в Global. Функционал S.Accept реализован как подключаемый модуль в составе большого приложения S.Control. При этом backend среды у них разные.

Архитектура S.Control + S.Accept AS IS:



Для S.Control сейчас функционал S.Accept подключен внутрь мобильного приложения как отдельный React Native компонент. Авторизация у них общая. При этом бекенды для S.Control и S.Accept разные.

Архитектура G.Control AS IS:



Для G.Control функционал приемки товара (S.Accept) не поддержан т.к. в бекенд и мобильной части S.Accept на текущий момент нет поддержки мультитинантности.

В рамках текущего ADR необходимо определить архитектуру целевого G.Accept.

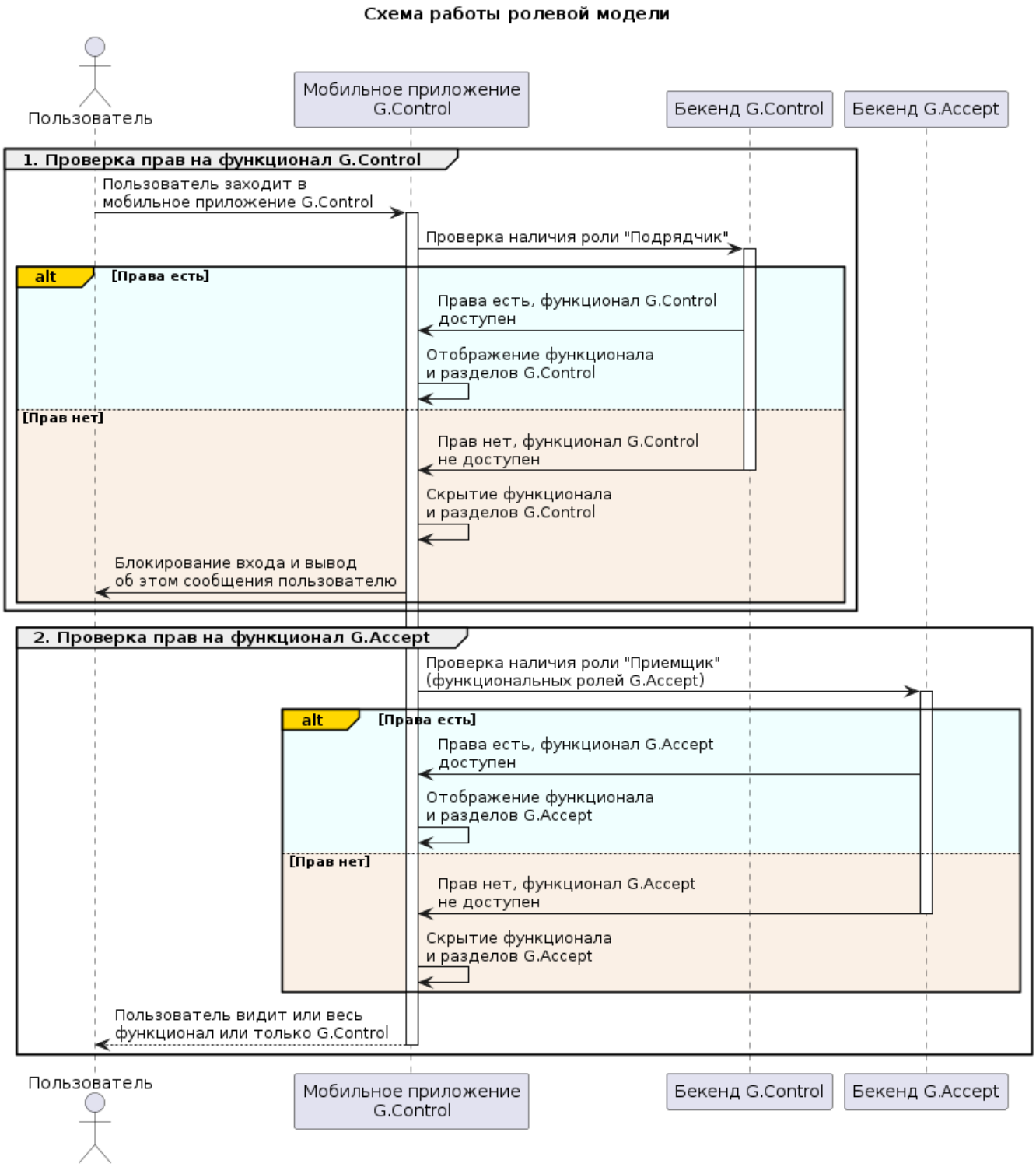
Цель (чтобы что?)

