DOMA API интеграция

Привет, ты попал на страницу, где мы покажем, как ты можешь использовать API платформы DOMA. DOMA – это платформа для автоматизации работы управляющих организаций (УО: УК, ТСЖ, ТСН), относится к классу “Property Management Platform” решений.

Работа системы строится вокруг объектов общей собственности (*Property*) управлением которыми занимается специальная организация (*Organization*).

Все запросы к API должны быть выполнены от имени учетной записи (*User*). Идентификатором учетной записи является номер телефона (*phone*). Существует два типа учетных записей:

- сотрудники управляющих организаций (*type=staff*) – это пользователи, которых пригласила управляющая организация (б2б пользователи)
- жители (*type=resident*) – это пользователи самостоятельно зарегистрировавшиеся в приложении жителя (б2с ползьователи)

Это значит, что с одним номером телефона, можно быть сотрудником управляющей компании и жителем, используя приложение жителя. Это две разные учетные записи!

Рассмотрим следующий клиентский путь для сотрудника УО:

1. Пользователь регистрируется на платформе используя номер телефона (phone)
2. Создает новую организацию указывая ИНН (*tin*) и название (*name*)
3. Заходит в список сотрудников и добавляет туда своих коллег указывая их номера телефонов (*phone*) и адреса эл. почты (*email*)
4. Заходит на страницу объектов общей собственности и создает добавляет туда адрес дома который обслуживает данная УО (*address*)
5. Заходит в список заявок и создает там новую заявку

Что создается с точки зрения API на каждом из этих шагов:

1. Создается учетная запись (*User*) с полями name, phone, email, password со значение User.type=*staff*
2. Создается организация (Organization) с полями tin, name, создается сотрудник этой организации (*OrganizationEmployee*) с ролью имеющей максимальные права в этой организации (*OrganizationEmployeeRole*)
3. Создается объект сотрудник (*OrganizationEmployee*) для каждого приглашенного сотрудника, она связывается с учетной записью (*User*) для этих сотрудников. Если в системе уже была учетная запись с User.phone и User.type=*staff*, то она будет использована для создания объекта сотрудник (*OrganizationEmployee*), если такой учетной записи нет, то она создастся и пользователю на номер телефона придут данные для первого входа
4. Создается объект общей собственности (*Property*) с полями address, addressKey, addressMeta

5. Создается объект заявки (*Ticket*) с полями details, number, classifier, source, assignee, executor, deadline, clinetName, clientPhone, clinetEmail; если заявка была от жителя, то создастся его контакт (*Contact*) с полями name, phone, email

Таким образом, для получения доступа к данным организации должен существовать объект сотрудник (*OrganizationEmployee*) с ролью имеющей соответствующие права доступа (*OrganizationEmployeeRole*).

Рассмотри пример действий жителя в связке с запросами к API:

A. Регистрируется на платформе подтверждая свой номер телефона (phone)
B. Вводит адрес проживания (address) указывая номер квартиры (unitName) и указывает тип помещения (unitType)
C. Вводит свой лицевой счет (accountNumber)
D. Создает заявку

Что создается с точки зрения API на каждом из этих шагов:

A. Создается учетная запись (*User*) с полями name, phone, email, type=*resident*
B. Создается объект проживание (*Resident*) с полями address, unitName, unitType
C. Создается объект потребление услуги (*ServiceConsumer*) с полем accountNumber
D. Создается объект Ticket с полями details, number и Contact с полями name, phone, email доступные на редактирование для УО зарегистрировавшей данный адрес

Подробные примеры запросов и ответов API платформы DOMA рассматривается на кейсах. Советуем начать 🎓 Общий подход к API

Оглавление

Оглавление **2**

🎓 Общий подход к API **5**

Работа с API playground (консоль для выполнения запросов) 5

Пример получения данных по организациям (organization) 5

Получение списка организаций через allOrganizations 5

Получение организации по идентификатору через Organization 7

Получение кол-ва организаций через _allOrnanizationsMeta 8

Список доступных сотруднику организаций с ролями и правами в них (OrganizationEmployees) 9

Список сотрудников в организации (OrganizationEmployees) 10

Работа с адресами (Property) 12

Работа с заявками (Ticket) 14

Получение списка заявок (allTickets) 14

Создание новой заявки (createTicket) 18

Изменение заявки (updateTicket) 20

Просмотр истории изменений (TicketChanges) 22

Получение классификатора, списка статусов, источников заявок 24
Просмотр списка комментариев (allTicketComments) 25
Прочтение комментариев к заявке 26
Работа с контактами жителей (Contact) 27
Получение списка контактов (allContacts) 27
Получение списка ролей контактов жителей (allContactRoles) 28
Создание новой роли для контактов жителей (createContactRole) 30
Обновление созданной роли для контактов жителей (updateContactRole) 31
Удаление созданной роли контакта жителя (updateContactRole) 33
Создание контакта жителя (createContact) 34
Обновление контакта жителя (updateContact) 36
Удаление контакта жителя (updateContact) 38
Работа с ИПУ (Meter) 39
Просмотр списка приборов учета (allMeters) 39
Просмотр переданных показаний для приборов учета (allMeterReadings) 40
Основные поля показаний приборов учета (MeterReading) 43
Основные поля прибора учета (Meter) 44
Создание счетчика (createMeter) 46
Обновление счетчика (updateMeter) 48
Передача показаний прибора учета (createMeterReading) 49
Работа с новостями 50
Создание новости (createNewsItem) 51
Получение списка новостей (allNewsItems) 54
Создание области видимости новости (createNewsItemScope) 56
Просмотр области видимости новости (allNewsItemScopes) 59
Редактирование новости (updateNewsItem) 59
Прочтение новости пользователем (createNewsItemUserRead) 59
Работа со счетами (invoice) 59
Подготовка 60
Создание счета 61
Расщепление 66
Редактирование счета 68
Публикация счета 70
Ссылка на оплату счета 70
Получение списка счетов (allInvoices) 70
Проверка оплаты счета 72
🔐 **Аутентификация и авторизация (#auth)** **75**

Open ID Connect и мини-приложения **75**
Как это работает? 76
Попробуем пройти этот процесс руками! 76

🎓 Общий подход к API

Данный раздел переехал: https://developers.doma.ai/ru/docs/api/about

Ниже мы также собрали некоторые примеры работы с API в **postman**.

Проще всего разобраться в нашем API рассмотрев примеры.

Работа с API playground (консоль для выполнения запросов)

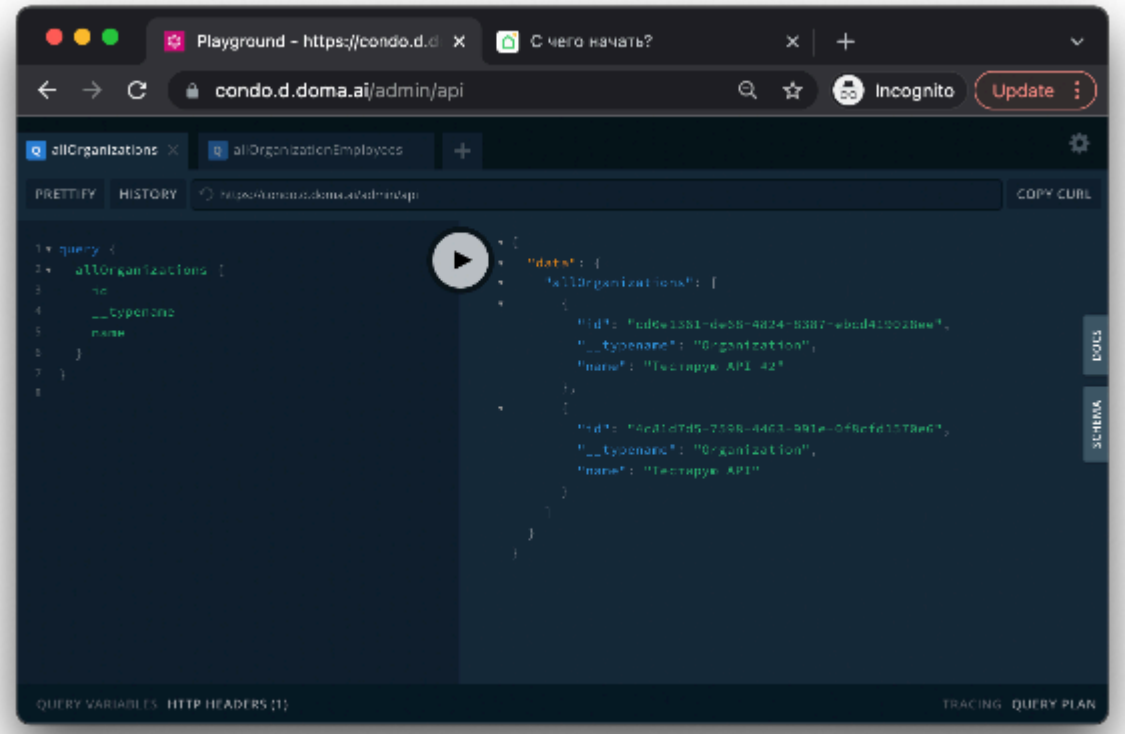
Раздел переехал: https://developers.doma.ai/ru/docs/api/playground

Пример получения данных по организациям (organization)

Рассмотрим работу с DOMA API на примере работы со списком организаций (organization). У каждого объекта типа организация есть набор атрибутов, это: имя (name), ИНН (tin). Для выполнения запросов, вы должны пройти аутентификацию и иметь необходимый набор прав доступов (смотри подробнее предыдущий раздел “Работа с API playground”).

Получение списка организаций через allOrganizations

Например, мы хотим получить список всех организаций и их названия:



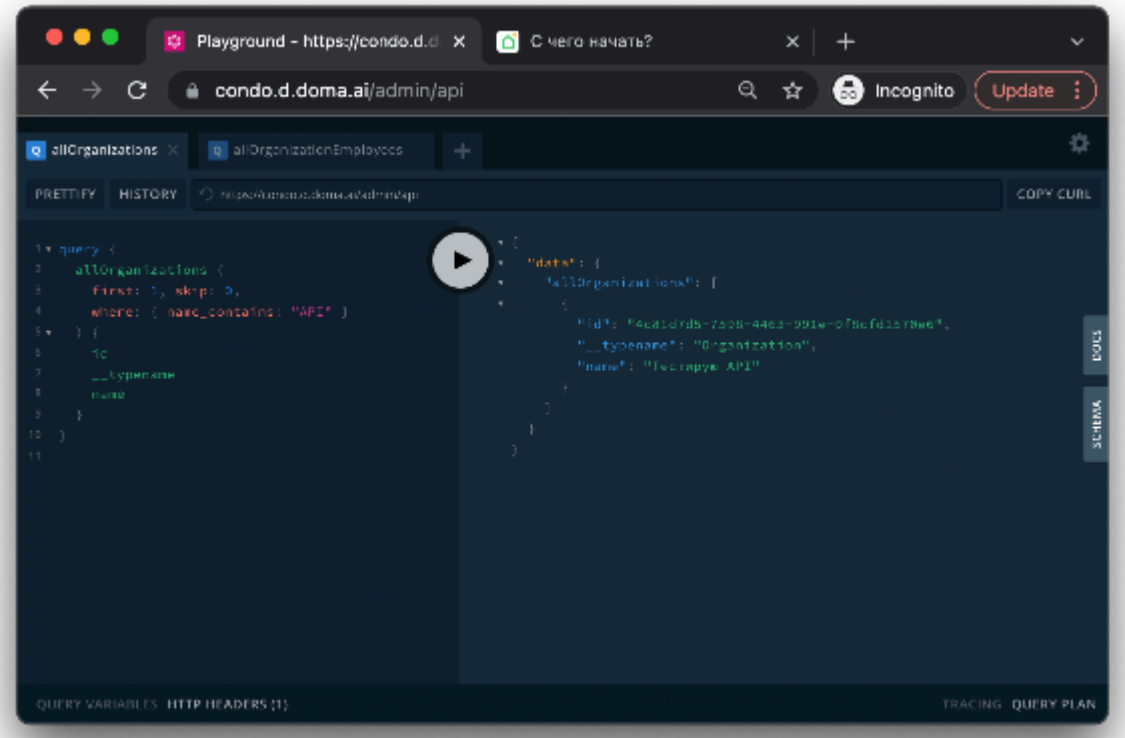
QUERY	RESULT

{
allOrganizations {
id
__typename
name
}
}
	{
	"data": {
	"allOrganizations": [
	{
	"id": "81cfe-3fcb-495f6cee",
	"__typename": "Organization",
	"name": "OOO DOMA"
	}
	]
	}
	}

Например, данный запрос можно выполнить с использованием команды curl:

```
curl 'https://condo.d.doma.ai/admin/api' -H 'Content-Type: application/json' -H 'Authorization: Bearer Z9jPhmz.5VCBI' --data-binary '{"query":"query { allOrganizations {  id __typename name } } "}'
```

Пример с постраничным выводом, фильтрацией и сортировкой:



QUERY	RESULT
{ allOrganizations (first: 1, skip: 0, where: { name_contains: "DOMA" }) {	{ "data": {

```
    id
    __typename
    name
  }
}
```

```
  "allOrganizations": [
    {
      "id": "81cfe-3fcb-495f6cee",
      "__typename": "Organization",
      "name": "OOO DOMA"
    }
  ]
 }
}
```

где **first** - сколько элементов мы хотим получить, **skip** - сдвиг относительно начала списка или сколько элементов мы хотим пропустить, **where** - условие фильтрации/поиска. Вы также можете добавить `**sortBy**: [createdAt_DESC]` для сортировки списка.

Подробное описание полей и параметров фильтрации и сортировки смотрите в (#2)

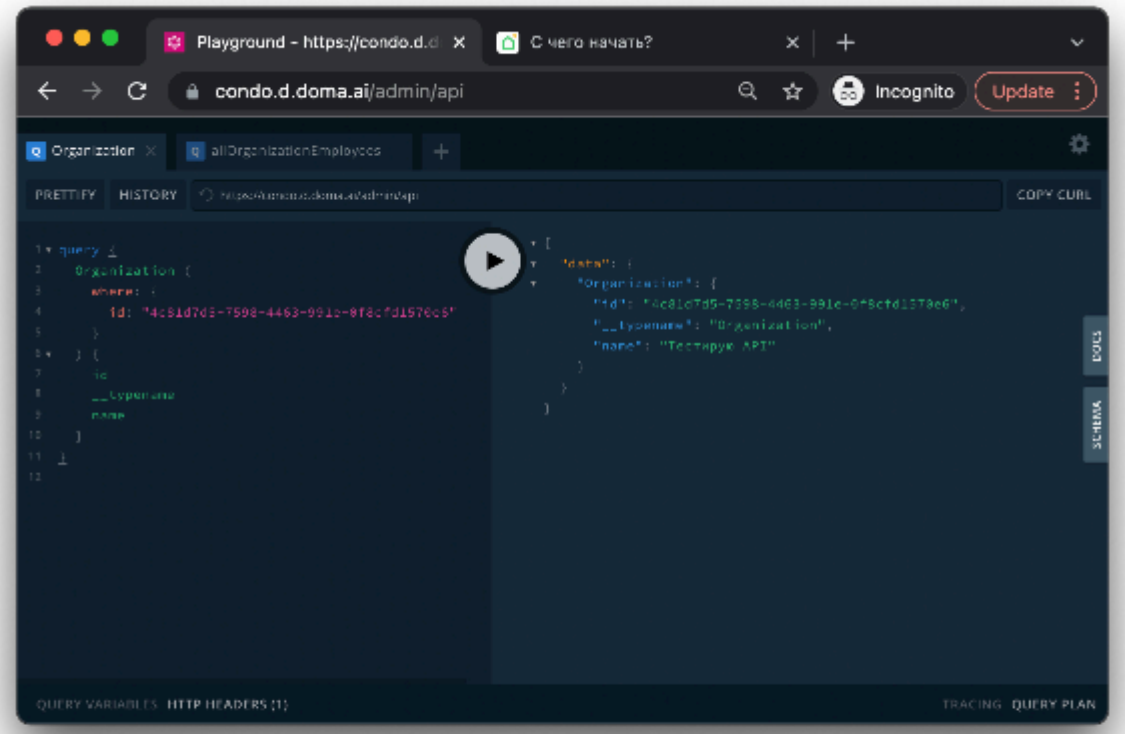
Мы не всегда отвечаем данными на запросы 😉, иногда, у вас не будет прав для чтения каких-то атрибутов или списков сущностей, выполнения сложных фильтраций или модификаций данных. Подробнее про формат ошибок смотрите в (#3). Стандартный ответ выглядит так:

```
{“data”: { ... }, “errors”: [ ... ]}
```

где **data** содержит запрошенные данные, а **errors** содержит список ошибок.

Получение организации по идентификатору через Organization

Например, мы хотим получить данные по организации:

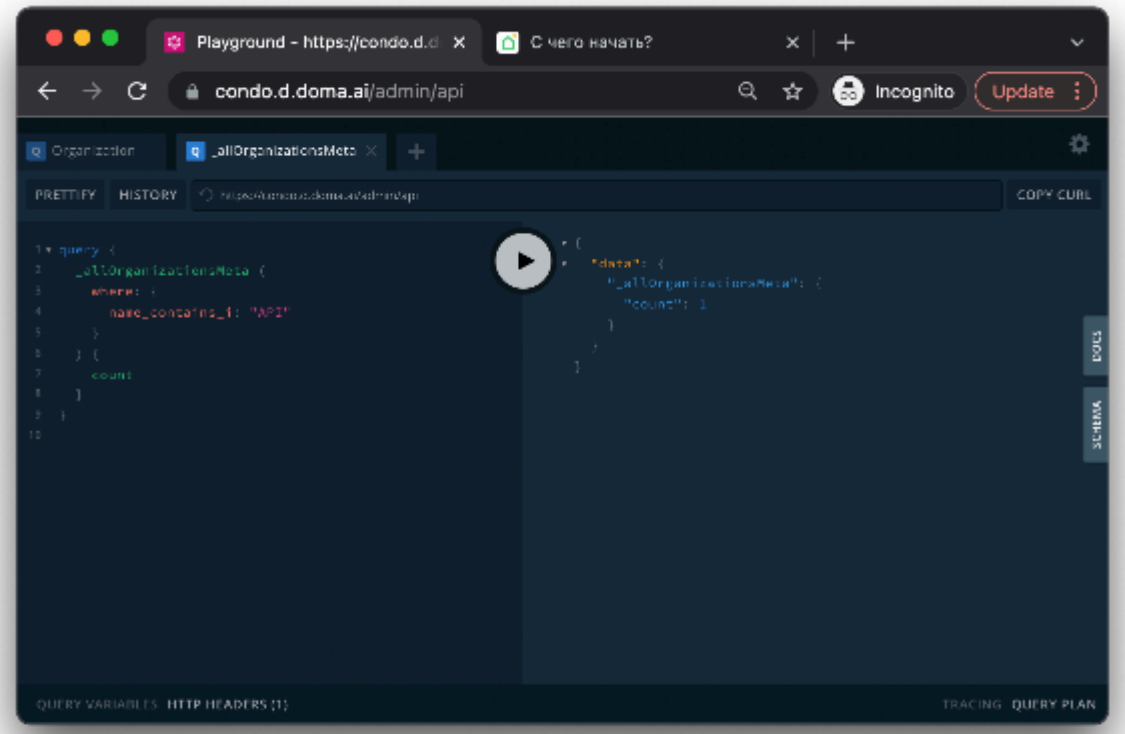


QUERY	RESULT

{
Organization (where: {id: "81cfe-3fcb-495f6cee"}) {
id
__typename
name
}
}
	{
	"data": {
	"Organization": {
	"id": "81cfe-3fcb-495f6cee",
	"__typename": "Organization",
	"name": "OOO DOMA"
	}
	}
	}

Получение кол-ва организаций через _allOrnanizationsMeta

Например, мы хотим посчитать кол-во объектов с определенным именем:



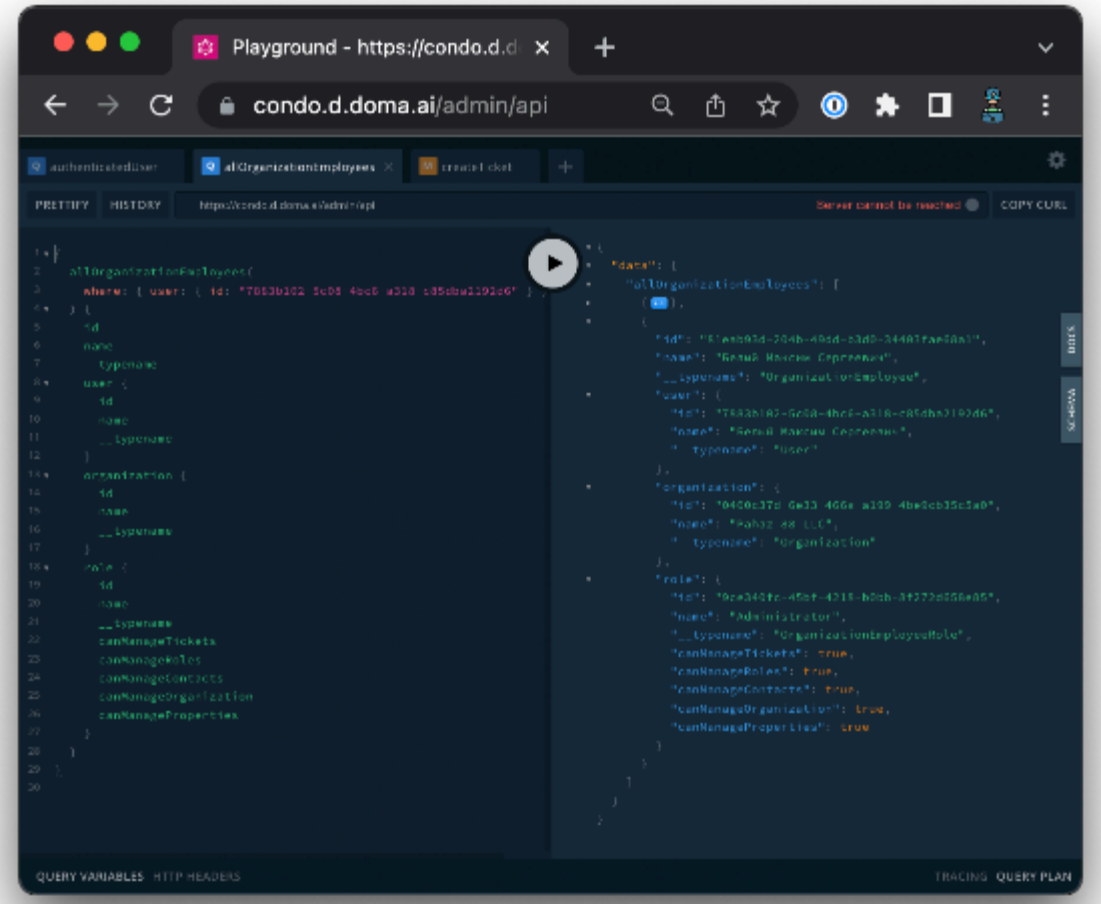
QUERY	RESULT
{ _allOrganizationsMeta (where: {name_contains_i: "OOO"}) { count } }	{ "data": { "_allOrganizationsMeta": { "count": 44, } } }

Данный запрос поддерживает параметры фильтрации как и у allOrganizations.