1. Определить механизм разработки и выкатки раздела S.Accept **(в мобильном приложении)**, чтобы упростить интеграцию в глобальном облаке для клиентов и локальном облаке Самолета. Цель - реализация функционала одной командой, без необходимости копировать код между двумя приложениями;
2. Определить мультитинантную архитектуру работы G.Accept (backend и frontend) в составе приложения G.Control.

Решение

Ролевая модель

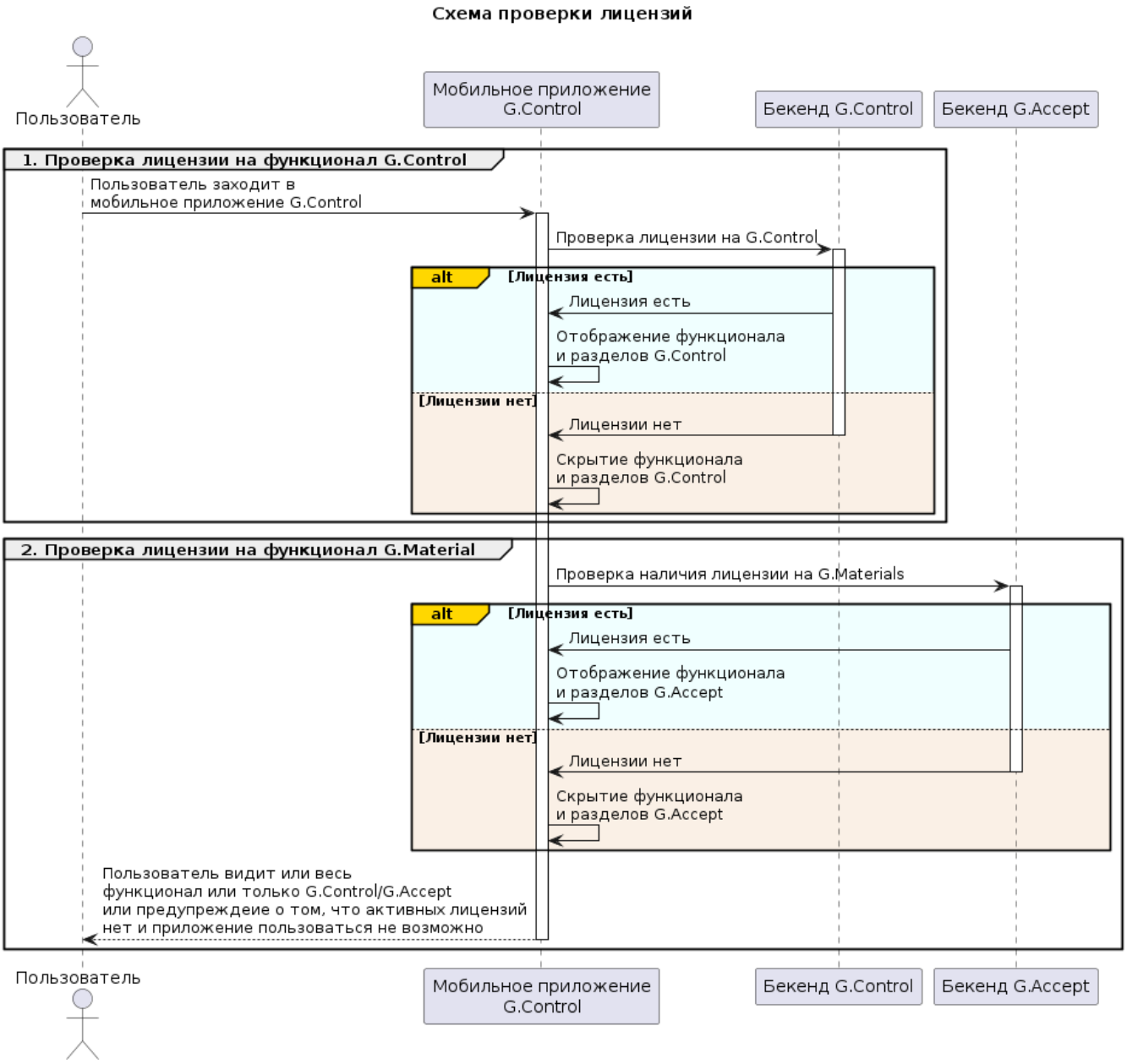
В приложении возможна работа пользователями с двумя разными ролями. Для G.Control роль - подрядчика. Для G.Accept своя ролевая модель и роль приемщика. В текущей логике в приложение пользователь входит под ролью подрядчика (иначе он не сможет войти в приложение), далее когда подрядчиком уже зашли идет доп. запрос в бекенд G.Accept где проверяются права уже по его ролевой модели (наличие роли приемщика). Если тут прав также нет, то никакой функционал в приложении будет не доступен.



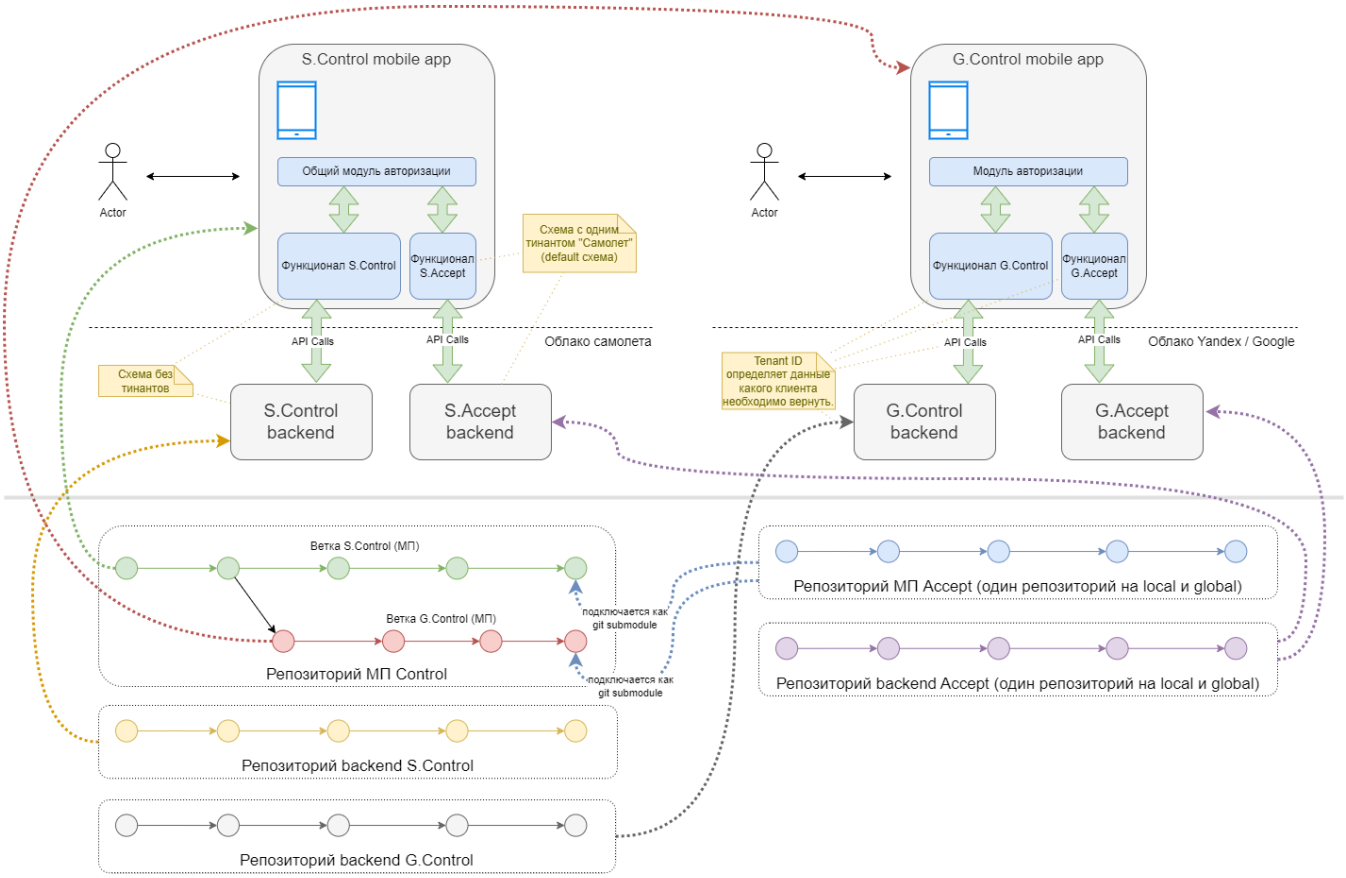
Проверки лицензий

На входе в приложение так-же необходимо проверять наличие лицензий. Функционал G.Control лицензируется отдельно, функционал G.Accept отдельно не лицензируется (входит в состав лицензии на G.Materials).

Таким образом если куплен функционал Control, но не куплен функционал Materials, то показывается только часть приложения по функционалу Control.



Предлагаемая архитектура



1. Мобильное приложение S.Control и G.Control сейчас собирается по двум веткам в **одном** репозитории. В перспективе предполагается разделить функционал приложения на два отдельных репозитория т.к. функционал сильно расходится уже сейчас.
2. В репозитории мобильных приложений S.Control и G.Control функционал Accept подключается как git submodule. При этом функционал Accept не разделяется на разные ветки или репозитории для local и global. Мы предполагаем, что функционал будет единым и не будет расходиться (стремимся к этому в архитектуре приложения). В случае если необходимо будет сделать разный функционал, для S.Accept и G.Accept, мы будем использовать DI-контейнеры для разделения функционала (вместо feature toggles). Подробнее про эту механику тут (ADR еще будет детализироваться) ADR 06.12.2023 Перевод S.Materials на модульную модель
3. Бекенд для локального и глобального Control остается разделен по двум независимым репозиториям (функционал сильно расходится).
4. Бекенд для локального и глобального Accept единый и не разделяется по репозиториям. Мы поднимаем один и тот-же код бекенда в разных ЦОД, в периметре самолета функционал работает как один tenant на все облако самолета.

Минусы текущей архитектуры в том, что у нас две Django админки (одна для Control другая для Accept). Это пока оставляем как есть, т.к. вопрос слияния будет дорогим и долгим по времени, а экономическая целесообразность сложна в оценке (пока нет большого количества продаж). Возможно в будущем Accept отделится в отдельное приложение, что будет существенно проще технически реализовать.

Помимо git submodules были рассмотрены и другие варианты подключения функционала S.Accept в состав приложения S.Control:

1) Создание отдельного npm-пакета

- Весь функционал вынесен в отдельный пакет

- Обновление происходит после обновления версии пакета.