Список доступных сотруднику организаций с ролями и правами в них (OrganizationEmployees)

Мы научились получать список организаций, по аналогии со списком организаций, можно получить список всех сотрудников через allOrganizationEmployees.

Рассмотрим пример для получения списка доступных организаций с ролями и правами в этих организациях.



QUERY	RESULT
{ allOrganizationEmployees(where: { user: { id: "7883b102-5c08-4bc6-a318-c85dba2192d6" } }) { id name __typename user { id name __typename } organization { id name __typename	{ "data": { "allOrganizationEmployees": [{ "id": "81eab93d-204b-49dd-b3d0-34403fae68a1", "name": "Белый Максим Сергеевич", "__typename": "OrganizationEmployee", "user": { "id": "7883b102-5c08-4bc6-a318-c85dba2192d6", "name": "Белый Максим Сергеевич", "__typename": "User" }, "organization": {

```
    }
    role {
      id
      name
      __typename
      canManageTickets
      canManageRoles
      canManageContacts
      canManageOrganization
      canManageProperties
    }
  }
}
```

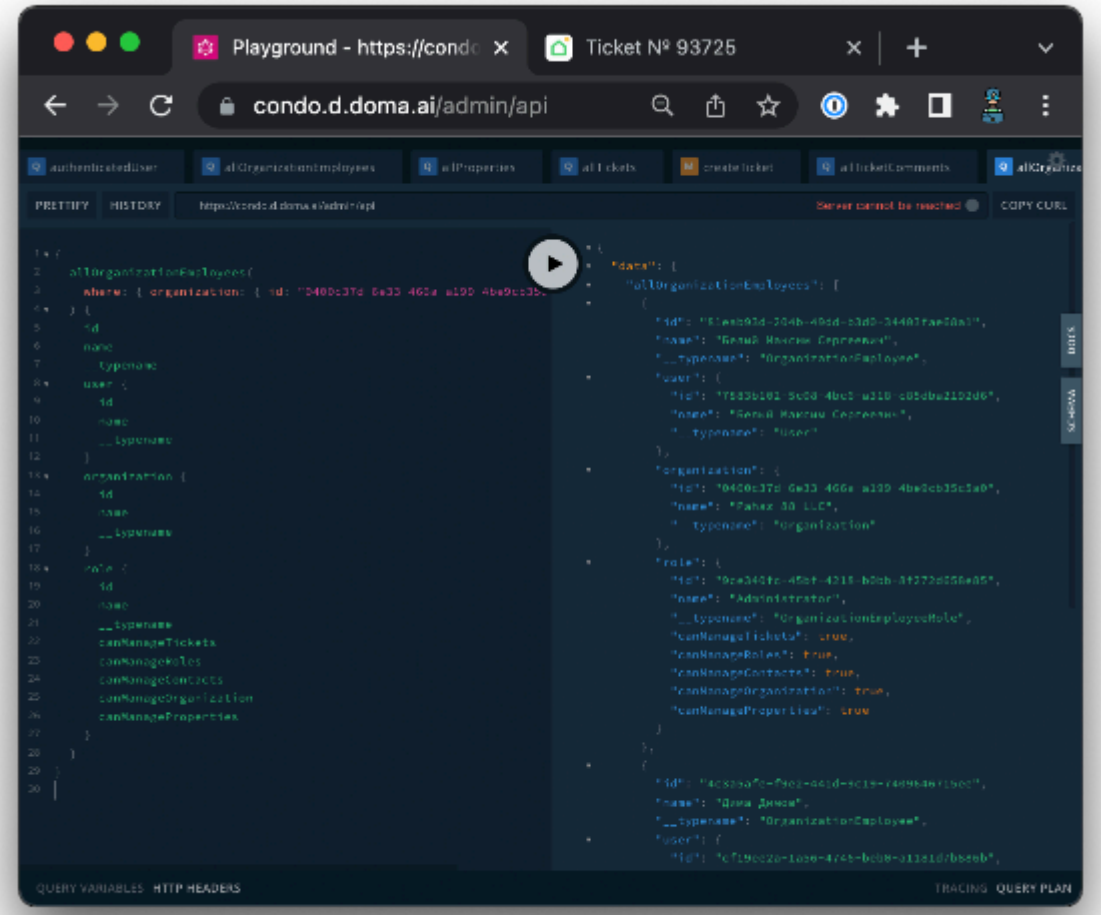
```
        "id":
"0400c37d-6e33-466a-a199-4be9cb35c5a0",
        "name": "Pahaz 88 LLC",
        "__typename": "Organization"
      },
      "role": {
        "id":
"9ce340fc-45bf-4218-b0bb-8f272d658e85",
        "name": "Administrator",
        "__typename": "OrganizationEmployeeRole",
        "canManageTickets": true,
        "canManageRoles": true,
        "canManageContacts": true,
        "canManageOrganization": true,
        "canManageProperties": true
      }
    }
  ]
 }
}
```

Перейдем к другим примерам.

Вы можете убрать фильтрацию по конкретному пользователю, чтобы получить список всех сотрудников во всех, доступных вам организациях.

Список сотрудников в организации (OrganizationEmployees)

Продолжаем работу со списком сотрудников, следующим примером, вы сможете получить список сотрудников для конкретной организации.



QUERY	RESULT
{ allOrganizationEmployees(where: { organization: { id: "0400c37d-6e33-466a-a199-4be9cb35c5a0" } }) { id name __typename user { id name __typename } organization { id name	{ "data": { "allOrganizationEmployees": [{ "id": "81eab93d-204b-49dd-b3d0-34403fae68a1", "name": "Белый Максим Сергеевич", "__typename": "OrganizationEmployee", "user": { "id": "7883b102-5c08-4bc6-a318-c85dba2192d6", "name": "Белый Максим Сергеевич", "__typename": "User" }, "organization": {

```
    __typename
   }
   role {
    id
    name
    __typename
    canManageTickets
    canManageRoles
    canManageContacts
    canManageOrganization
    canManageProperties
   }
  }
}
```

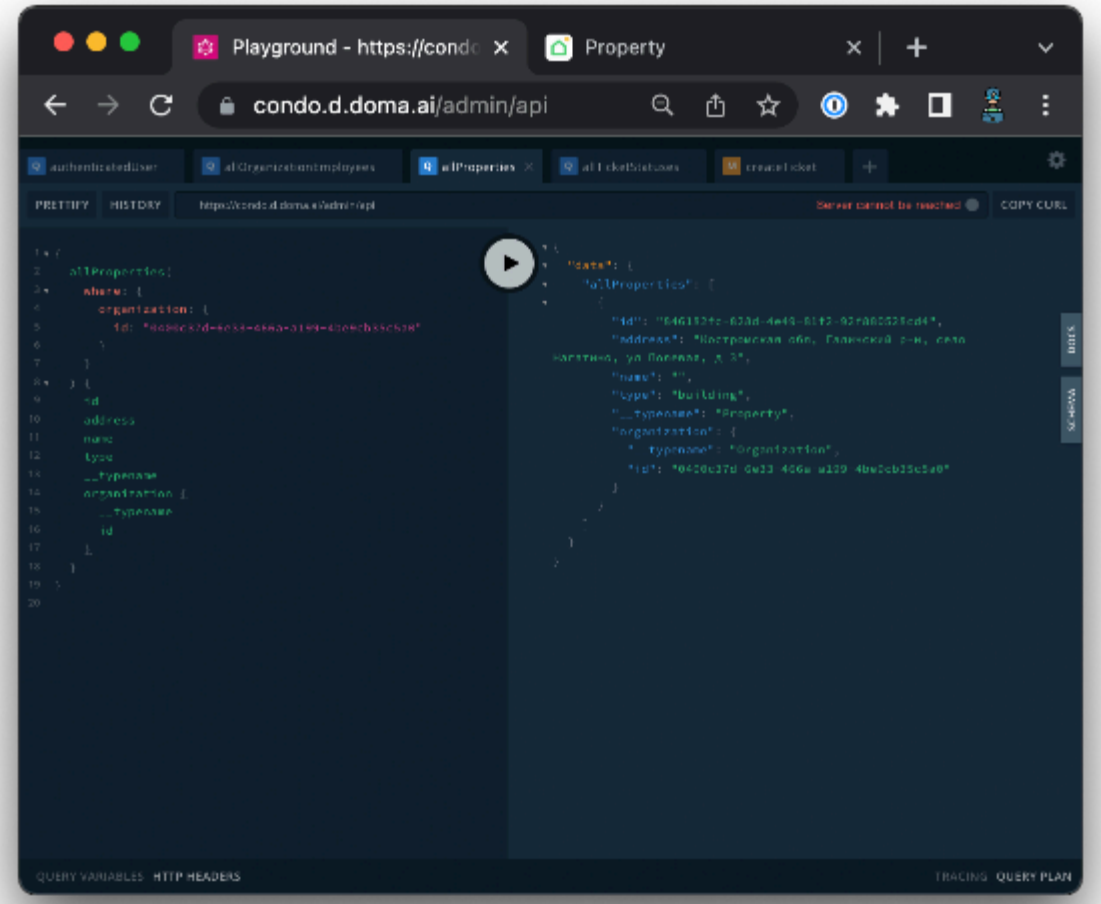
```
        "id":
"0400c37d-6e33-466a-a199-4be9cb35c5a0",
        "name": "Pahaz 88 LLC",
        "__typename": "Organization"
       },
       "role": {
        "id":
"9ce340fc-45bf-4218-b0bb-8f272d658e85",
        "name": "Administrator",
        "__typename": "OrganizationEmployeeRole",
        "canManageTickets": true,
        "canManageRoles": true,
        "canManageContacts": true,
        "canManageOrganization": true,
        "canManageProperties": true
       }
     }
    ]
  }
}
```

Перейдем к другим примерам.

Работа с адресами (Property)

Одной из центральных сущностей нашей доменной модели, являются **объекты общей собственности** (property). Каждый такой объект представляет из себя некоторый адрес, тип объекта общей собственности и некоторую технологическую карту частей общей собственности – шахматку.

На текущий момент мы поддерживаем два типа объектов: поселки и дома.



QUERY	RESULT
{ allProperties(where: { organization: { id: "0400c37d-6e33-466a-a199-4be9cb35c5a0" } }) { id address name type __typename organization { __typename id	{ "data": { "allProperties": [{ "id": "846182fc-828d-4e49-81f2-92f880528cd4", "address": "Костромская обл, Галичский р-н, село Наратино, ул Полевая, д 3", "name": "", "type": "building", "__typename": "Property", "organization": { "__typename": "Organization", "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0" }

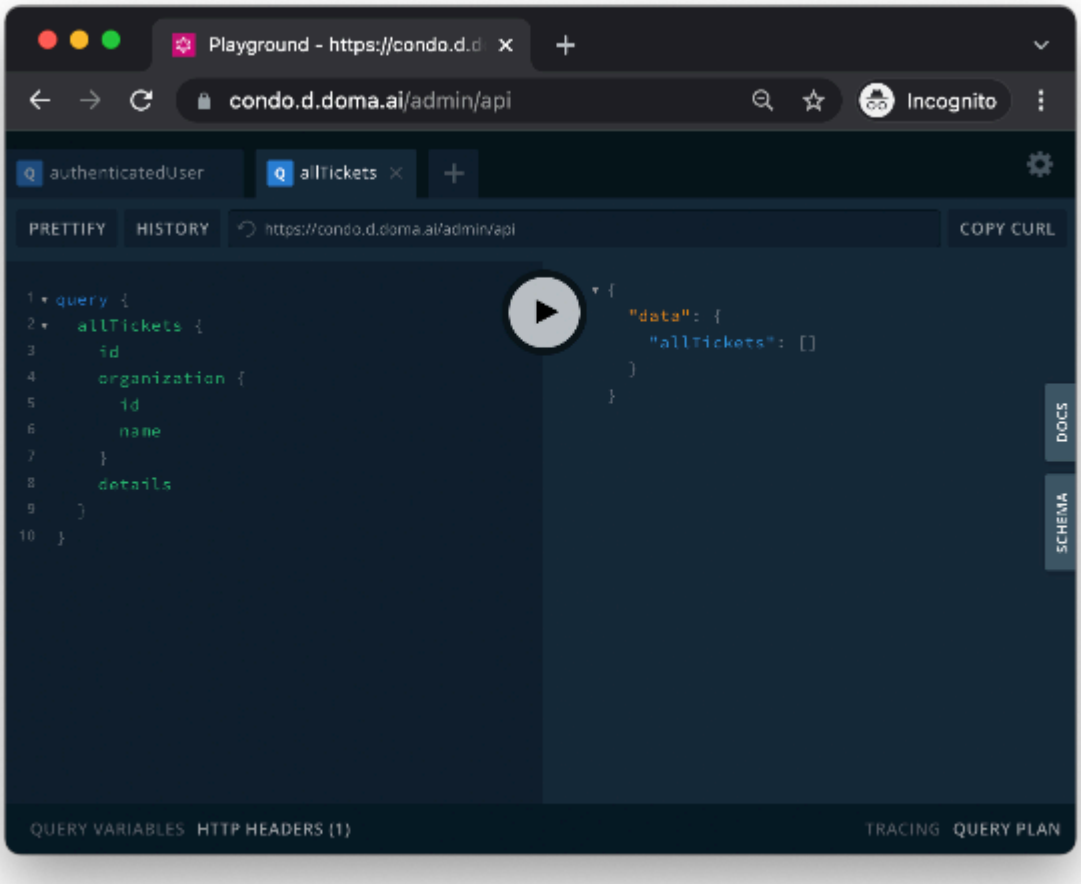
} } }	}] } }

Пример получения списка всех адресов с фильтрацией по организации.

Работа с заявками (Ticket)

Получение списка заявок (allTickets)

Мы только создали тестовую организацию, в ней еще нет ни одного тикета. Попробуем по аналогии со списком организаций получить список заявок.



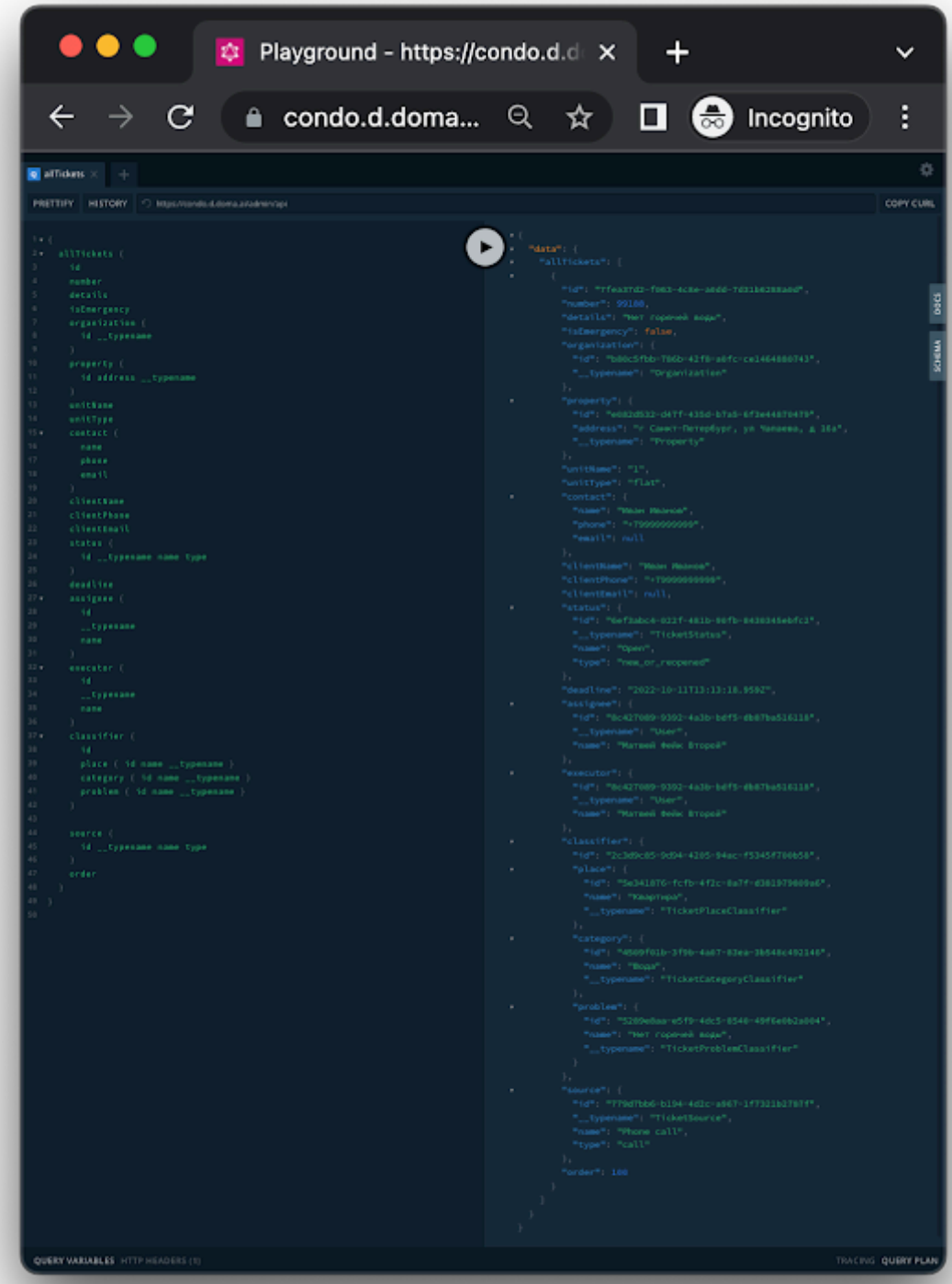
QUERY	RESULT

{
allTickets {
id
organization {
	{
	"data": {
	"allTickets": []
	}

id name } details } }	}

Вы можете перейти в интерфейс https://condo.d.doma.ai/ticket/create и создать там заявку для вашей тестовой организации, после этого вы сможете увидеть ее в этом списке.

Рассмотри пример с большим кол-вом полей.


Playground - https://condo.d.d
condo.d.doma...
Incognito
allTickets
PRETTIFY
HISTORY
COPY CURL
QUERY VARIABLES
HTTP HEADERS (0)
TRACING
QUERY PLAN
DOCS
SCHEMA

QUERY

```
{
  allTickets {
    id
    number
    details
    isEmergency
    organization {
      id __typename
    }
    property {
      id address __typename
    }
    unitName
    unitType
    contact {
      name
      phone
      email
    }
    clientName
    clientPhone
    clientEmail
    status {
      id __typename name type
    }
    deadline
    assignee {
      id
      __typename
      name
    }
    executor {
      id
      __typename
      name
    }
    classifier {
      id
      place { id name __typename }
      category { id name __typename }
      problem { id name __typename }
    }

    source {
      id __typename name type
    }
    order
  }
}
```

RESULT


```
{
  "data": {
    "allTickets": [
      {
        "id": "7fea37d2-f063-4c8e-a0dd-7d31b6288a0d",
        "number": 99188,
        "details": "Нет горячей воды",
        "isEmergency": false,
        "organization": {
          "id": "b80c5fbb-786b-42f8-a0fc-ce1464880743",
          "__typename": "Organization"
        },
        "property": {
          "id": "e082d532-d47f-435d-b7a5-6f3e44870479",
          "address": "г Санкт-Петербург, ул Чапаева, д 16а",
          "__typename": "Property"
        },
        "unitName": "1",
        "unitType": "flat",
        "contact": {
          "name": "Иван Иванов",
          "phone": "+79999999999",
          "email": null
        },
        "clientName": "Иван Иванов",
        "clientPhone": "+79999999999",
        "clientEmail": null,
        "status": {
          "id": "6ef3abc4-022f-481b-90fb-8430345ebfc2",
          "__typename": "TicketStatus",
          "name": "Open",
          "type": "new_or_reopened"
        },
        "deadline": "2022-10-11T13:13:18.959Z",
        "assignee": {
          "id": "8c427089-9392-4a3b-bdf5-db87ba516118",
          "__typename": "User",
          "name": "Матвей Фейк Второй"
        },
        "executor": {
          "id": "8c427089-9392-4a3b-bdf5-db87ba516118",
          "__typename": "User",
          "name": "Матвей Фейк Второй"
        },
        "classifier": {
          "id": "2c3d9c85-9d94-4205-94ac-f5345f700b58",
          "place": {
            "id": "5e341876-fcfb-4f2c-8a7f-d381979809a6",
            "name": "Квартира",
            "__typename": "TicketPlaceClassifier"
          },
          "category": {
            "id": "4509f01b-3f9b-4a07-83ea-3b548c492146",
            "name": "Вода",
            "__typename": "TicketCategoryClassifier"
          },
          "problem": {
```

```
          "id": "5289e8aa-e5f9-4dc5-8540-49f6e0b2a004",
          "name": "Нет горячей воды",
          "__typename": "TicketProblemClassifier"
        }
      },
      "source": {
        "id": "779d7bb6-b194-4d2c-a967-1f7321b2787f",
        "__typename": "TicketSource",
        "name": "Phone call",
        "type": "call"
      },
      "order": 100
    }
  ]
 }
}
```


Некоторые поля: number – номер заявки, isEmergency – признак аварийности заявки, unitName – номер помещения для домов или номер участка для поселков, contact – контакт жителя, status – статус заявки, deadlint – срок выполнения заявки, assignee – ответственный, executor – исполнитель, source – источник появления заявки

Давайте попробуем понять, как работает поиск и фильтрация. Вы можете заметить, что на странице работы со списком заявок, вам доступны обширные возможности фильтрации данных:
https://condo.d.doma.ai/ticket?filters=%7B%22property%22%3A%5B%22846182fc-828d-4e49-81f2-92f880528cd4%22%5D%7D

Давайте попробуем воспользоваться фильтрами, попробуем найти все заявки по конкретной организации, конкретному адресу дома и конкретной квартире.