2) Создание Git Submodules

- Весь функционал вынесен в отдельный репозиторий, который подключен к обеим проектам

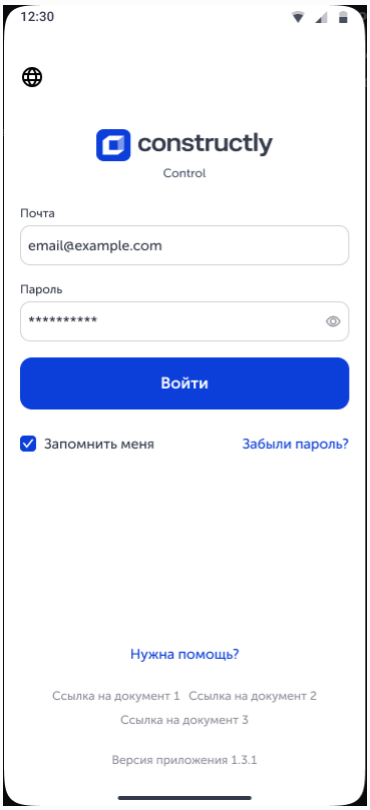
- Обновление происходит после обновления метки подмодуля.

3) Изменение структуры проекта

- Оба приложения лежат в одном проекте, общие компоненты вынесены, различные относятся уже к конкретным проектам

По итогу был выбран вариант с Git Submodules, можно будет независимо разрабатывать раздел, при этом при необходимости, останется возможность взаимодействия и с остальными модулями приложений. Разрабатывать можно будет в одном из проектов, в другом проекте просто обновлять метку на модуль.