...

Создание новой заявки (createTicket)

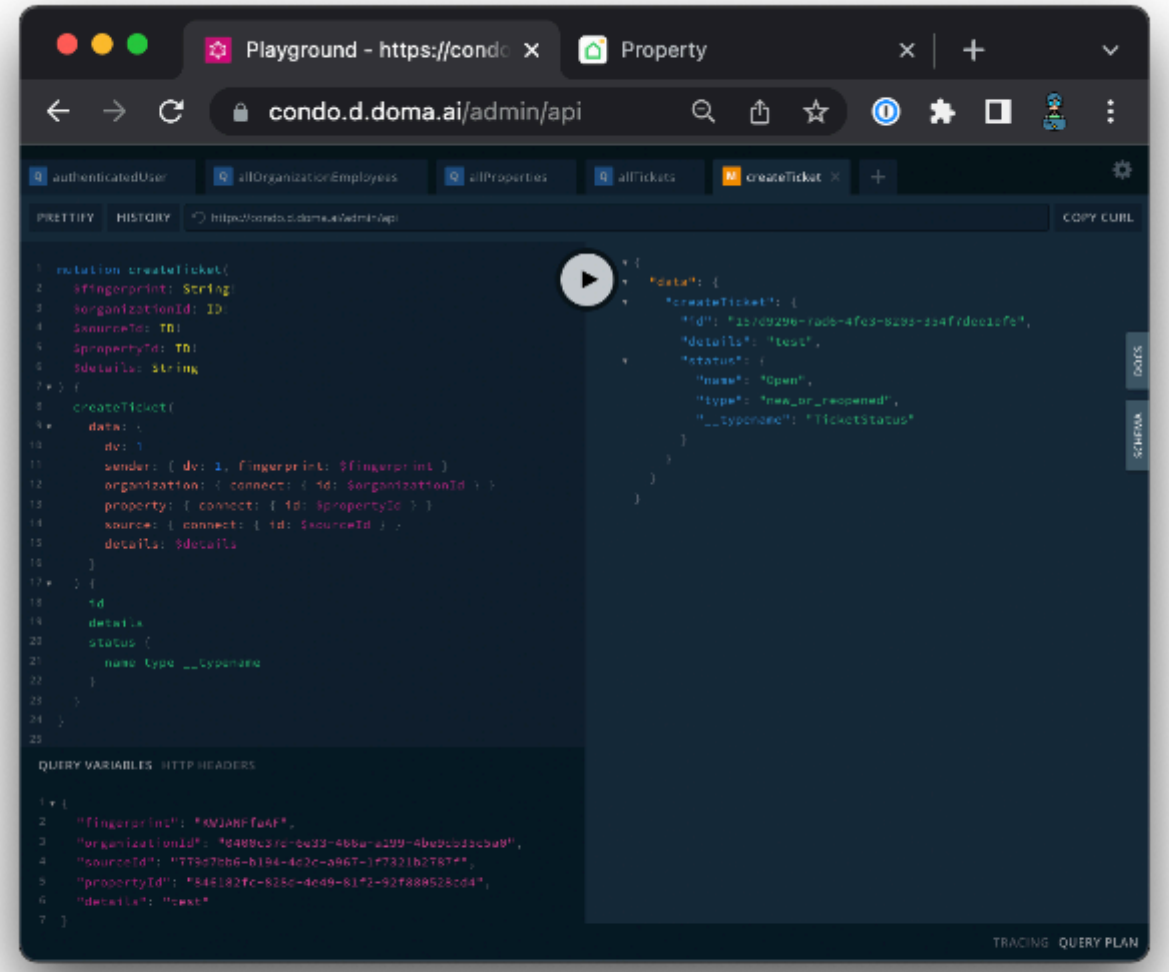
Рассмотрим пример создания заявки. В предыдущих примерах мы рассматривали только запросы на получение данных. Это первый пример, где мы рассмотрим создание.

Для создания заявки, нам нужно передать организацию, адрес объекта общей собственности (где произошла проблема), источник возникновения заявки (звонок, приложение, сообщение в мессенджере, и тп) и текст проблемы.

Есть еще два параметра, которые необходимо передавать с каждым запросом на изменение данных:

- **dv** – версия структуры переданных данных. При изменениях в API, это значение будет изменяться, оно необходимо для версионирования и обеспечения обратной совместимости. Для каждого типа объектов, эта версия будет своя.
- **sender** – вложенный объект, описывающий устройство делающее запросы в наше API. Формат: { “dv”: 1, “fingerprint”: “<device-uid>” }. Мы предполагаем, что структура передаваемых данных в поле **sender** может поменяться со временем, поэтому, внутри вложенного объекта, так же передается поле **dv**. В первой версии (dv = 1), необходимо передать только **fingerprint** – это уникальный идентификатор устройства (идентификатор сервера, телефона, браузера) с которого был выполнен запрос. Используется для детектирования спама и выявления

отклонений в работе клиентов API. Лучше использовать UUID для идентификации устройств делающих запросы. Например, мы используем identifierForVendor для запросов из iOS приложений. Если запросы делаются с какого-то сервера, то можно использовать случайно сгенерированный UUID идентификатор. Подробнее про фингерпринт можно посмотреть на wikipedia.org/Device_fingerprint



QUERY	RESULT

```
mutation createTicket(
  $fingerprint: String!
  $organizationId: ID!
  $sourceId: ID!
  $propertyId: ID!
  $details: String
) {
  createTicket(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      organization: { connect: { id: $organizationId } }
      property: { connect: { id: $propertyId } }
      source: { connect: { id: $sourceId } }
      details: $details
    }
  ) {
    id
    details
    status {
      name type __typename
    }
  }
}
```

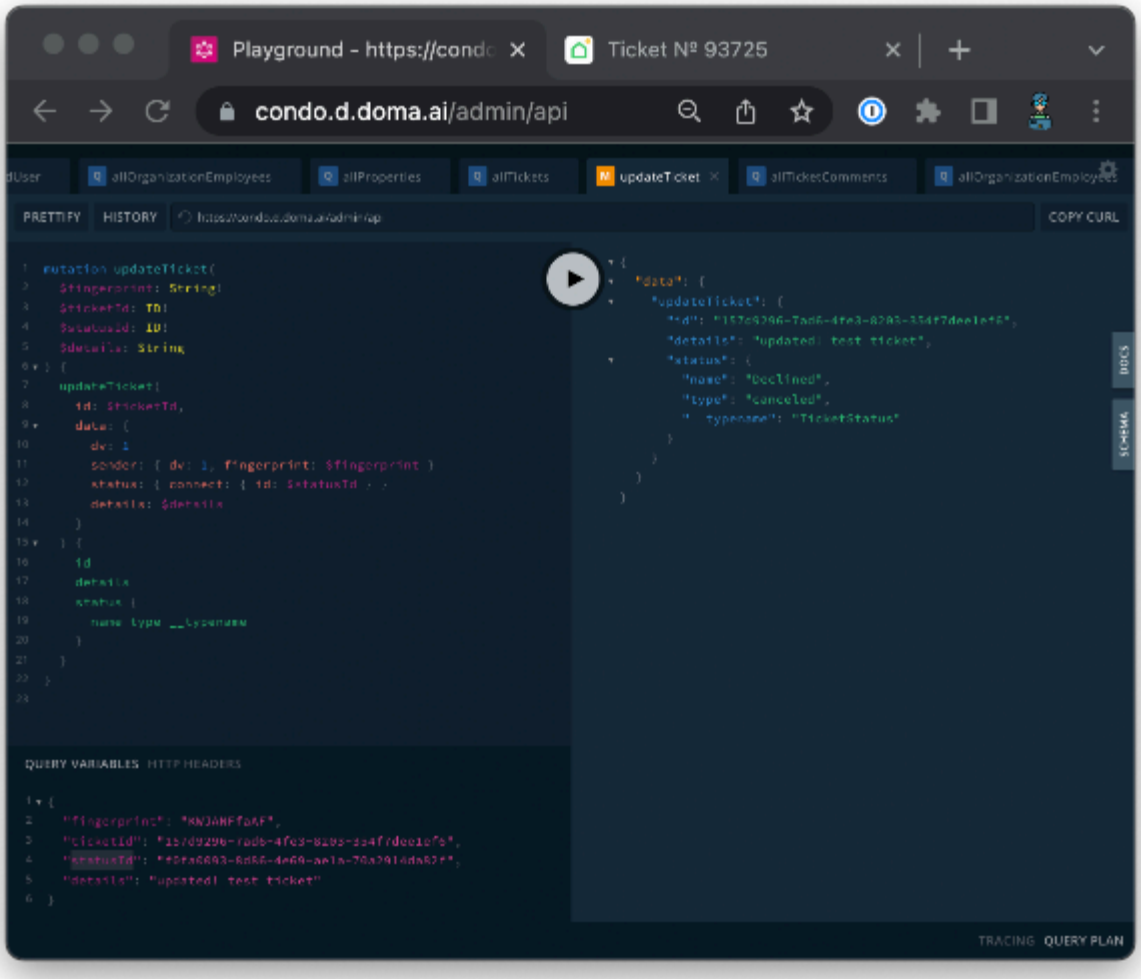
```
{
  "fingerprint": "<device-uid>",
  "organizationId":
"0400c37d-6e33-466a-a199-4be9cb35c5a0",
  "sourceId":
"779d7bb6-b194-4d2c-a967-1f7321b2787f",
  "propertyId":
"846182fc-828d-4e49-81f2-92f880528cd4",
  "details": "test"
}
```

```
{
  "data": {
    "createTicket": {
      "id": "157d9296-7ad6-4fe3-8203-354f7dee1ef6",
      "details": "test",
      "status": {
        "name": "Open",
        "type": "new_or_reopened",
        "__typename": "TicketStatus"
      }
    }
  }
}
```

Кроме минимального набора полей, можно также передавать и другие поля описанные в прошлом разделе.

Изменение заявки (updateTicket)

Рассмотрим пример изменения статуса заявки и обновления текста заявки.


Playground - https://condo
Ticket № 93725
condo.d.doma.ai/admin/api
allOrganizationEmployees
allProperties
allTickets
updateTicket
allTicketComments
allOrganizationEmployees
PRETTIFY
HISTORY
COPY CURL
mutation updateTicket(
$fingerprint: String!
$ticketId: ID!
$statusId: ID!
$details: String
) {
updateTicket(
id: $ticketId,
data: {
dv: 1
sender: { dv: 1, fingerprint: $fingerprint }
status: { connect: { id: $statusId } }
details: $details
}
) {
id
details
status {
name type __typename
}
}
}
{
"data": {
"updateTicket": {
"id": "157c9206-7ad6-4fe3-8203-354f7dee1ef6",
"details": "updated: test ticket",
"status": {
"name": "Declined",
"type": "canceled",
"__typename": "TicketStatus"
}
}
}
}
QUERY VARIABLES
HTTP HEADERS
{
"fingerprint": "KNJAHFOAF",
"ticketId": "157d9206-7ad6-4fe3-8203-354f7dee1ef6",
"statusId": "f4fa6893-6d66-4e69-ae1a-70a291dda82f",
"details": "updated: test ticket"
}
TRACING
QUERY PLAN
DOCS
SCHEMA

QUERY

```
mutation updateTicket(
  $fingerprint: String!
  $ticketId: ID!
  $statusId: ID!
  $details: String
) {
  updateTicket(
    id: $ticketId,
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      status: { connect: { id: $statusId } }
      details: $details
    }
  ) {
    id
    details
    status {
      name type __typename
    }
  }
}
```

```
{
  "fingerprint": "<device-uid>",
  "ticketId":
"157d9296-7ad6-4fe3-8203-354f7dee1ef6",
  "statusId":
"f0fa0093-8d86-4e69-ae1a-70a2914da82f",
  "details": "updated! test ticket"
}
```

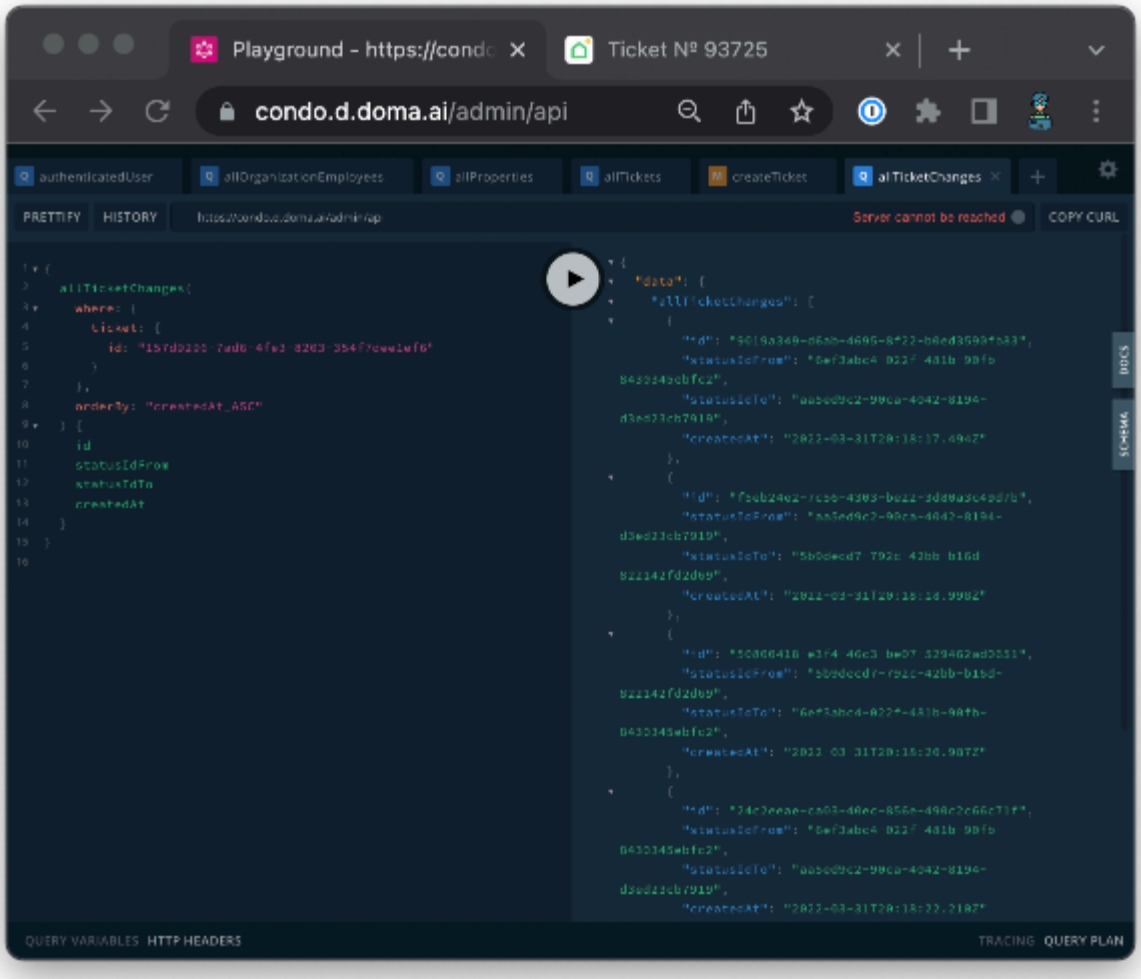
RESULT

```
{
  "data": {
    "updateTicket": {
      "id": "157d9296-7ad6-4fe3-8203-354f7dee1ef6",
      "details": "updated! test ticket",
      "status": {
        "name": "Declined",
        "type": "canceled",
        "__typename": "TicketStatus"
      }
    }
  }
}
```

Можно заметить, что для изменения как и для создания, нужно передавать dv и sender.

Просмотр истории изменений (TicketChanges)

При изменении заявки, автоматически создается объект-слепок состояния. Это позволяет смотреть на историю изменений заявки.



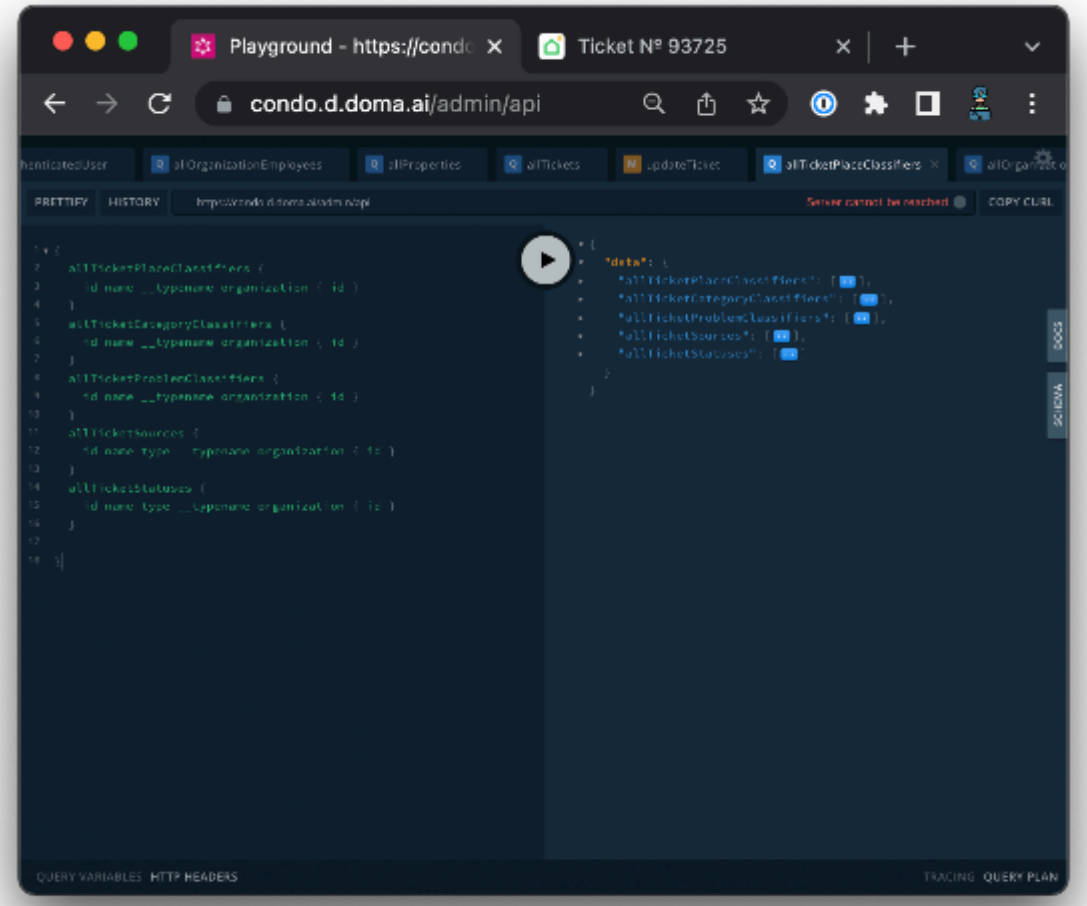
QUERY	RESULT
{ allTicketChanges(where: { ticket: { id: "157d9296-7ad6-4fe3-8203-354f7dee1ef6" } }, orderBy: "createdAt_ASC") { id statusIdFrom statusIdTo createdAt } }	{ "data": { "allTicketChanges": [{ "id": "9019a349-d6ab-4695-8f22-b0ed3590fb83", "statusIdFrom": "6ef3abc4-022f-481b-90fb-8430345ebfc2", "statusIdTo": "aa5ed9c2-90ca-4042-8194-d3ed23cb7919", "createdAt": "2022-03-31T20:18:17.494Z" }, { "id": "f5eb24e2-7c56-4303-be22-3d80a3c49d7b", "statusIdFrom": "aa5ed9c2-90ca-4042-8194-d3ed23cb7919", "statusIdTo": "5b9decd7-792c-42bb-b16d-822142fd2d69", "createdAt": "2022-03-31T20:18:18.998Z" },]

	} }

Пример, истории изменения статусов в заявке с датами изменения

Получение классификатора, списка статусов, источников заявок

По аналогии с предыдущими примерами, можно получить списки: allTicketPlaceClassifiers, allTicketCategoryClassifiers, allTicketProblemClassifiers, allTicketSources, allTicketStatuses



QUERY	RESULT

{
allTicketClassifiers (first: 100) {
place { id name }
category { id name }
problem { id name }
}
allTicketPlaceClassifiers {
	{
	"data": {
	“allTicketClassifiers”: [
	],
	"allTicketPlaceClassifiers": [
	],
	"allTicketCategoryClassifiers": [

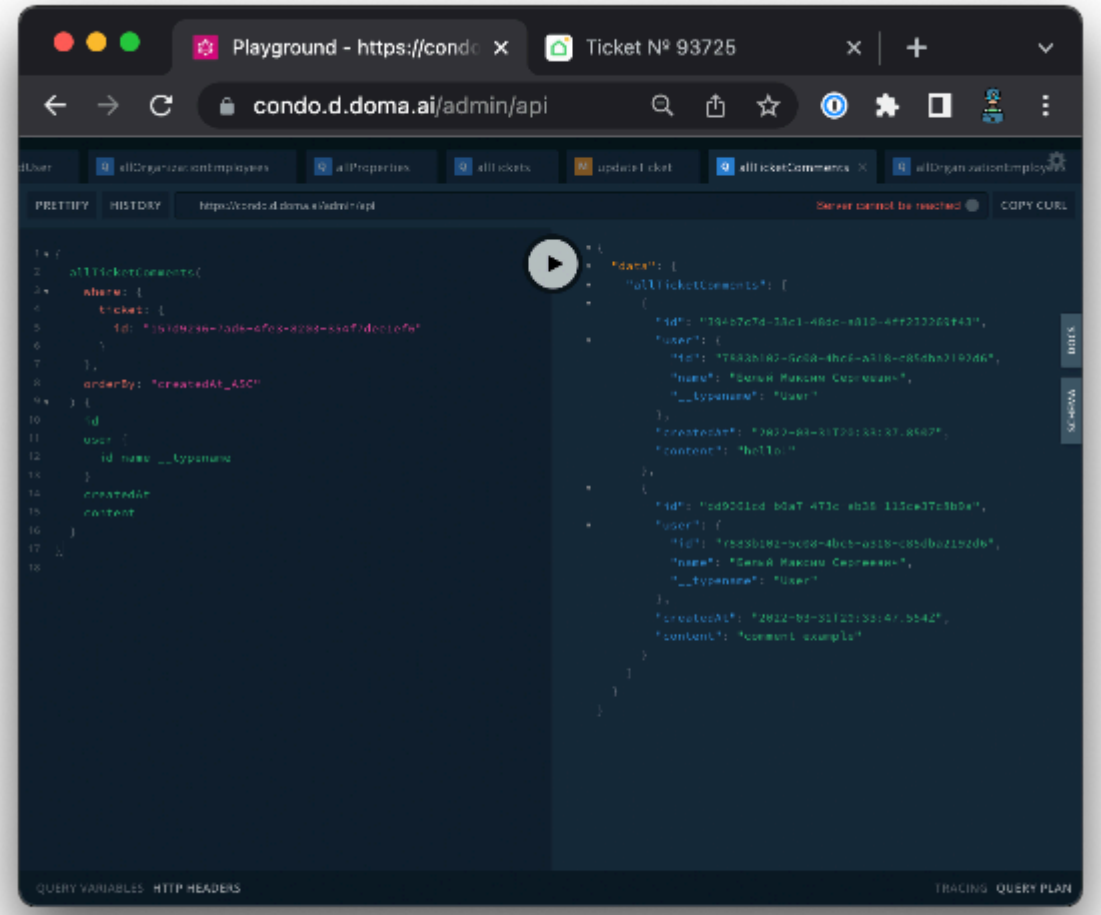
```
    id name __typename organization { id }
  }
  allTicketCategoryClassifiers {
    id name __typename organization { id }
  }
  allTicketProblemClassifiers {
    id name __typename organization { id }
  }
  allTicketSources {
    id name type __typename organization { id }
  }
  allTicketStatuses {
    id name type __typename organization { id }
  }
}
```

```
    ],
    "allTicketProblemClassifiers": [

    ],
    "allTicketSources": [

    ],
    "allTicketStatuses": [

    ]
  }
}
```

Можете самостоятельно написать запросы для других связанных с тикетом объектов.

Просмотр списка комментариев (allTicketComments)

Получим список комментариев и пользователей которые его оставили.



QUERY	RESULT

{
allTicketComments(
where: {
ticket: {
id: "157d9296-7ad6-4fe3-8203-354f7dee1ef6"
}
},
orderBy: "createdAt_ASC"
) {
id
type
user {
id name __typename
}
createdAt
content
	{
	"data": {
	"allTicketComments": [
	{
	"id": "394b7c7d-38c1-48dc-a810-4ff232269f43",
	"type": "organization",
	"user": {
	"id": "7883b102-5c08-4bc6-a318-c85dba2192d6",
	"name": "Белый Максим Сергеевич",
	"__typename": "User"
	},
	"createdAt": "2022-03-31T20:33:37.850Z",
	"content": "hello!"
	}
	]
	}

} }	}

Список отсортирован по дате создания комментария.

Все комментарии делятся на “внутренние” (type = organization) и “с жителем” (type = resident). Первые видны только сотрудникам УК, а вторые будут видны жителям в мобильном приложении.

Прочтение комментариев к заявке

В момент прочтения комментария каким-то пользователем, он должен создать или обновить UserTicketCommentReadTime объект. При помощи мутации createUserTicketCommentReadTime или updateUserTicketCommentReadTime.

Все комментарии делятся на “внутренние” и “с жителем”. Первые видны только сотрудникам УК, а вторые будут видны жителям в мобильном приложении.

Для работы с этими мутациями необходимо передать следующие обязательные поля:

- ticket – заявка, комментарии который были прочитаны
- user – пользователь, который прочитал комментарии в заявке
- readCommentAt – дата и время прочтения пользователем комментариев

Если, пользователь прочитал вкладку комментариев “с жителем”, то необходимо передать еще одно поле:

- readResidentCommentAt – дата и время прочтения пользователем комментариев с жителем (TicketComment.type = resident)

NOTE: Отдельного поля для прочтения вкладки “внутренние” комментарии не предусмотрено. Если значение readCommentAt будет больше чем readResidentCommantAt, то значит была прочитана только вкладка “внутренние” комментарии. Если значение readCommentAt совпадает с readResidentCommantAt, значит была прочитана вкладка комментарии “с жителем”.

Получить последнее время прочтения комментариев в заявке можно с помощью allUserTicketCommentReadTimes. Если для пользователя и заявки уже существует UserTicketCommentReadTime, то, при новом прочтении комментариев, время прочтения нужно обновить используя updateUserTicketCommentReadTime. Нельзя создать два объекта UserTicketCommentReadTime с одинаковым значением полей user и ticket.

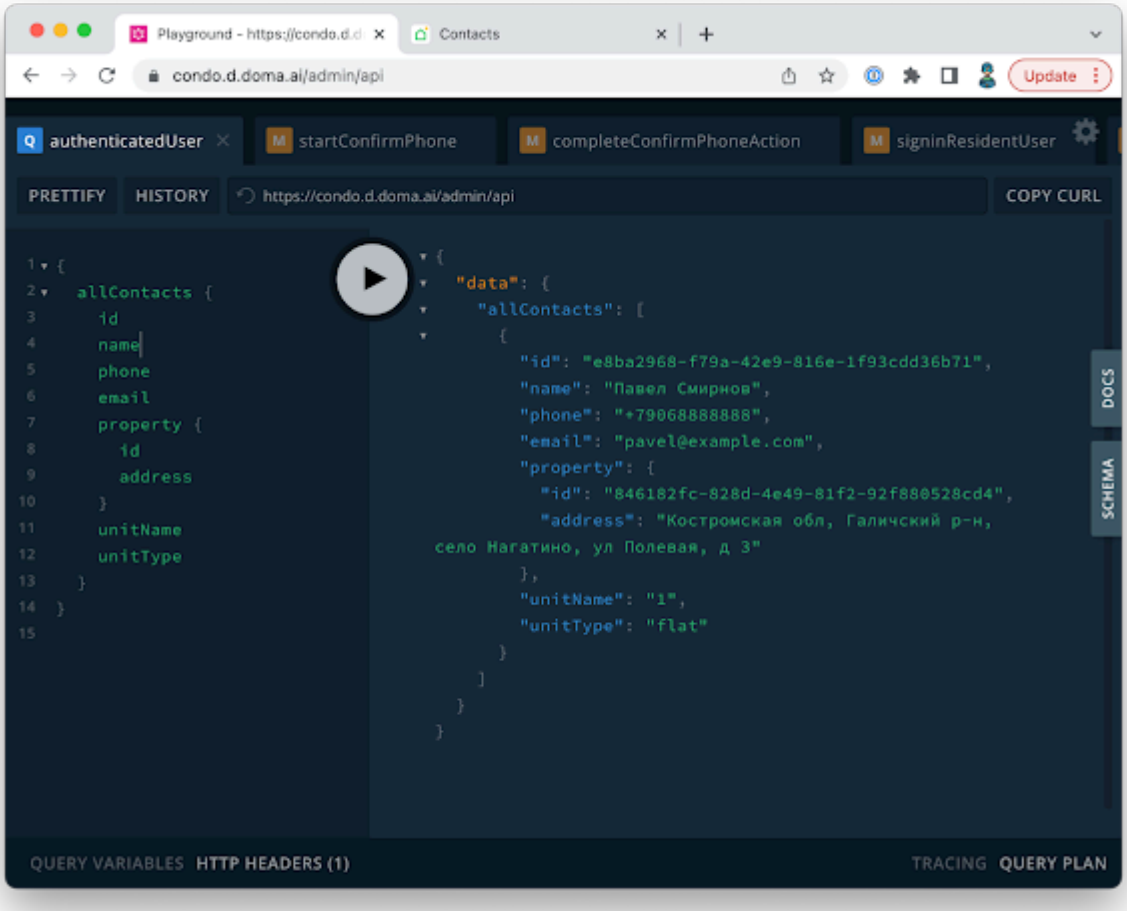
Работа с контактами жителей (Contact)

Управляющая компания ведет реестр контактов жителей. Реестр доступен по адресу: https://condo.d.doma.ai/contact

Контакт жителя включает в себя набор обязательных полей: Имя (name), Телефон (phone), Адрес дома (ссылка на Property), Номер помещения (unitName), Тип помещения (unitType). Все контакты привязываются к организации (ссылка на Organizations)

Помимо обязательных полей, есть еще дополнительные: Роль (ссылка на ContactRole), Эл. почту (email).

Получение списка контактов (allContacts)



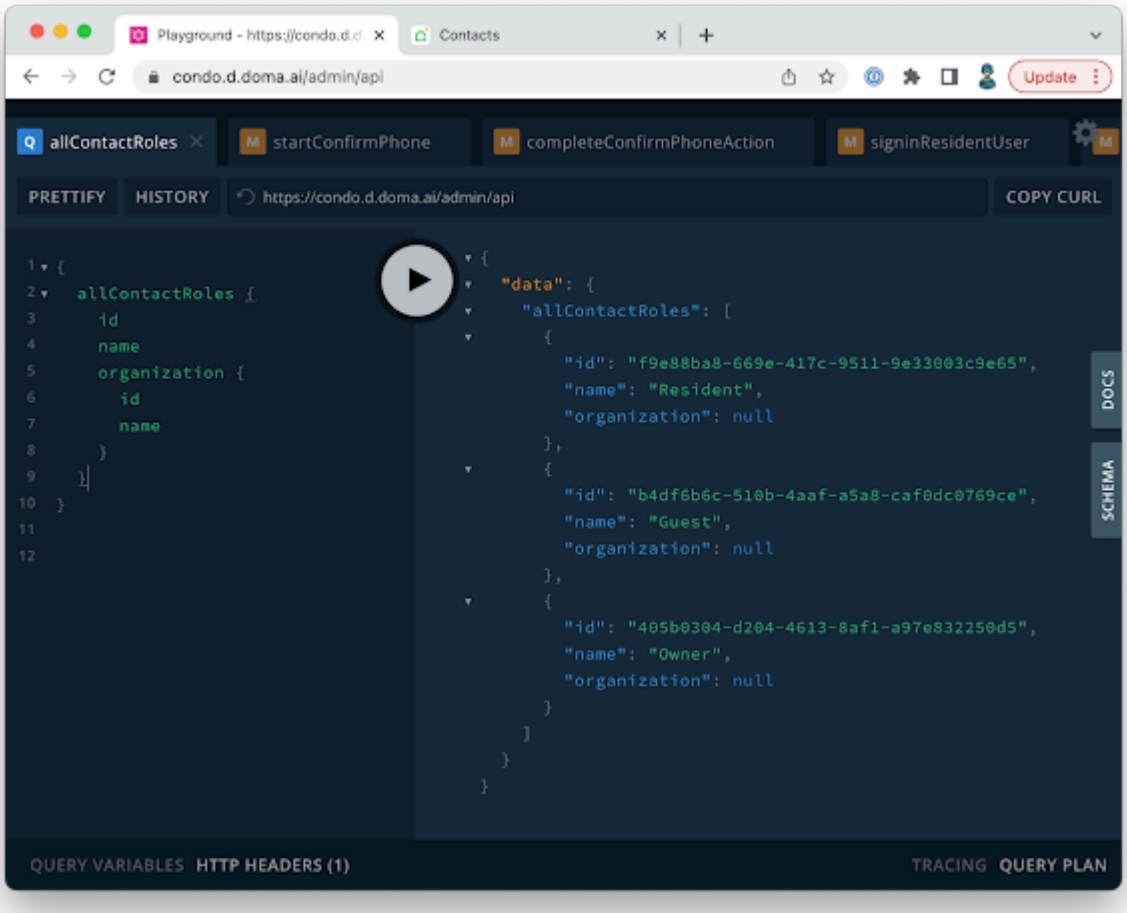
QUERY	RESULT
{ allContacts { id name phone email property { id address } unitName unitType } }	{ "data": { "allContacts": [{ "id": "e8ba2968-f79a-42e9-816e-1f93cdd36b71", "name": "Павел Смирнов", "phone": "+79068888888", "email": "pavel@example.com", "property": { "id": "846182fc-828d-4e49-81f2-92f880528cd4", "address": "Костромская обл, Галичский р-н, село Нагатино, ул Полевая, д 3" }, "unitName": "1", "unitType": "flat" }] } }

Получение списка ролей контактов жителей (allContactRoles)

Каждому контакту жителя, можно установить не обязательный атрибут – роль. Этот атрибут не участвует в бизнес логике приложения. Атрибут позволяет управляющей компании организовать бизнес процесс работы с контактами.

По умолчанию у всех организаций есть список из трех ролей: Собственник, Житель и Гость. Эти роли можно использовать, но нельзя редактировать. Но, через API можно создать роли, привязанные к организации. Эти роли можно редактировать.

Вы можете посмотреть список доступных ролей на странице настроек: https://condo.d.doma.ai/settings?tab=contactRoles



QUERY	RESULT
{ allContactRoles { id name organization { id name } }	{ "data": { "allContactRoles": [{ "id": "f9e88ba8-669e-417c-9511-9e33003c9e65", "name": "Житель", "organization": null },

<table>
<tr>
<td>

```
  }
}
```

</td>
<td>

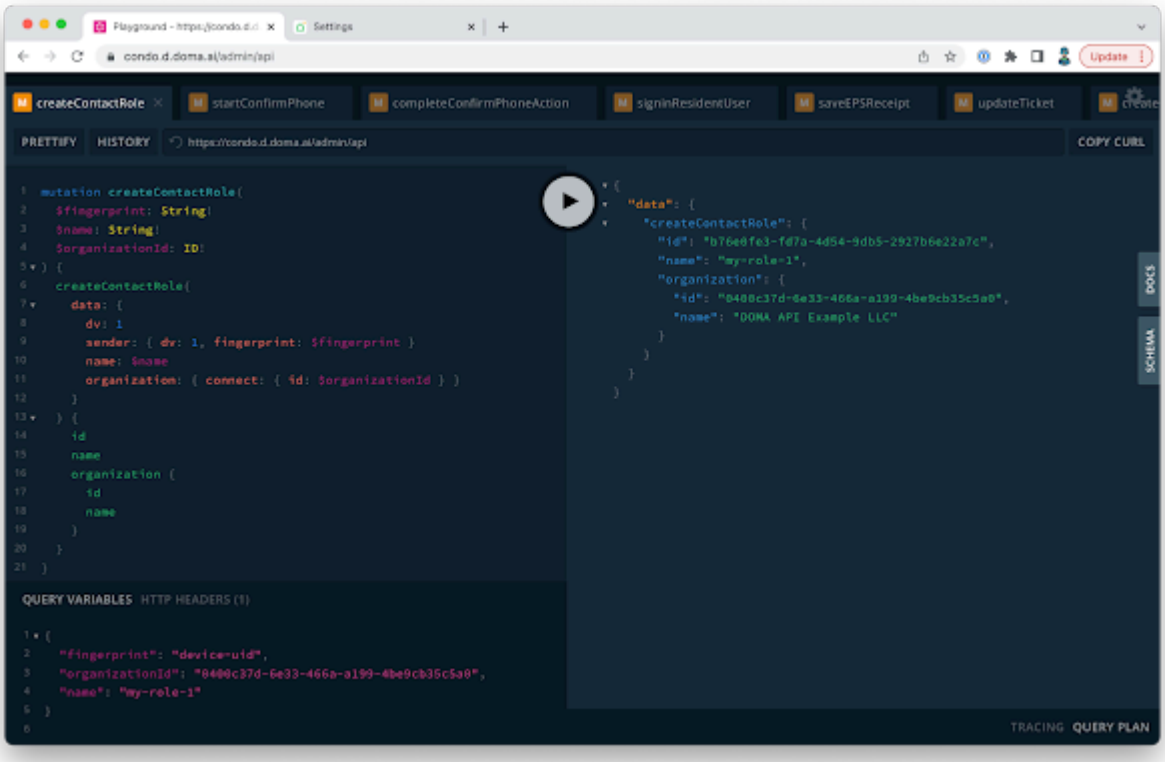
```
    {
      "id": "b4df6b6c-510b-4aaf-a5a8-caf0dc0769ce",
      "name": "Гость",
      "organization": null
    },
    {
      "id": "405b0304-d204-4613-8af1-a97e832250d5",
      "name": "Собственник",
      "organization": null
    }
  ]
 }
}
```

</td>
</tr>
</table>

Создание новой роли для контактов жителей (createContactRole)

Вы можете создать свои роли только в рамках своей организации.

Вы также можете создать новую роль через интерфейс:
https://condo.d.doma.ai/settings?tab=contactRoles



QUERY	RESULT

```
mutation createContactRole(
  $fingerprint: String!
  $name: String!
  $organizationId: ID!
) {
  createContactRole(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      name: $name
      organization: { connect: { id: $organizationId } }
    }
  ) {
    id
    name
    organization {
      id
      name
    }
  }
}
```

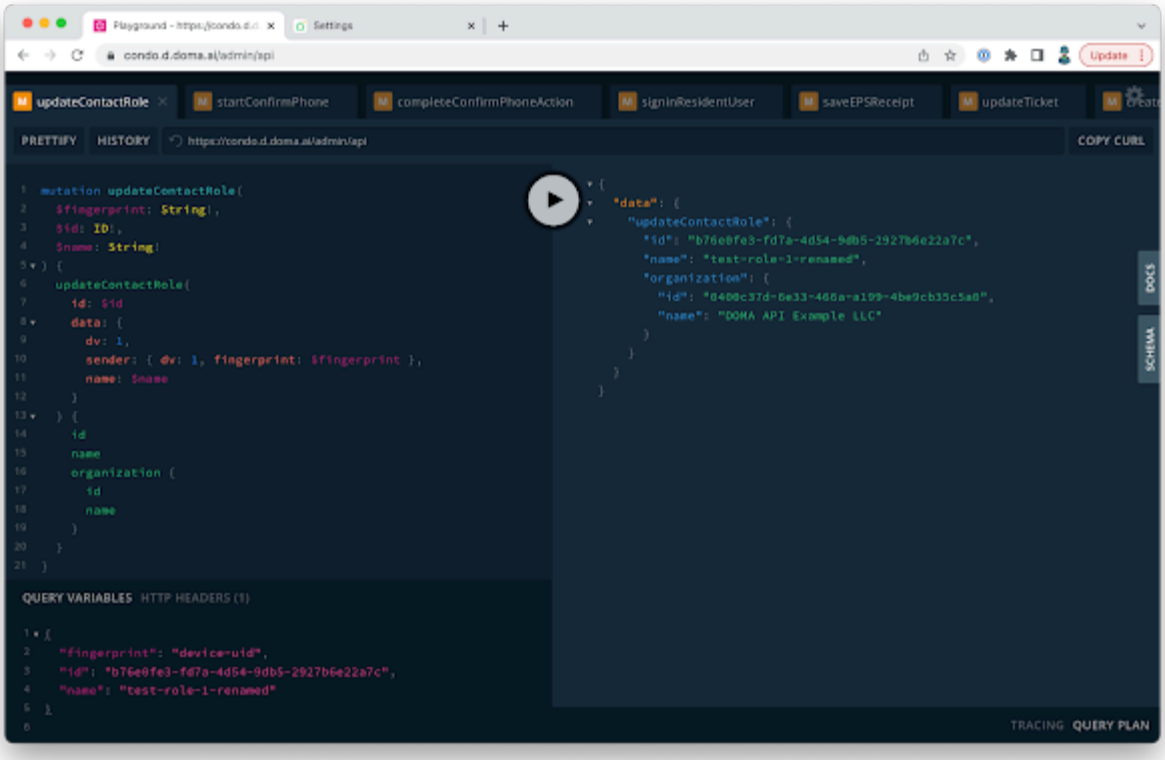
```
{
  "fingerprint": "<device-uid>",
  "organizationId":
"d7225c73-df68-4058-929e-40d3bafdf86e",
  "name": "my-role-1"
}
```

```
{
  "data": {
    "createContactRole": {
      "id": "b76e0fe3-fd7a-4d54-9db5-2927b6e22a7c",
      "name": "my-role-1",
      "organization": {
        "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
        "name": "DOMA API Example LLC"
      }
    }
  }
}
```

Обновление созданной роли для контактов жителей (updateContactRole)

Для обновления роли нужно передать идентификатор редактируемой роли и изменяемые данные. По аналогии с другими примерами обновления данных.

Вы также можете сделать это через интерфейс:
https://condo.d.doma.ai/settings?tab=contactRoles



QUERY	RESULT

mutation updateContactRole(
  $fingerprint: String!,
  $id: ID!,
  $name: String!
) {
  updateContactRole(
  id: $id
  data: {
  dv: 1,
  sender: { dv: 1, fingerprint: $fingerprint },
  name: $name
  }
) {
  id
  name
  organization {
  id
  name
  }
  }
}
	{
	  "data": {
	  "updateContactRole": {
	  "id": "b76e0fe3-fd7a-4d54-9db5-2927b6e22a7c",
	  "name": "test-role-1-renamed",
	  "organization": {
	  "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
	  "name": "DOMA API Example LLC"
	  }
	  }
	  }
	}

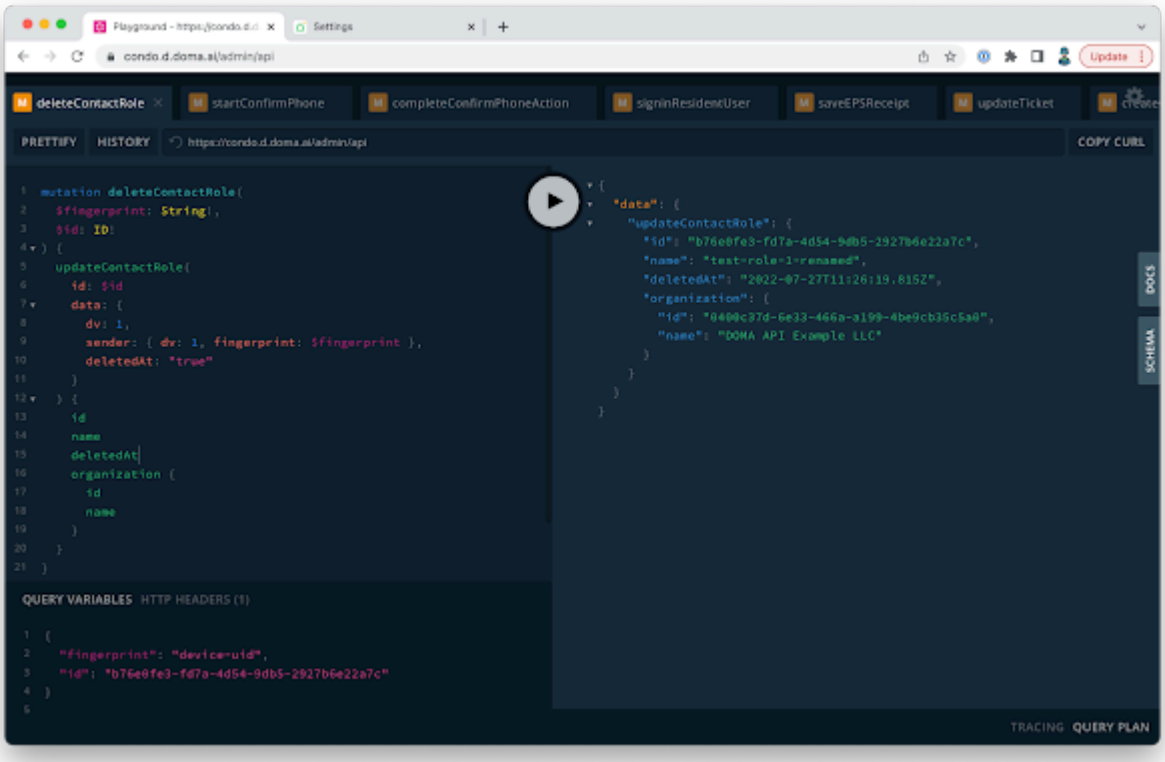
{
  "fingerprint": "<device-uid>",
  "id": "b76e0fe3-fd7a-4d54-9db5-2927b6e22a7c",
  "name": "test-role-1-renamed"	

}	

Удаление созданной роли контакта жителя (updateContactRole)

Для удаления роли контакта жителя, нужно обновить её передав поле deletedAt со значением “true”. После удаления, эту роль нельзя будет получить в списке ролей. Она будет недоступна для отображения!