Тинант будет определяться по логину пользователя. См. ADR KeyCloak for mobile

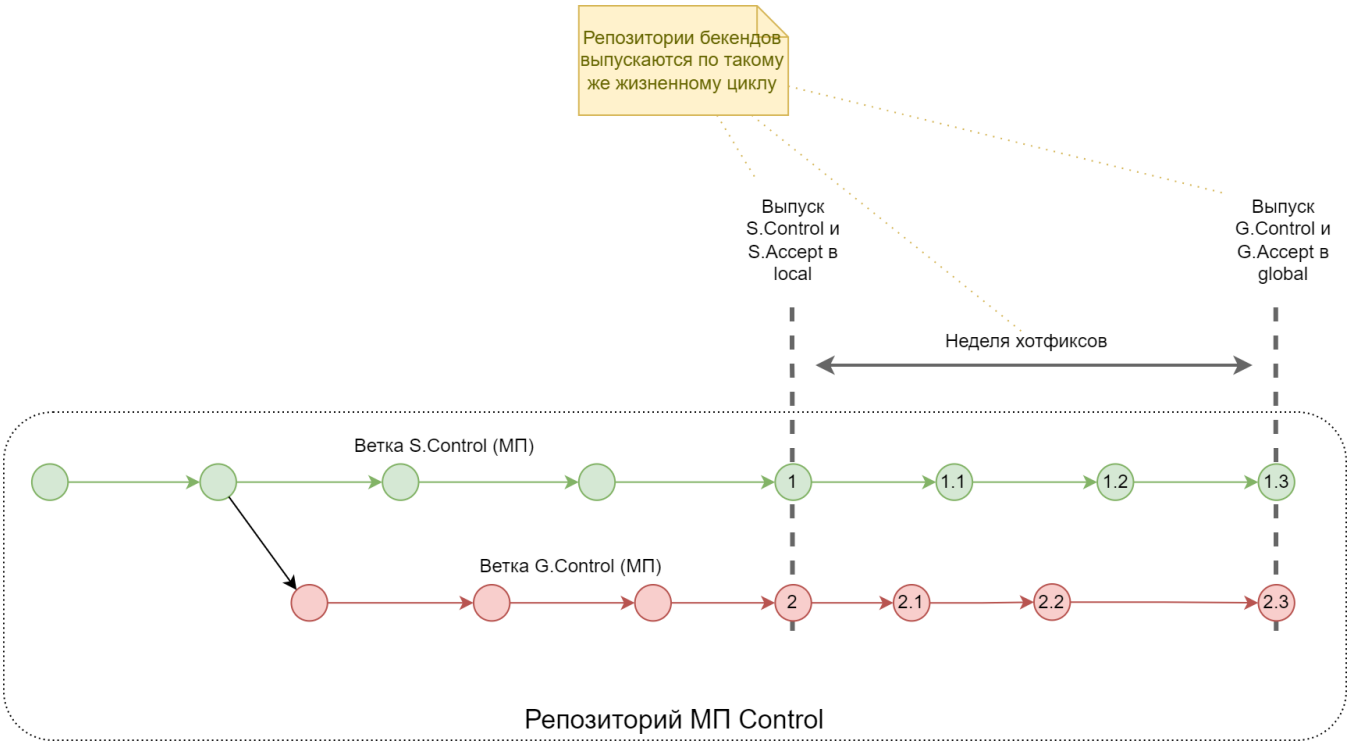


На стороне приложения парсится введенный логин - email, достается значение после "@" и до точки (реалм) login@**company**.xyz, далее шлётся в кейклок логин\почта и пароль на определенный реалм. В данном случае company это и есть идентификатор тинанта.

Всем клиентам выдается email с доменом основного сайта их компании. Если у клиента нет сайта, то при создании его учетки мы просто пишем какую-то строку, например v.petrol@**ip_matveev**.ru и ip_matveev будет идентифицировать его тинант. Понятно, что такого адреса email физически не существует, но в данном случае у нас это просто строка - логин похожая на email.

Релизный цикл

Весь функционал сначала релизится для самолета. Потом идет период хотфиксов (ориентировочно неделя) и если нет никаких блокеров то выпускается для Global.



Очевидно что клиенты могут не обновлять свое мобильное приложение из маркета, в этом случае мы поддерживаем работу старого API какое-то количество версий назад. После этого выводим пользователю предупреждение при входе "Ваша версия устарела, пожалуйста, обновитесь". Подобная механика позволит не изводить пользователя постоянными предложениями обновить версию приложения.

Поддержку старых приложений по API можно сделать таким образом, что версия передается в заголовках запроса к API (в этом случае в методах должна быть условная if логика на разные версии приложения) или новый метод именовать по новому (или реализовать его так, чтобы он содержал другое число параметров), в этом случае какое-то время в коде бекенда будет два разных метода (старый и новый).