Вы также можете сделать это через интерфейс:
https://condo.d.doma.ai/settings?tab=contactRoles



QUERY	RESULT

```
mutation deleteContactRole(
  $fingerprint: String!,
  $id: ID!
) {
  updateContactRole(
    id: $id
    data: {
      dv: 1,
      sender: { dv: 1, fingerprint: $fingerprint },
      deletedAt: "true"
    }
  ) {
    id
    name
    deletedAt
    organization {
      id
      name
    }
  }
}
```

```
{
  "fingerprint": "<device-uid>",
  "id": "b76e0fe3-fd7a-4d54-9db5-2927b6e22a7c"
}
```

```
{
  "data": {
    "updateContactRole": {
      "id": "b76e0fe3-fd7a-4d54-9db5-2927b6e22a7c",
      "name": "test-role-1-renamed",
      "deletedAt": "2022-07-27T11:26:19.815Z",
      "organization": {
        "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
        "name": "DOMA API Example LLC"
      }
    }
  }
}
```

Создание контакта жителя (createContact)

Контакт жителя включает в себя набор обязательных полей: Имя (name), Телефон (phone), Адрес дома (ссылка на Property), Номер помещения (unitName), Тип помещения (unitType). Все контакты привязываются к организации (ссылка на Organizations).

Данная модель является связкой адреса + контактная информация + доп. параметры:

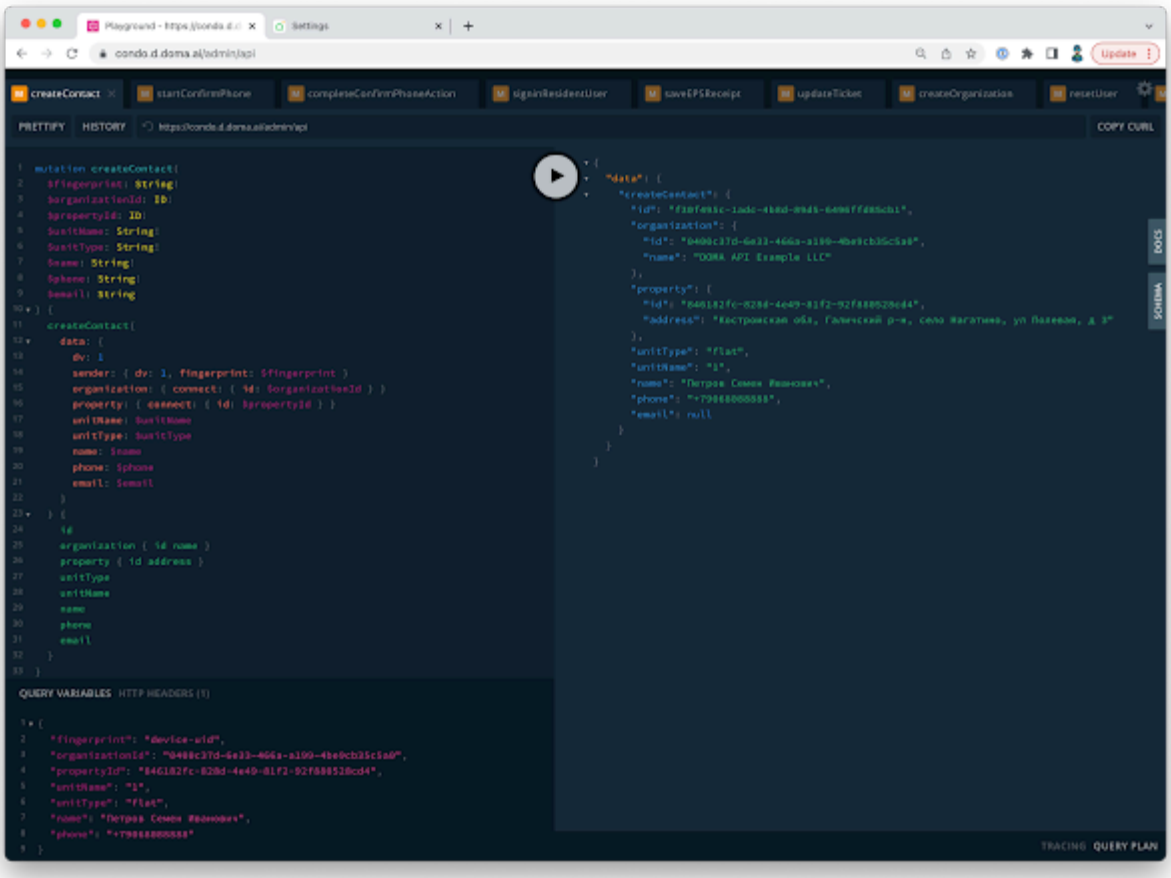
- адрес (адрес дома + номер и тип помещения) +
- контактную информацию (имя + номер телефона + эл. почта)
- доп. параметры (роль контакта)

Допустимые значения для номера и типов помещений указываются в момент создания шахматки дом (Property). Попробуйте самостоятельно создать объект Property через интерфейс на странице https://condo.d.doma.ai/property?tab=buildings и добавить к нему “шахматку”. Мы пока не поддерживаем создание шахматок домов через API.

Какие бывают типы помещений:

- `'flat' – квартира`
- `'parking' - парковочное место`
- `'apartment' - апартаменты`
- `'commercial' - коммерческое помещение`
- `'warehouse' - кладовая комната`

Посмотрим пример создания контакта для имеющегося в системе адреса дома с созданной шахматкой.



QUERY	RESULT

mutation createContact(
  $fingerprint: String!
  $organizationId: ID!
  $propertyId: ID!
  $unitName: String!
  $unitType: String!
  $name: String!
  $phone: String!
  $email: String
) {
  createContact(
  data: {
  dv: 1
  sender: { dv: 1, fingerprint: $fingerprint }
  organization: { connect: { id: $organizationId } }
  property: { connect: { id: $propertyId } }
  unitName: $unitName
  unitType: $unitType
  name: $name
  phone: $phone
  email: $email
  }
) {
	{
	  "data": {
	  "createContact": {
	  "id": "f30f495c-1adc-4b8d-89d5-6496ffd85cb1",
	  "organization": {
	  "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
	  "name": "DOMA API Example LLC"
	  },
	  "property": {
	  "id": "846182fc-828d-4e49-81f2-92f880528cd4",
	  "address": "Костромская обл, Галичский р-н, село Нагатино, ул Полевая, д 3"
	  },
	  "unitType": "flat",
	  "unitName": "1",
	  "name": "Петров Семен Иванович",
	  "phone": "+79068088888",
	  "email": null
	  }
	  }
	}

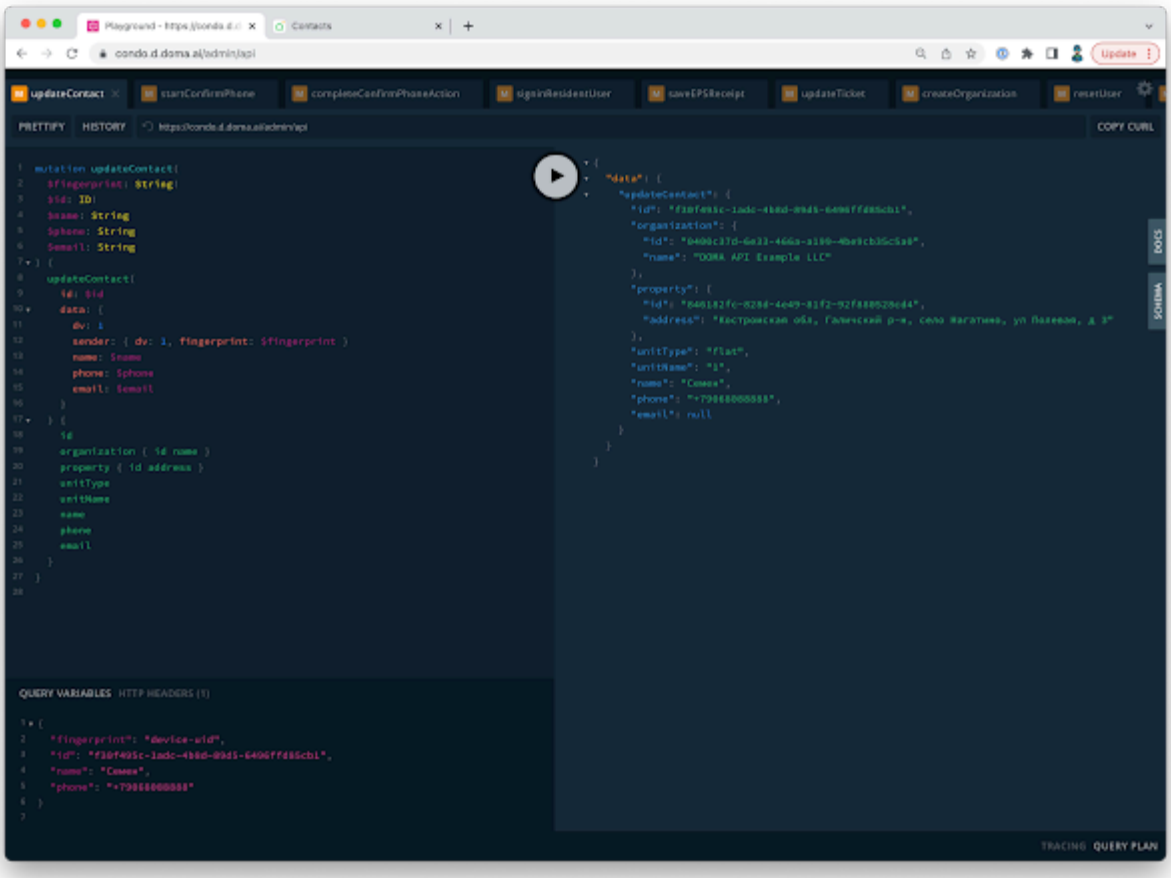
id organization { id name } property { id address } unitType unitName name phone email } }	

{
"fingerprint": "<device-uid>",
"organizationId":
"0400c37d-6e33-466a-a199-4be9cb35c5a0",
"propertyId": "846182fc-828d-4e49-81f2-92f880528cd4",
"unitName": "1",
"unitType": "flat",
"name": "Петров Семен Иванович",
"phone": "+79068088888"
}	

Обновление контакта жителя (updateContact)

Для обновления контакта жителя нужно передать идентификатор контакта, который вы хотите изменить и обновленные данные. По аналогии с другими примерами обновления данных.

Вы также можете сделать это через интерфейс: https://condo.d.doma.ai/contact



QUERY	RESULT

mutation updateContact(
  $fingerprint: String!
  $id: ID!
  $name: String
  $phone: String
  $email: String
) {
  updateContact(
  id: $id
  data: {
  dv: 1
  sender: { dv: 1, fingerprint: $fingerprint }
  name: $name
  phone: $phone
  email: $email
  }
) {
  id
  organization { id name }
  property { id address }
  unitType
  unitName
  name
	{
	  "data": {
	  "updateContact": {
	  "id": "f30f495c-1adc-4b8d-89d5-6496ffd85cb1",
	  "organization": {
	  "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
	  "name": "DOMA API Example LLC"
	  },
	  "property": {
	  "id": "846182fc-828d-4e49-81f2-92f880528cd4",
	  "address": "Костромская обл, Галичский р-н, село Нагатино, ул Полевая, д 3"
	  },
	  "unitType": "flat",
	  "unitName": "1",
	  "name": "Семен",
	  "phone": "+79068088888",
	  "email": null
	  }
	  }
	}

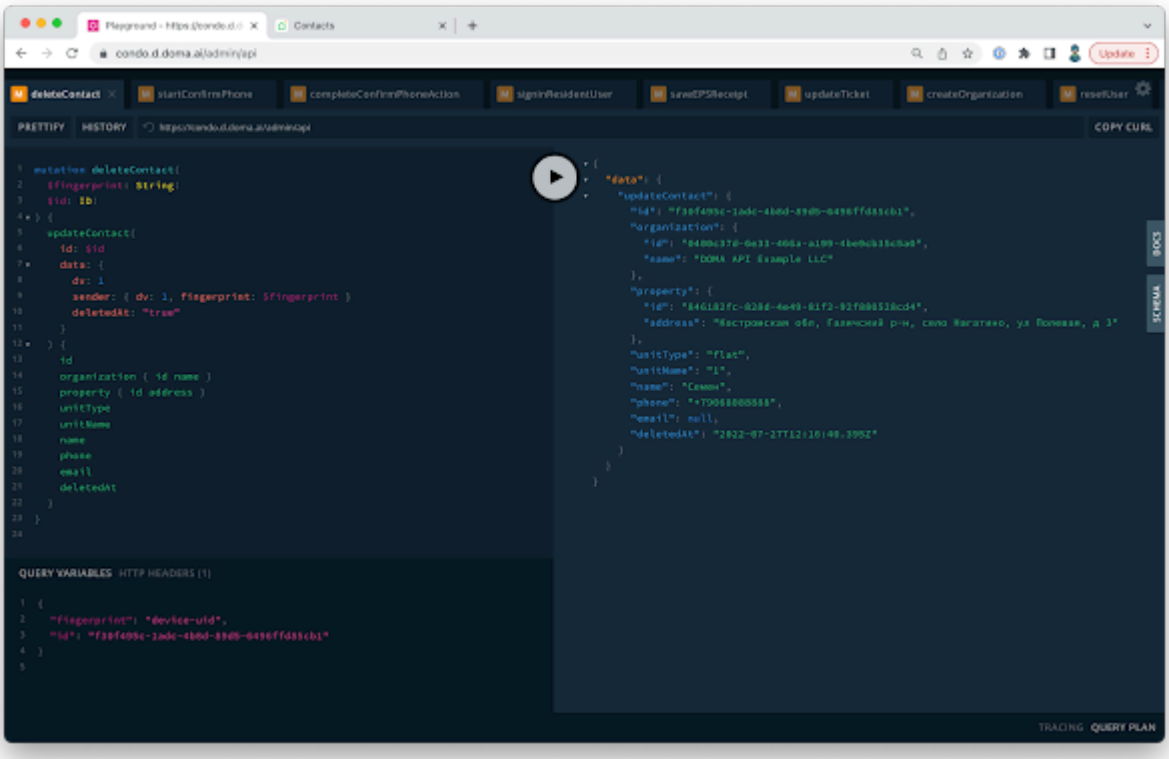
phone email } }	

{
"fingerprint": "<device-uid>",
"id": "f30f495c-1adc-4b8d-89d5-6496ffd85cb1",
"name": "Семен",
"phone": "+79068088888"
}	

Удаление контакта жителя (updateContact)

Для удаления контакта жителя, нужно обновить её передав поле deletedAt со значением “true”. После удаления, этот контакт уже нельзя будет получить в списке контактов. Он будет недоступна для отображения!

Вы также можете сделать это через интерфейс: https://condo.d.doma.ai/contact



QUERY	RESULT

```
mutation deleteContact(
  $fingerprint: String!
  $id: ID!
) {
  updateContact(
    id: $id
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      deletedAt: "true"
    }
  ) {
    id
    organization { id name }
    property { id address }
    unitType
    unitName
    name
    phone
    email
    deletedAt
  }
}
```

```
{
  "fingerprint": "<device-uid>",
  "id": "f30f495c-1adc-4b8d-89d5-6496ffd85cb1"
}
```

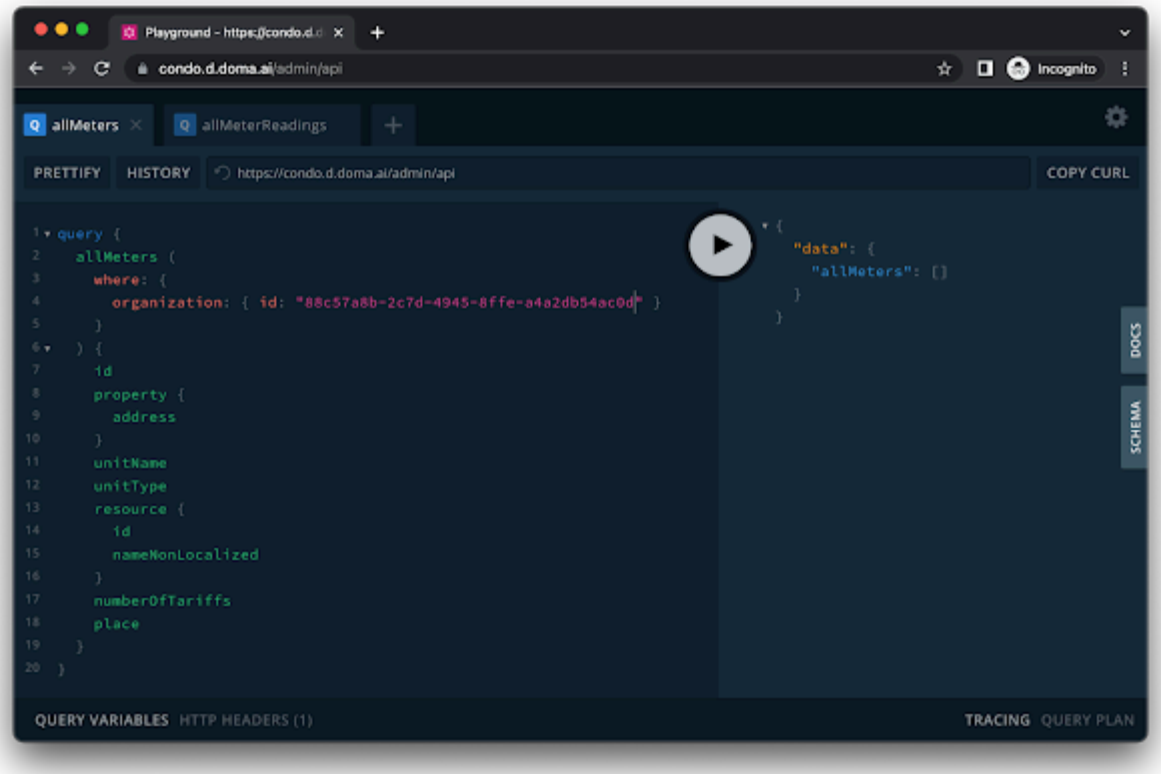
```
{
  "data": {
    "updateContact": {
      "id": "f30f495c-1adc-4b8d-89d5-6496ffd85cb1",
      "organization": {
        "id": "0400c37d-6e33-466a-a199-4be9cb35c5a0",
        "name": "DOMA API Example LLC"
      },
      "property": {
        "id": "846182fc-828d-4e49-81f2-92f880528cd4",
        "address": "Костромская обл, Галичский р-н, село Нагатино, ул Полевая, д 3"
      },
      "unitType": "flat",
      "unitName": "1",
      "name": "Семен",
      "phone": "+79068088888",
      "email": null,
      "deletedAt": "2022-07-27T12:16:40.395Z"
    }
  }
}
```

Работа с ИПУ (Meter)

Помимо заявок от жителей управляющая компания принимает и показания их приборов учета в разделе “ИПУ” (https://condo.d.doma.ai/meter).

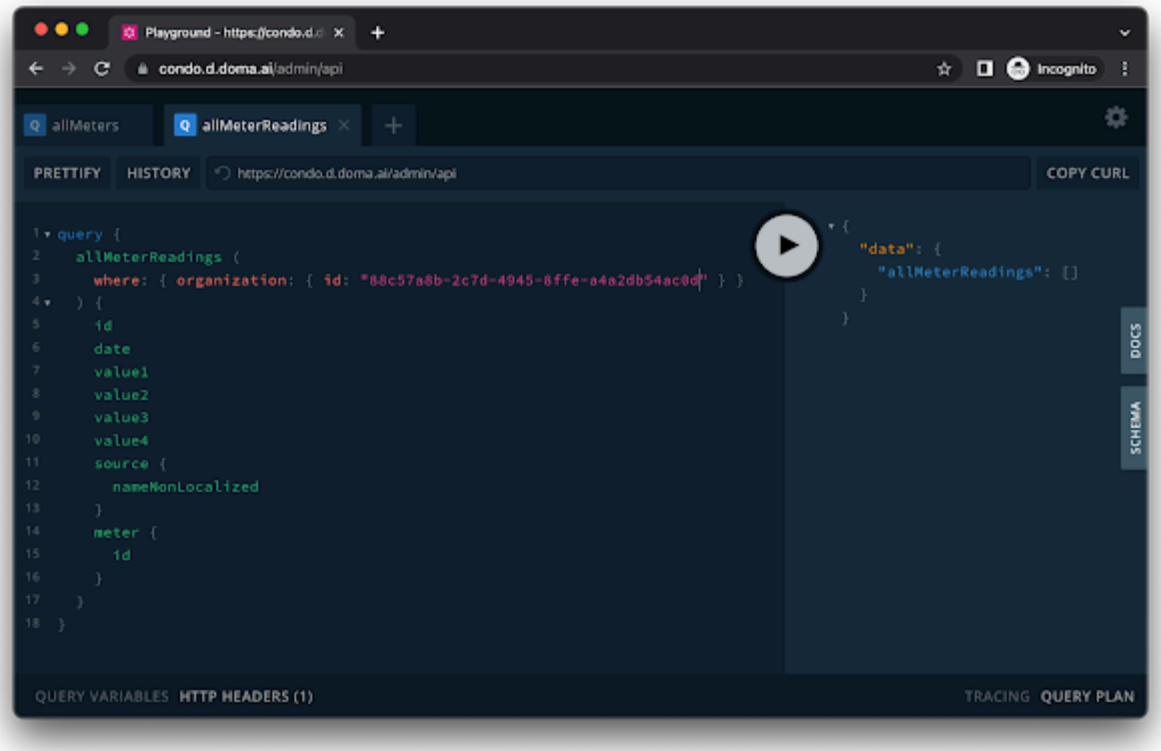
Просмотр списка приборов учета (allMeters)

В нашу недавно созданную УК еще не было передано ни одного показания. Попробуем получить список всех ИПУ и их показаний для нашей организации.



QUERY	RESULT
{ allMeters (where: { organization: { id: "88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d" } }) { id property { address } unitName unitType resource { id nameNonLocalized } numberOfTariffs place } }	{ "data": { "allMeters": [] } }

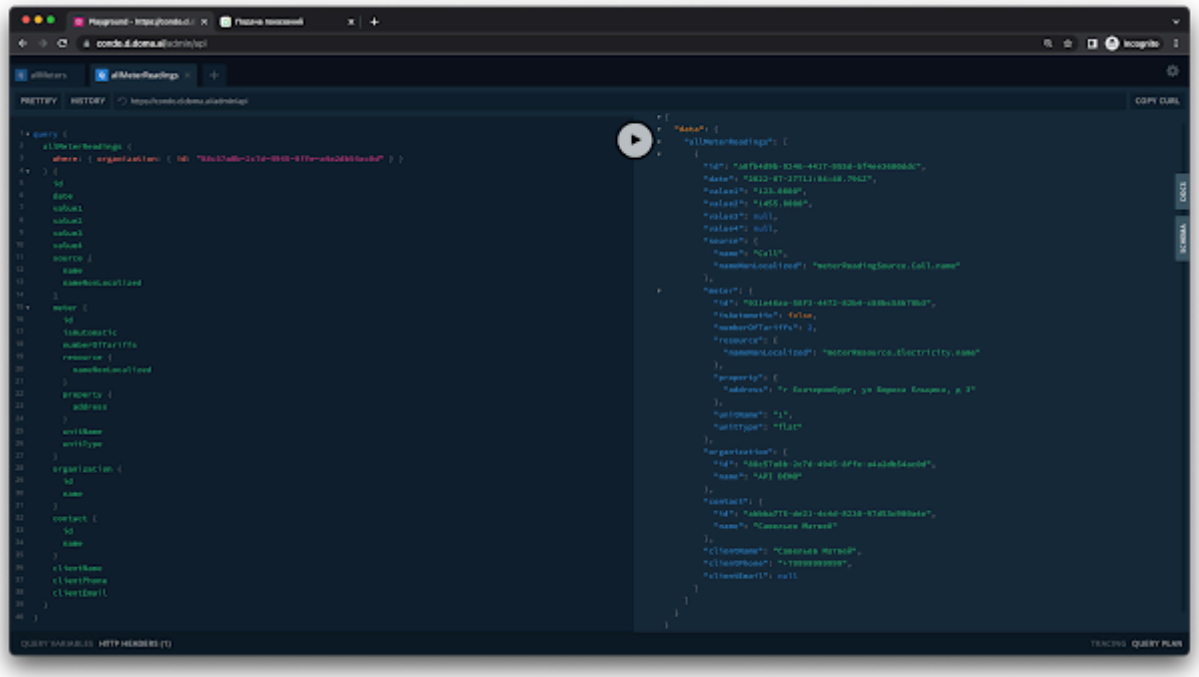
Просмотр списка переданных показаний для приборов учета (allMeterReadings)



QUERY	RESULT

{
allMeterReadings (
where: { organization: { id: "88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d" } }
) {
id
date
value1
value2
value3
value4
source {
nameNonLocalized
}
meter {
id
}
}
}
	{
	"data": {
	"allMeterReadings": []
	}
	}

Теперь создадим показания в разделе ИПУ (https://condo.d.doma.ai/meter/create) и посмотрим на показания снова.



QUERY	RESULT
{ allMeterReadings (where: { organization: { id: "88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d" } }) { id date value1 value2 value3 value4 source { name nameNonLocalized } meter { id isAutomatic numberOfTariffs resource { nameNonLocalized } property { address } unitName unitType } organization {	{ "data": { "allMeterReadings": [{ "id": "a0fb4d9b-9246-4417-955d-bf4ee3680ddc", "date": "2022-07-27T11:04:40.791Z", "value1": "123.0000", "value2": "1455.0000", "value3": null, "value4": null, "source": { "name": "Call", "nameNonLocalized": "meterReadingSource.Call.name" }, "meter": { "id": "931e46aa-58f3-4472-82b4-c08bc58b78b3", "isAutomatic": false, "numberOfTariffs": 2, "resource": { "nameNonLocalized": "meterResource.Electricity.name" }, "property": { "address": "г Екатеринбург, ул Бориса Ельцина, д 3" },

```
      id
      name
    }
    contact {
      id
      name
    }
    clientName
    clientPhone
    clientEmail
  }
}
```

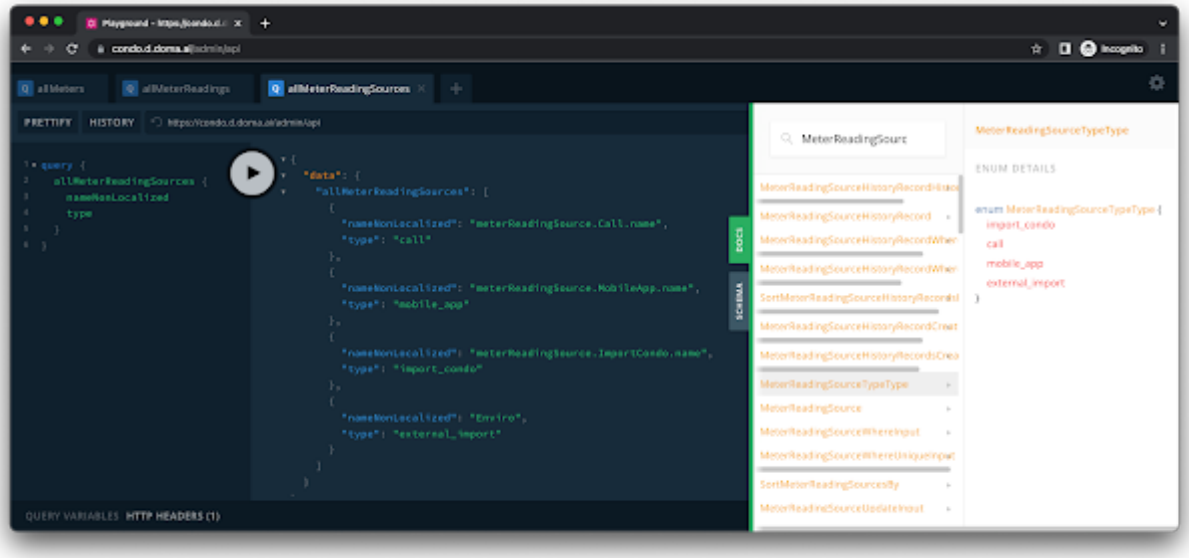
```
        "unitName": "1",
        "unitType": "flat"
      },
      "organization": {
        "id":
"88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d",
        "name": "API DEMO"
      },
      "contact": {
        "id":
"abbba776-de21-4c4d-8230-97d53c900a4e",
        "name": "Савельев Матвей"
      },
      "clientName": "Савельев Матвей",
      "clientPhone": "+79999999999",
      "clientEmail": null
    }
  ]
 }
}
```

Основные поля показаний приборов учета (MeterReading)

Поле	Значение
meter	ИПУ, к которому относятся показания
date	Время и дата сдачи показания в формате ISO (в зоне UTC)
value1, value2, value3, value4	Значение показания ИПУ. Число с плавающей точкой в строковом представлении (точка в качестве разделителя). Используется во всех типах ИПУ. Бывают три разновидности приборов учета: • однотарифный (T1) — ведёт учёт по единой тарификации, независимо от времени суток • двухтарифный — сутки делятся на два периода: дневной (T1) и ночной (T2). Во второй фазе электроэнергия стоит дешевле; • трёхтарифный — помимо T1 и T2, дополнительно включает ещё и T3 — пиковые часы. В это время

	действует тариф, предполагающий самую высокую цену. value1, value2, value3 – значения для T1, T2, T3 соответственно
source	Источник показаний Связь с MeterReadingSource

У показаний есть источник (**MeterReadingSource**), источник могут иметь разные названия, однако бывают всего 4 типов: “звонок”, “мобильное приложение”, “внутренний импорт с помощью загрузки файла” и “внешний импорт”.

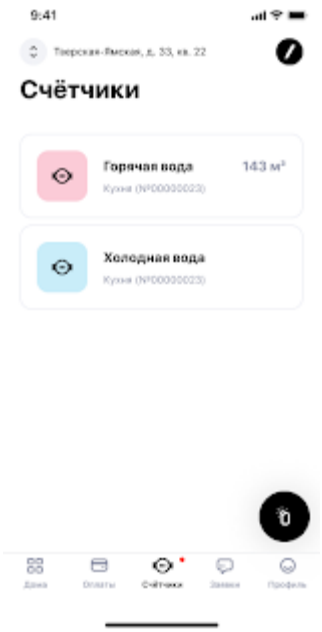


NOTE: Так же часть строковых полей имеет 2 версии: **<поле>** и **<поле>NonLocalized**. Версия NonLocalized содержит ключ для перевода значения или самое значение, а обычная версия является виртуальным полем и может возвращать различные значения в зависимости от локали. Стоит учесть этот нюанс, если вы собираетесь использовать данные поля для фильтрации.

Основные поля прибора учета (Meter)

Поле	Значение
organization	Организацию, к которой относится ИПУ Связь с моделью Organization
property	Объект общей собственности, с которым связан прибор учета (ПУ / ИПУ)

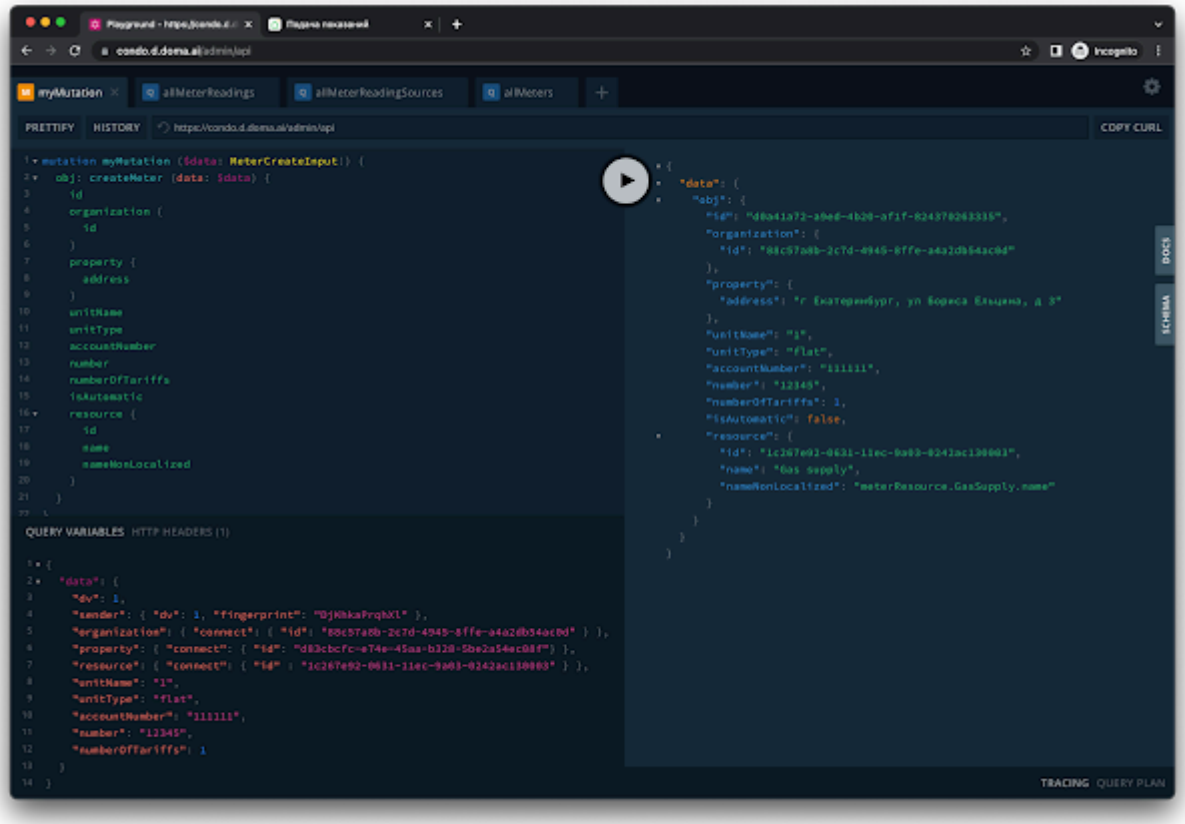
unitName	Номер помещения
unitType	Тип помещения
number	Номер счетчика, например “A03 9908”
accountNumber	Номер лицевого счета с которым связан ИПУ
place	Место установки счетчика
resource	Тип ресурса (Электроэнергия, газоснабжение и т.п.)
numberOfTariffs	Количество тарифов (однотарифный, вухтарифный, трехтарифный)
installationDate	Дата установки ПУ (формат ISO)
comissioningDate	Дата ввода в эксплуатацию (ISO)
verificationDate	Дата поверки (ISO)
nextVerificationDate	Дата следующей поверки (ISO)
controlReadingsDate	Дата контрольных показаний (ISO)
sealingDate	Дата опломбировки (ISO)
isAutomatic	Является ли счетчик автоматическим
b2bApp	Мини-апп в CRM, подключенное у organization, являющееся мастер-системой для данного ИПУ. Обязательное поле, если счетчик автоматический.
b2cApp	Мини-апп в мобильном приложении, которое будет открываться вместо стандартного интерфейса МП при тапе на счетчик в разделе “Счетчики” (см. ниже).



Этран со счетчиками в мобильном приложении жителя

Создание счетчика (createMeter)

Попробуем создать счетчик газоснабжения в одном из домов нашей организации, используя мутацию **createMeter.** Отметим, что до этого мы использовали несколько переменных, которые позже собирались в единую набор **data.** Здесь в демонстрационных целях срзау будем описывать переменную **data,** так как данный подход более удобен при работе с API вне Playground.



MUTATION	RESULT

mutation myMutation ($data: MeterCreateInput!) {
  obj: createMeter (data: $data) {
  id
  organization {
  id
  }
  property {
  address
  }
  unitName
  unitType
  accountNumber
  number
  numberOfTariffs
  isAutomatic
  resource {
  id
  name
  nameNonLocalized
  }
  }
}
	{
	  "data": {
	  "obj": {
	  "id":
	"d0a41a72-a9ed-4b20-af1f-824370263335",
	  "organization": {
	  "id":
	"88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d"
	  },
	  "property": {
	  "address": "г Екатеринбург, ул Бориса Ельцина, д 3"
	  },
	  "unitName": "1",
	  "unitType": "flat",
	  "accountNumber": "111111",
	  "number": "12345",
	  "numberOfTariffs": 1,
	  "isAutomatic": false,
	  "resource": {

<table>
<tr>
<td>

```
{
  "data": {
    "dv": 1,
    "sender": { "dv": 1, "fingerprint": "DjKhkaPrqhXl" },
    "organization": { "connect": { "id": "88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d"
} },
    "property": { "connect": { "id": "d83cbcfc-e74e-45aa-b328-5be2a54ec08f"} },
    "resource": { "connect": { "id" : "1c267e92-0631-11ec-9a03-0242ac130003" }
},
    "unitName": "1",
    "unitType": "flat",
    "accountNumber": "111111",
    "number": "12345",
    "numberOfTariffs": 1
  }
}
```

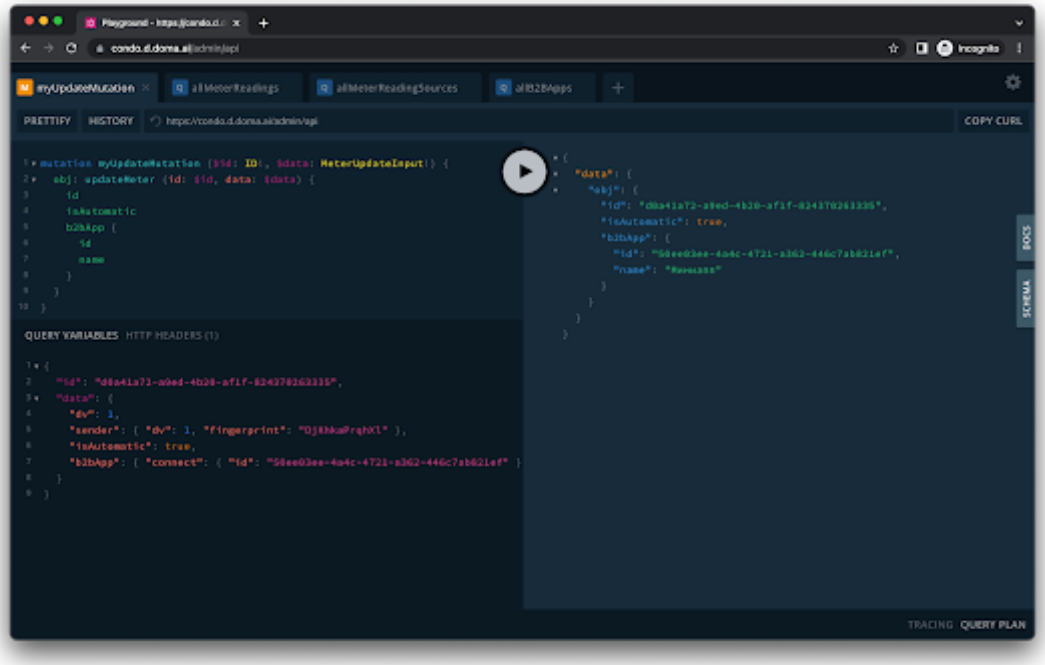
</td>
<td>

```
      "id":
"1c267e92-0631-11ec-9a03-0242ac130
003",
      "name": "Gas supply",
      "nameNonLocalized":
"meterResource.GasSupply.name"
    }
   }
  }
}
```

</td>
</tr>
</table>

Обновление счетчика (updateMeter)

Теперь попробуем обновить существующий счётчик и сделаем его автоматическим, привязав к мастер-системе, находящейся в мини-приложении “Миниапп”



MUTATION	RESULT

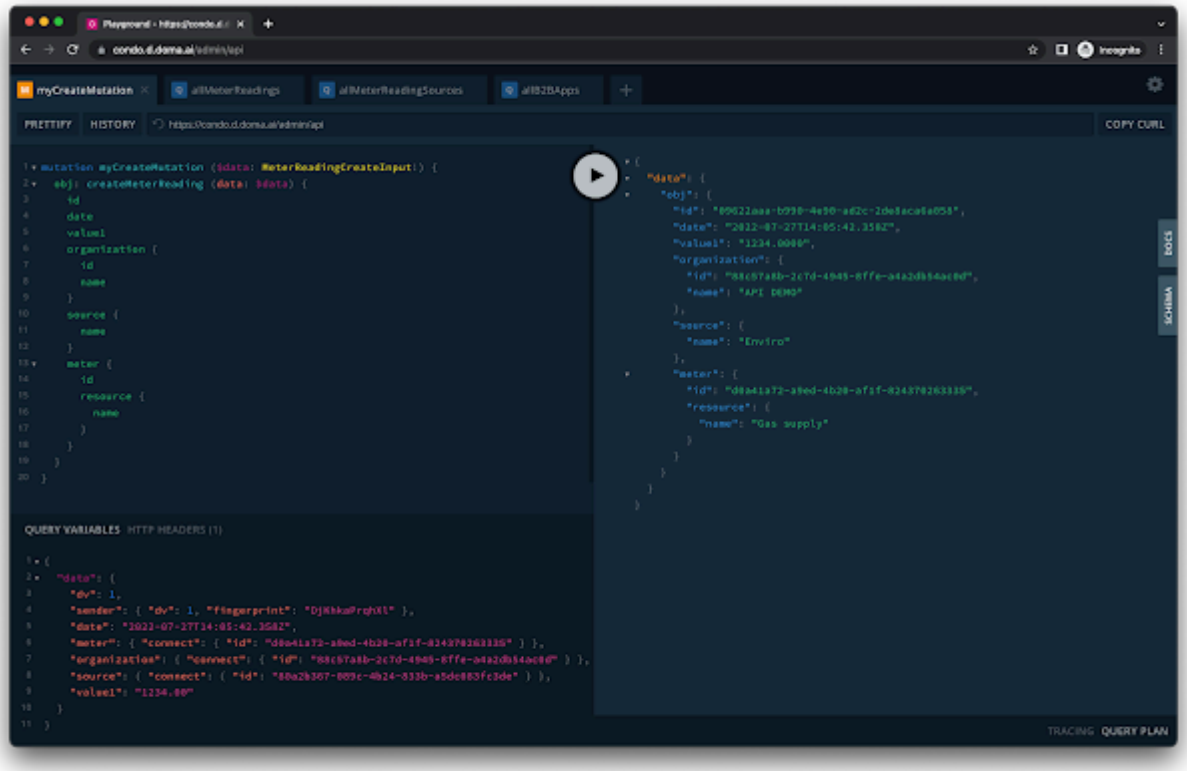
```
mutation myUpdateMutation ($id: ID!, $data: MeterUpdateInput!) {
  obj: updateMeter (id: $id, data: $data) {
    id
    isAutomatic
    b2bApp {
      id
      name
    }
  }
}
```

```
{
  "id": "d0a41a72-a9ed-4b20-af1f-824370263335",
  "data": {
    "dv": 1,
    "sender": { "dv": 1, "fingerprint": "DjKhkaPrqhXl" },
    "isAutomatic": true,
    "b2bApp": { "connect": { "id": "50ee03ee-4a4c-4721-a362" } }
  }
}
```

```
{
  "data": {
    "obj": {
      "id": "d0a41a72-a9ed-4b20-af1f",
      "isAutomatic": true,
      "b2bApp": {
        "id": "50ee03ee-4a4c-4721-a362",
        "name": "Миниапп"
      }
    }
  }
}
```

Передача показаний прибора учета (createMeterReading)

Ну и напоследок давайте передадим показания в наш ПУ, источник которых - внешний импорт из системы Enviro (добавление и редактирование источников осуществляется техподдержкой).