С прекращением поддержки старых версий мобильного приложения надо будет удалять старые методы или часть кода по старым if веткам.

Следует учитывать что необходимость поддержки старых версий требует большей нагрузки на QA, поэтому вопрос целесообразности поддержки старых версий необходимо согласовывать с бизнесом.

Ответственность команд

Команда S.Accept - отвечает за бекенд S.Accept и G.Accept. При необходимости им может помогать команда S.Materials и G.Supply.

Команда Control - отвечает за разработку мобильной части как S и G.Control, так и за S и G.Accept.

QA Accept - отвечает за тестирование функционала Accept в мобилке S и G.Control.

QA Control - отвечает за тестирование функционала Control в мобилке S и G.Control.

Пошаговый план действий

1. Влить обновленную версию S.Accept в G.Control (сейчас там очень старая версия), добиться корректной компиляции и сборки.
2. В репозитории S.Control и G.Control подключить функционал мобильного S.Accept как git submodule (предварительно выяснить какие библиотеки мобильного S.Accept надо добавлять в основной проект S и G.Control чтобы не возникло конфликтов зависимостей).
3. Выяснить все зависимости сервиса S.Accept (на бекенде) и перевести их на мультитинантную схему **или** избавиться от этих зависимостей (теоретически должны зависеть только от S.Materials, который уже в процессе перевода в Global).
4. Добавить поддержку мультитинантности в функционал G.Accept на бекенде (и всем зависимым сервисам от которых не сможем избавиться в рамках реализации п.3).
5. Развернуть функционал Accept в облаке Яндекса.
6. Выпустить функционал Accept в облаке Самолет с одной default схемой.
7. Profit!