MUTATION

```
mutation myCreateMutation ($data: MeterReadingCreateInput!) {
  obj: createMeterReading (data: $data) {
    id
    date
    value1
    organization {
      id
      name
    }
    source {
      name
    }
    meter {
      id
      resource {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "dv": 1,
    "sender": { "dv": 1, "fingerprint": "DjKhkaPrqhXl" },
              "date": "2022-07-27T14:05:42.358Z",
    "meter": { "connect": { "id": "d0a41a72-a9ed-4b20-af1f-824370263335" } },
    "organization": { "connect": { "id":
"88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d" } },
    "source": { "connect": { "id": "80a2b367-089c-4b24-833b-a5dc083fc3de" } },
              "value1": "1234.00"
  }
}
```

RESULT

```
{
  "data": {
    "obj": {
      "id":
"09622aaa-b990-4e90-ad2c-2de8aca6a05
8",
      "date": "2022-07-27T14:05:42.358Z",
      "value1": "1234.0000",
      "organization": {
        "id":
"88c57a8b-2c7d-4945-8ffe-a4a2db54ac0d
",
        "name": "API DEMO"
      },
      "source": {
        "name": "Enviro"
      },
      "meter": {
        "id":
"d0a41a72-a9ed-4b20-af1f-82437026333
5",
        "resource": {
          "name": "Gas supply"
        }
      }
    }
  }
}
```

Работа с новостями

Иногда, управляющей компании нужно уведомить своих жителей о каких-то событиях, это могут быть плановые улучшения, мелкие ремонты, обновления или аварии. Это может касаться конкретных квартир, лицевых счетов, или нескольких домов.

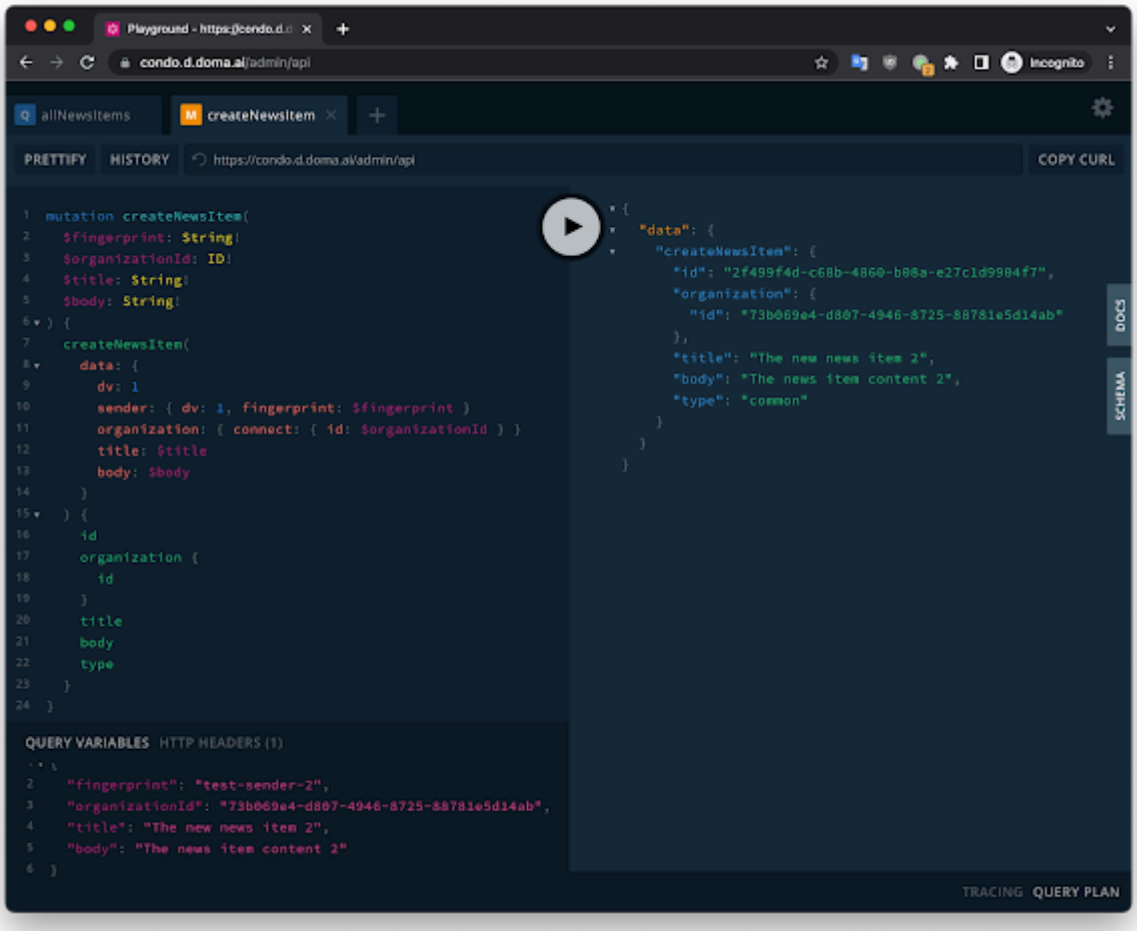
Новость – некоторое важное событие, созданное управляющей компанией, для уведомления своих жителей.

Новость может быть двух типов (type): “обычная” (common) и “аварийная” (emergency). В случае публикации аварийной новости, жители незамедлительно получают push-уведомление. Если не указывать тип, то автоматически будет назначен тип common.

У новости есть дата публикации (sendAt). До наступления этой даты жители не увидят новость в мобильном приложении и не получат пуш уведомление. По факту отправки в модели такой отложенной новости будет заполнено поле sentAt - это дата отправки уведомления жителям. Есть также дата актуальности (validBefore): новость будет отображаться на главной странице мобильного приложения до наступления этой даты, а после - в списке новостей в отдельном разделе приложения.

Чтобы указать жителей, которые увидят новость в приложении и получат пуш-уведомление, используется модель NewsItemScope. Без областей видимости пользователи не увидят новость. Для отображения новости вообще всем жителям организации достаточно указать пустой (то есть без фильтров) newsItemScope. Если же нужно отправить таргетированную новость, то scope, привязанный к этой новости, может содержать property, unitType и/или unitName. Последние три поля аналогичны полям в модели Resident.

Создание новости (createNewsItem)



QUERY

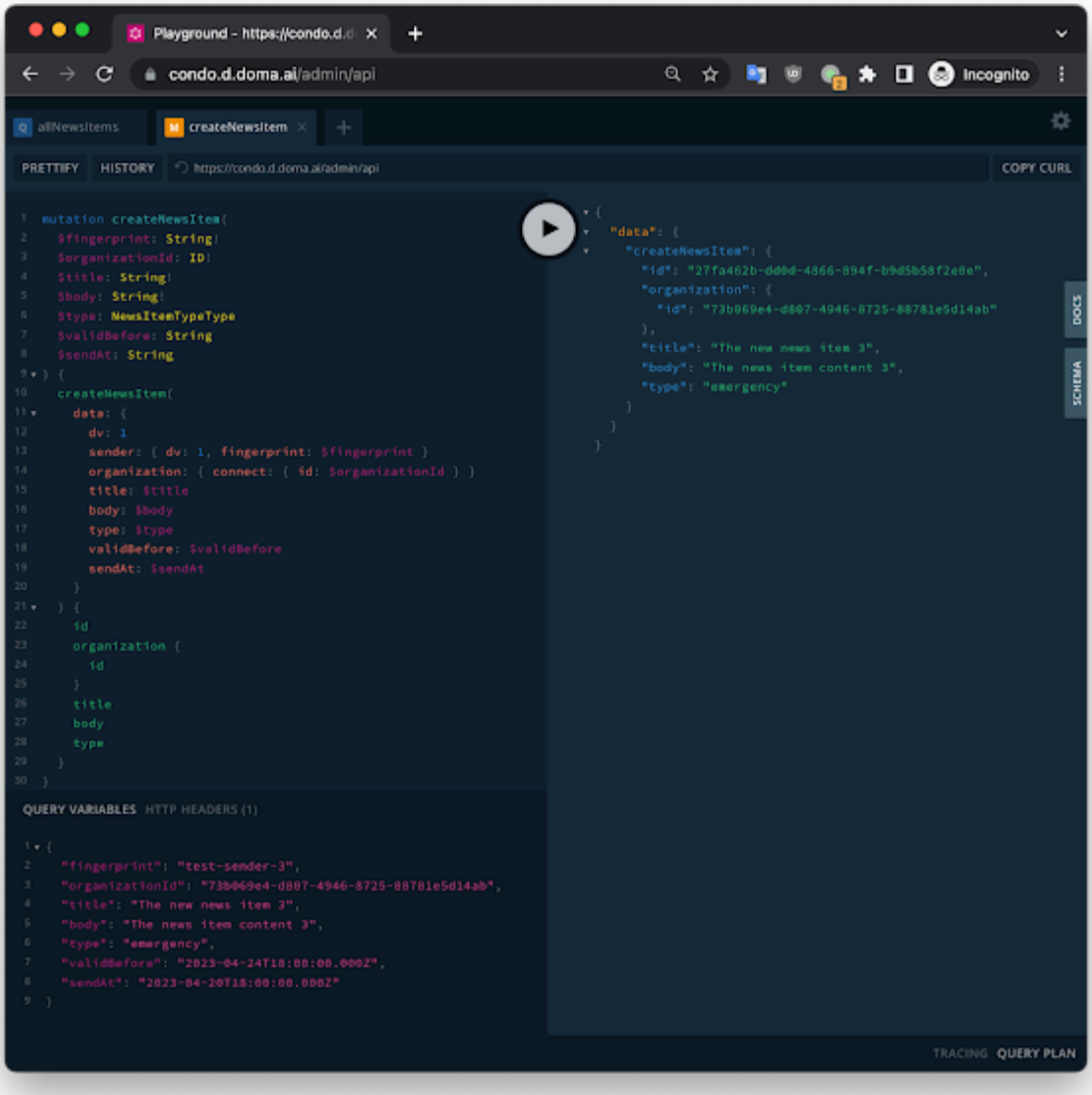
```
mutation createNewsItem(
  $fingerprint: String!
  $organizationId: ID!
  $title: String!
  $body: String!
) {
  createNewsItem(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      organization: { connect: { id: $organizationId } }
      title: $title
      body: $body
    }
  ) {
    id
    organization {
      id
    }
    title
    body
    type
  }
}
```

```
{
  "fingerprint": "test-sender02",
  "organizationId":
"73b069e4-d807-4946-8725-88781e5d14ab",
  "title": "The new news item 2",
  "body": "The news item content 2"
}
```

RESULT


```
{
  "data": {
    "createNewsItem": {
      "id": "9e4c1776-90cb-4bbc-a2c0-17fec16f3a72",
      "organization": {
        "id": "73b069e4-d807-4946-8725-88781e5d14ab"
      },
      "title": "The new news item",
      "body": "The news item content",
      "type": "common"
    }
  }
}
```


Можно указать дополнительные поля, такие как тип заявки, дата актуальности и дата отправки



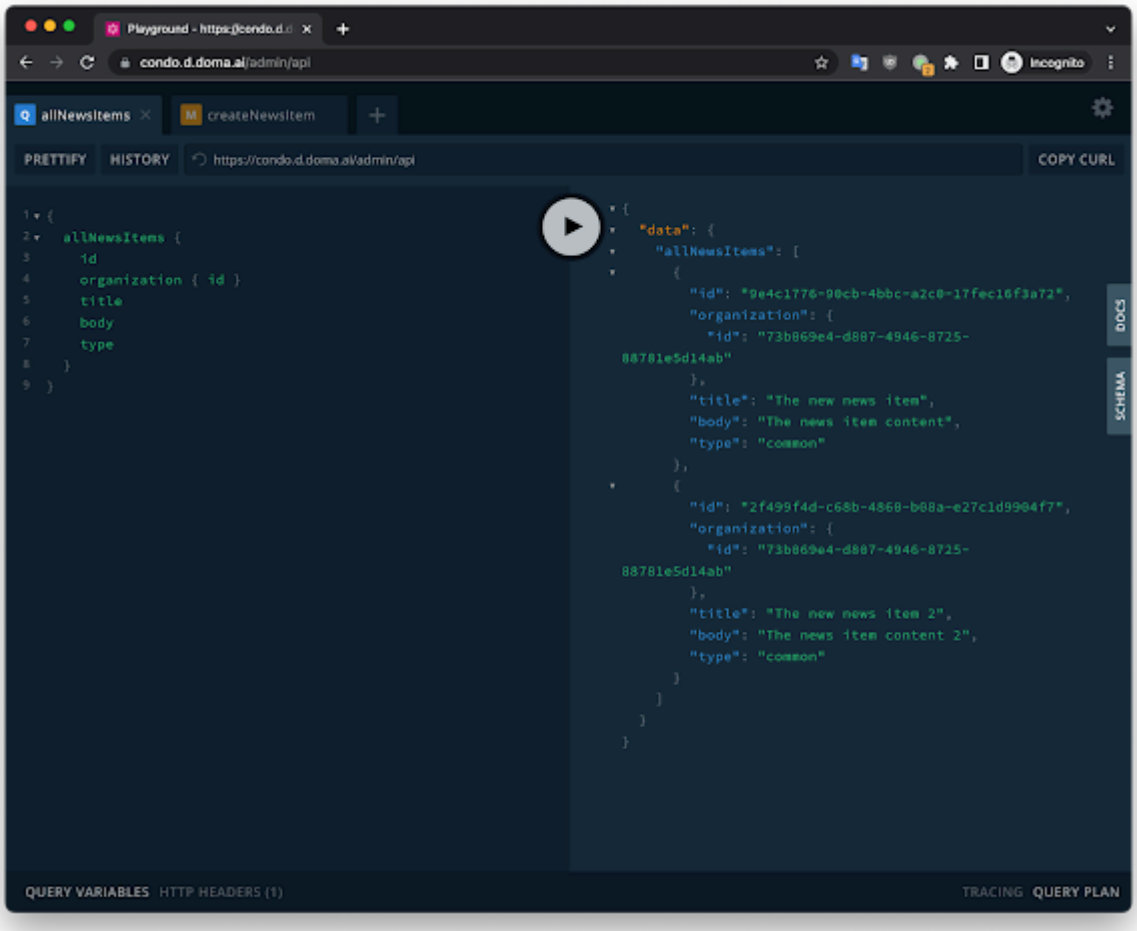
QUERY	RESULT

```
mutation createNewsItem(
  $fingerprint: String!
  $organizationId: ID!
  $title: String!
  $body: String!
  $type: NewsItemTypeType
  $validBefore: String
  $sendAt: String
) {
  createNewsItem(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      organization: { connect: { id: $organizationId } }
      title: $title
      body: $body
      type: $type
      validBefore: $validBefore
      sendAt: $sendAt
    }
  ) {
    id
    organization {
      id
    }
    title
    body
    type
  }
}
```

```
{
  "fingerprint": "test-sender-3",
  "organizationId":
"73b069e4-d807-4946-8725-88781e5d14ab",
  "title": "The new news item 3",
  "body": "The news item content 3",
  "type": "emergency",
  "validBefore": "2023-04-24T18:00:00.000Z",
  "sendAt": "2023-04-20T18:00:00.000Z"
}
```

```
{
  "data": {
    "createNewsItem": {
      "id": "27fa462b-dd0d-4866-894f-b9d5b58f2e0e",
      "organization": {
        "id": "73b069e4-d807-4946-8725-88781e5d14ab"
      },
      "title": "The new news item 3",
      "body": "The news item content 3",
      "type": "emergency"
    }
  }
}
```

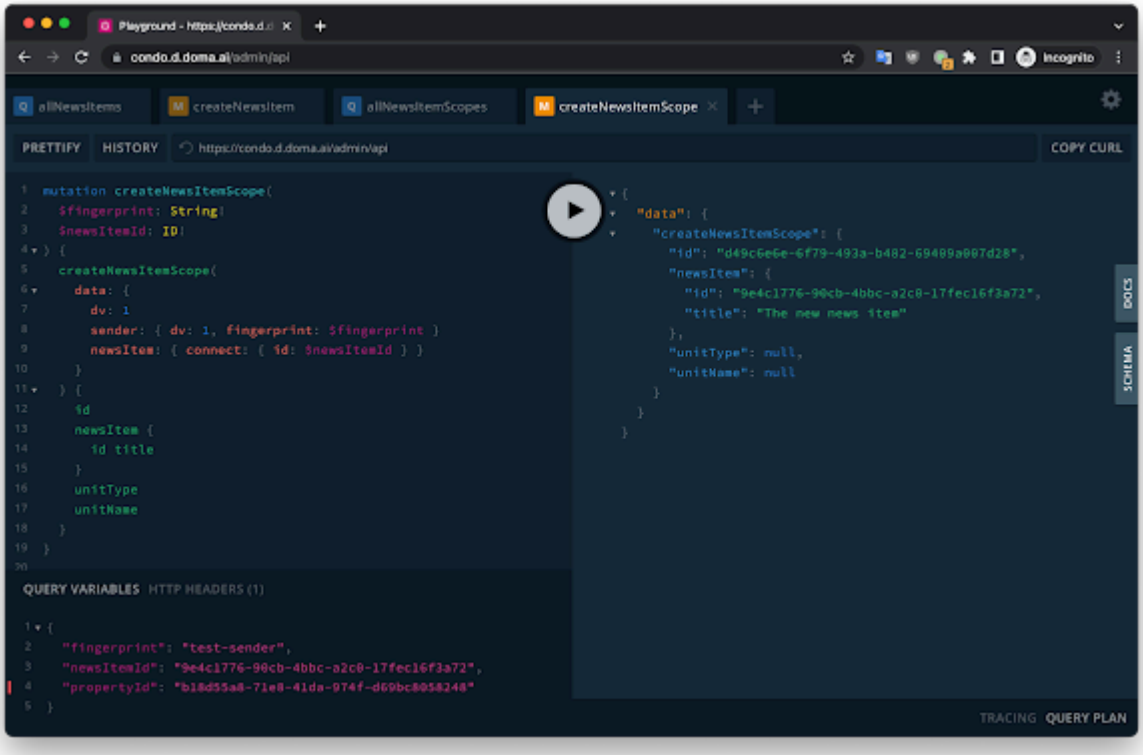
Получение списка новостей (allNewsItems)



QUERY	RESULT
{ allNewsItems { id organization { id } title body type } }	{ "data": { "allNewsItems": [{ "id": "9e4c1776-90cb-4bbc-a2c0-17fec16f3a72", "organization": { "id": "73b069e4-d807-4946-8725-88781e5d14ab" }, "title": "The new news item", "body": "The news item content", "type": "common" }] } }

Создание области видимости новости (createNewsItemScope)

Пример создания области видимости новости для всех жителей в организации



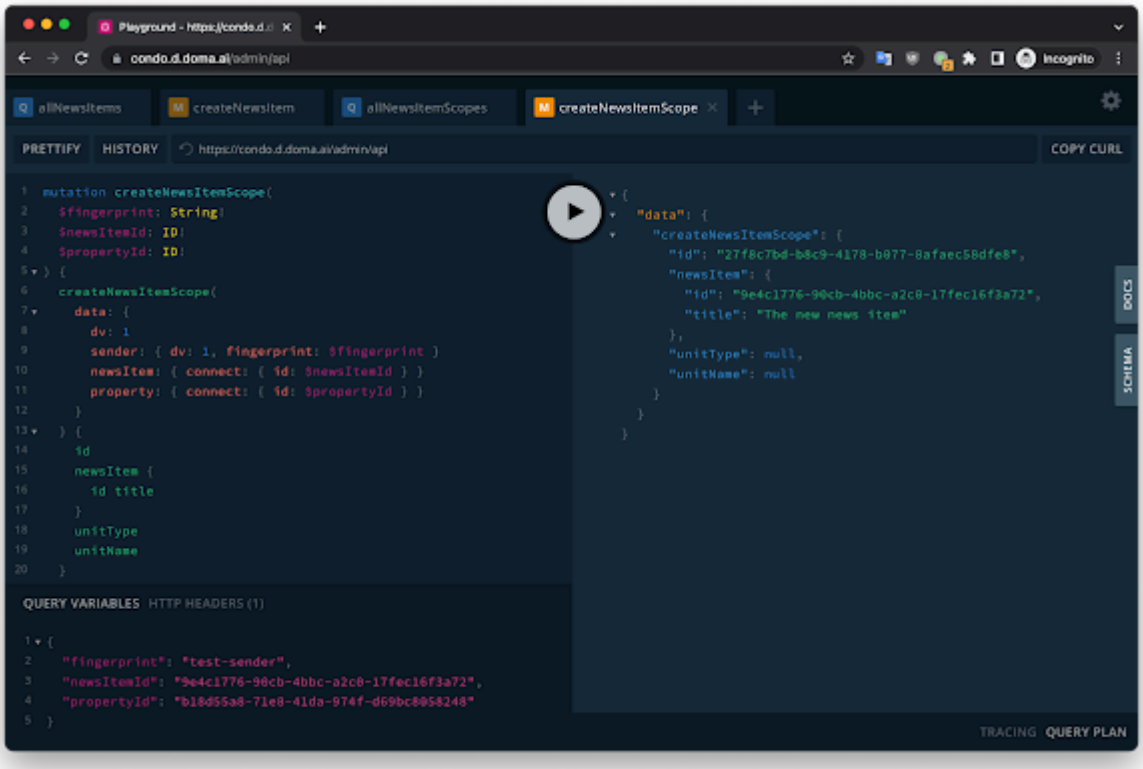
QUERY	RESULT

```
mutation createNewsItemScope(
  $fingerprint: String!
  $newsItemId: ID!
) {
  createNewsItemScope(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      newsItem: { connect: { id: $newsItemId } }
    }
  ) {
    id
    newsItem {
      id title
    }
    unitType
    unitName
  }
}
```

```
{
  "fingerprint": "test-sender",
  "newsItemId":
"9e4c1776-90cb-4bbc-a2c0-17fec16f3a72"
}
```

```
{
  "data": {
    "createNewsItemScope": {
      "id": "d49c6e6e-6f79-493a-b482-69409a007d28",
      "newsItem": {
        "id": "9e4c1776-90cb-4bbc-a2c0-17fec16f3a72",
        "title": "The new news item"
      },
      "unitType": null,
      "unitName": null
    }
  }
}
```

Если нужно отправить новость, предназначенную для какого-то дома, то нужно указать propertyId

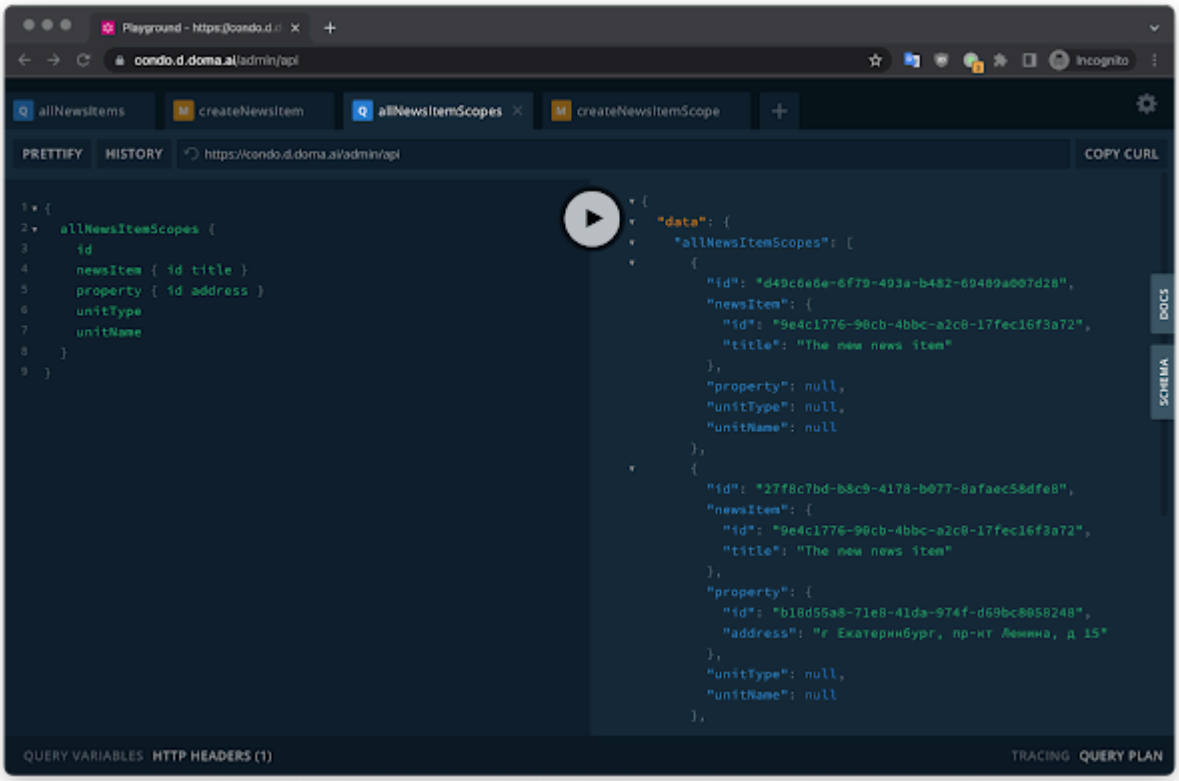


QUERY	RESULT

mutation createNewsItemScope(
$fingerprint: String!
$newsItemId: ID!
) {
createNewsItemScope(
data: {
dv: 1
sender: { dv: 1, fingerprint: $fingerprint }
newsItem: { connect: { id: $newsItemId } }
}
) {
id
newsItem {
id title
}
unitType
unitName
}
}
	{
	"data": {
	"createNewsItemScope": {
	"id": "d49c6e6e-6f79-493a-b482-69409a007d28",
	"newsItem": {
	"id": "9e4c1776-90cb-4bbc-a2c0-17fec16f3a72",
	"title": "The new news item"
	},
	"unitType": null,
	"unitName": null
	}
	}
	}

{ "fingerprint": "test-sender", "newsItemId": "9e4c1776-90cb-4bbc-a2c0-17fec16f3a72", "propertyId": "b18d55a8-71e8-41da-974f-d69bc8058248" }	

Просмотр области видимости новости (allNewsItemScopes)



Редактирование новости (updateNewsItem)

Правила редактирования в процессе проектирования.

Прочтение новости пользователем (createNewsItemUserRead)

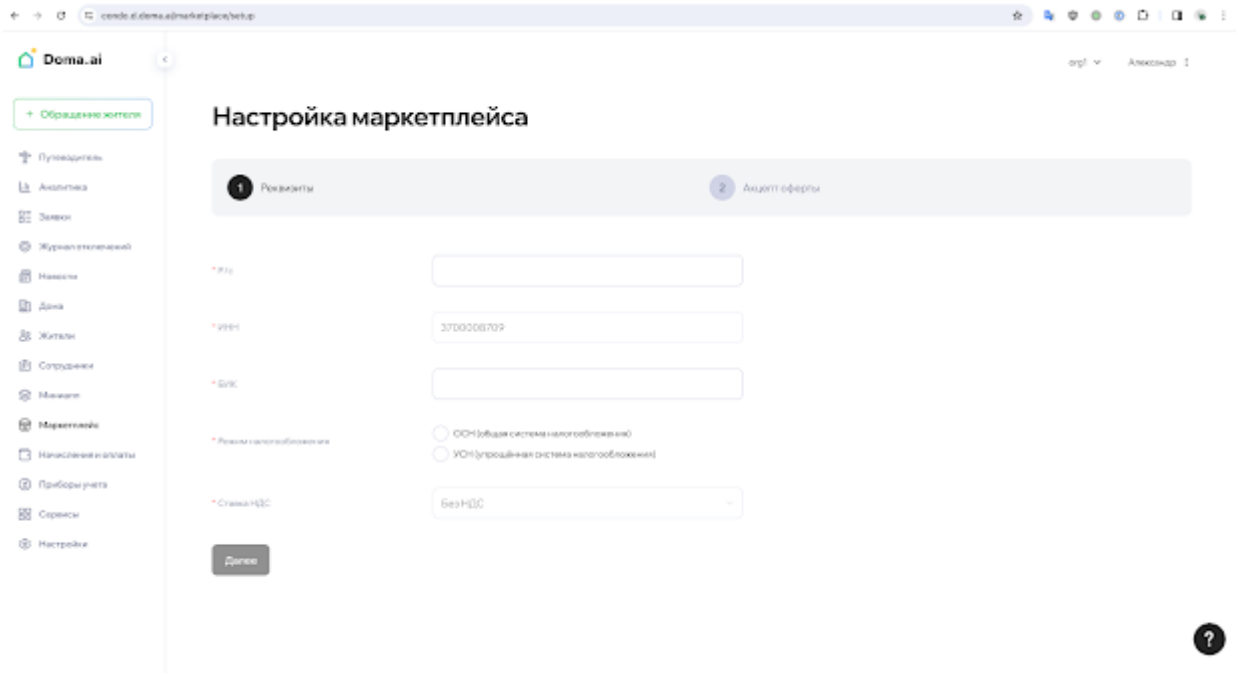
Работа со счетами (invoice)

У управляющих компаний есть потребность продавать своим жителям услуги и товары. Например, сотрудники УК могут оказывать услуги по ремонту коммуникаций и сетей, замене и подключению оборудования. Чтобы жители могли видеть список услуг, у нас есть раздел Маркетплейс (https://condo.d.doma.ai/marketplace). Тут сгруппирован функционал управления витриной и счетами.

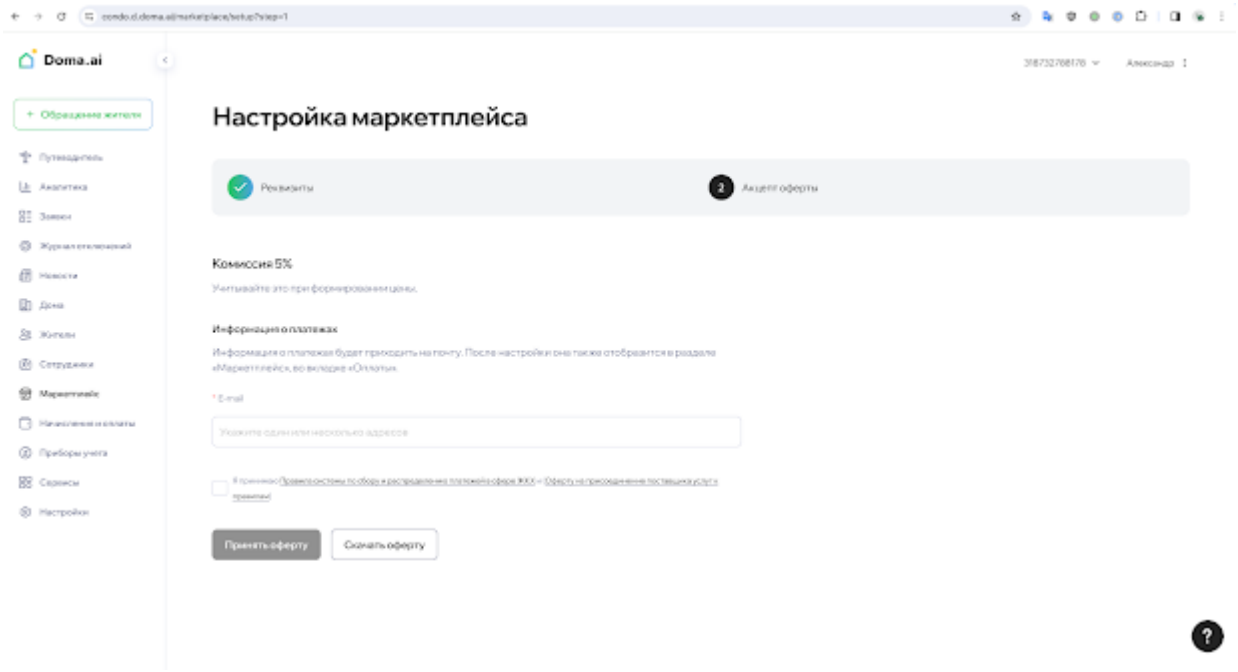
Подготовка

Чтобы пользоваться функционалом маркетплейса необходимо выполнить первоначальную настройку (подключить маркетплейс).

Нужно указать реквизиты для получения оплаты за услуги, выбрать режим налогообложения и ставку НДС. Это на первом шаге.



На втором шаге нужно указать список почтовых адресов для получения детализации платежей и принять оферту.



Принятая оферта будет отправлена на указанную почту.

Создание счета

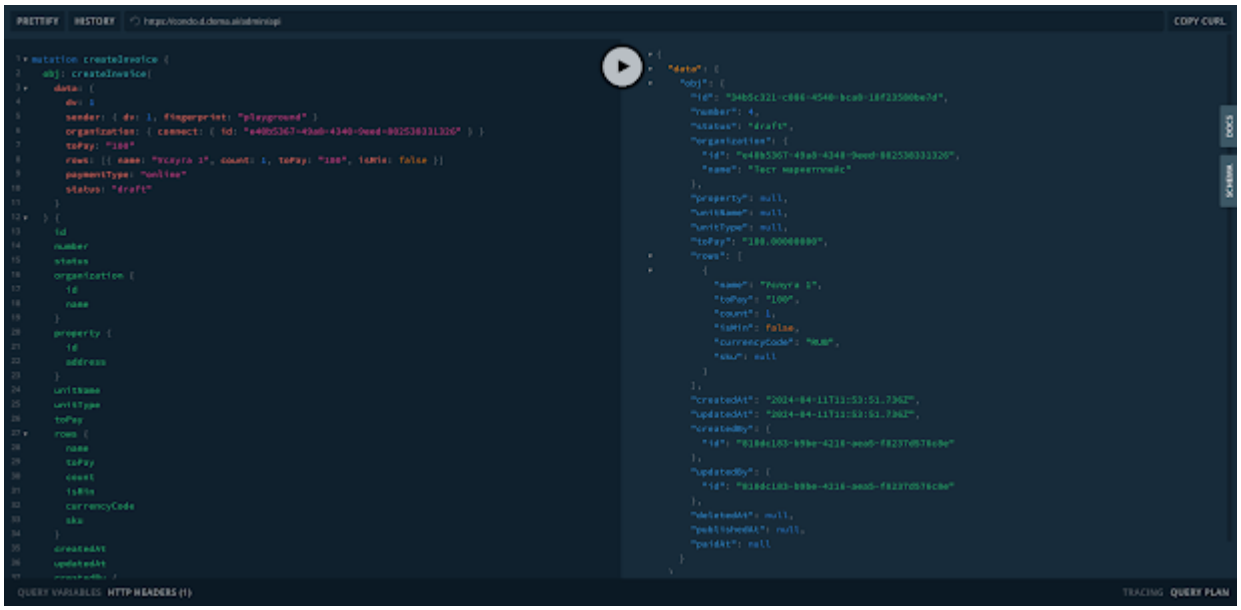
Счет можно создать с указанием данных о плательщике (адрес, телефон, имя) или без. В первом случае житель увидит счет в мобильном приложении и оттуда же сможет оплатить. Во втором - счет можно будет оплатить только по ссылке.

Анонимный счет

Минимальный набор полей, который в принципе необходим для создания счета, создает анонимный счет. Такой счет не виден в мобильном приложении и может быть оплачен только по ссылке.

- dv и sender
- organization
  { id: UUID } организация, которая создает счет
- toPay
  Общая сумма счета
- rows
  Позиции, включенные в счет (услуги/товары). Это массив объектов вида:
  { name: "Услуга 1", count: 1, toPay: "100", isMin: false }
  Поле isMin означает, что toPay - это минимальная цена (цена от). Ставится в false после окончательного согласования цены.
  Важно: счет, в котором есть не конкретные цены, невозможно опубликовать.
- paymentType
  Тип оплаты. Возможные варианты: online и cash. Это поле носит скорее информационный характер. Всегда можно сформировать ссылку на оплату и оплатить наличными.
  Важно: автоматическая смена статусов будет работать только для онлайн оплаты. При наличной оплате статусы переключаются вручную или через АПИ.
- status
  Это статус счета. Возможные значения: draft, published, paid, canceled. Черновик

невозможно оплатить и нельзя увидеть в мобильном приложении. Опубликованный счет нельзя редактировать, как и оплаченный.



QUERY	RESULT

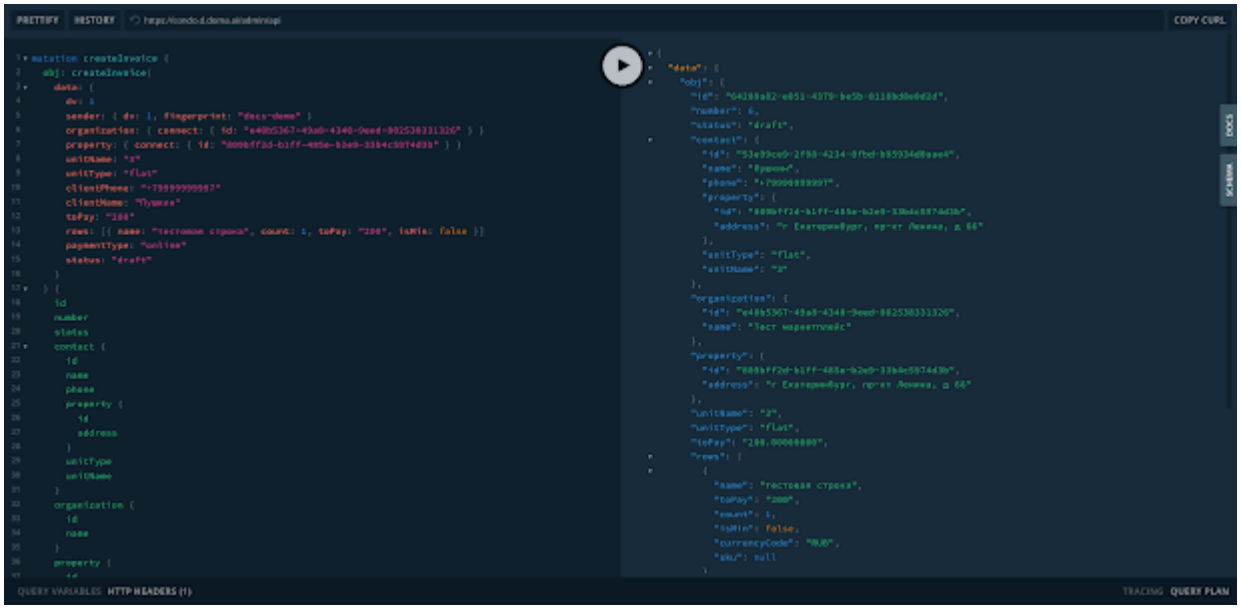
mutation createInvoice {
  obj: createInvoice(
  data: {
  dv: 1
  sender: { dv: 1, fingerprint: "playground" }
  organization: { connect: { id: "e40b5367-49a8-4340-9eed-802538331326" } }
  toPay: "100"
  rows: [{ name: "Услуга 1", count: 1, toPay: "100", isMin: false }]
  paymentType: "online"
  status: "draft"
  }
) {
  id
  number
  status
  organization {
  id
  name
  }
  property {
  id
  address
  }
  unitName
  unitType
  toPay
  rows {
  name
	{
	  "data": {
	  "obj": {
	  "id": "34b5c321-c086-4540-bca8-18f23580be7d",
	  "number": 4,
	  "status": "draft",
	  "organization": {
	  "id": "e40b5367-49a8-4340-9eed-802538331326",
	  "name": "Тест маркетплейс"
	  },
	  "property": null,
	  "unitName": null,
	  "unitType": null,
	  "toPay": "100.00000000",
	  "rows": [
	  {
	  "name": "Услуга 1",
	  "toPay": "100",
	  "count": 1,
	  "isMin": false,
	  "currencyCode": "RUB",
	  "sku": null
	  }
	  ],
	  "createdAt": "2024-04-11T11:53:51.736Z",
	  "updatedAt": "2024-04-11T11:53:51.736Z",
	  "createdBy": {
	  "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
	  },
	  "updatedBy": {

```
    toPay
    count
    isMin
    currencyCode
    sku
   }
   createdAt
   updatedAt
   createdBy {
    id
   }
   updatedBy {
    id
   }
   deletedAt
   publishedAt
   paidAt
  }
}
```

```
      "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
    },
    "deletedAt": null,
    "publishedAt": null,
    "paidAt": null
   }
  }
}
```

Счет для жителя

Чтобы счет был виден в мобильном приложении, нужно указать адрес плательщика. Это поля property, unitType и unitName. При наличии, можно указать имя и телефон жителя в полях clientName и clientPhone. Тогда к счету будет привязан контакт жителя (поле contact в ответе сервера). Если такой контакт уже добавлен, то привяжется он. Если такого контакта еще нет, то он будет создан и привязан к счету.



QUERY	RESULT

```
mutation createInvoice {
  obj: createInvoice(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: "docs-demo" }
      organization: { connect: { id:
"e40b5367-49a8-4340-9eed-802538331326" } }
      property: { connect: { id:
"809bff2d-b1ff-485e-b2e9-33b4c5974d3b" } }
      unitName: "3"
      unitType: "flat"
      clientPhone: "+79999999997"
      clientName: "Пушкин"
      toPay: "200"
      rows: [{ name: "тестовая строка", count: 1, toPay:
"200", isMin: false }]
      paymentType: "online"
      status: "draft"
    }
  ) {
    id
    number
    status
    contact {
      id
      name
      phone
      property {
        id
        address
      }
      unitType
      unitName
    }
    organization {
      id
      name
    }
    property {
      id
      address
    }
    unitName
    unitType
    toPay
    rows {
      name
      toPay
      count
      isMin
      currencyCode
      sku
    }
    createdAt
    updatedAt
    createdBy {
      id
    }
```


```
{
  "data": {
    "obj": {
      "id": "64289a82-e051-4379-be5b-8118bd8e0d2d",
      "number": 6,
      "status": "draft",
      "contact": {
        "id": "53e99ce9-2f98-4234-8fbd-b95934d0aae4",
        "name": "Пушкин",
        "phone": "+79999999997",
        "property": {
          "id": "809bff2d-b1ff-485e-b2e9-33b4c5974d3b",
          "address": "г Екатеринбург, пр-кт Ленина, д 66"
        },
        "unitType": "flat",
        "unitName": "3"
      },
      "organization": {
        "id": "e40b5367-49a8-4340-9eed-802538331326",
        "name": "Тест маркетплейс"
      },
      "property": {
        "id": "809bff2d-b1ff-485e-b2e9-33b4c5974d3b",
        "address": "г Екатеринбург, пр-кт Ленина, д 66"
      },
      "unitName": "3",
      "unitType": "flat",
      "toPay": "200.00000000",
      "rows": [
        {
          "name": "тестовая строка",
          "toPay": "200",
          "count": 1,
          "isMin": false,
          "currencyCode": "RUB",
          "sku": null
        }
      ],
      "createdAt": "2024-04-12T04:17:58.297Z",
      "updatedAt": "2024-04-12T04:17:58.297Z",
      "createdBy": {
        "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
      },
      "updatedBy": {
        "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
      },
      "deletedAt": null,
      "publishedAt": null,
      "paidAt": null
    }
  }
}
```

```
    updatedBy {
      id
    }
    deletedAt
    publishedAt
    paidAt
  }
}
```

b2c

Счет может быть выставлен жителю с помощью мини приложения. Запрос на создание счета отправляется с бэкенда мини приложения. Для этого мини приложение должно быть авторизовано под сервисным пользователем (см. раздел аутентификация).

Сервисный пользователь мини приложения будет иметь доступ к выставлению, редактированию и чтению счетов от имени организации (поле organization) после того как организация подключит мини приложение (b2b часть).

Организация (organization), указанная при создании счета, будет видеть счет в разделе “Маркетплейс”. Чтобы житель получил пуш-уведомление и увидел новый счет у себя в мобильном приложении, необходимо указать следующие поля:

- client
  Это идентификатор пользователя
- property
  Дом, в котором зарегистрировался житель
- unitType
  Тип помещение жителя
- unitName
  Наименование помещения (номер квартиры)

```
mutation createInvoice {
  obj: createInvoice(
    data: {
      dv: 1,
      sender: { dv: 1, fingerprint: "docs-demo" },
      organization: { connect: { id:
"73b069e4-d807-4946-8725-88781e5d14ac"} },
      property: { connect: { id:
"2c398c8d-da20-4b4f-b8c8-a19df803e21d" } },
      client: { connect: { id:
"cf609bd7-5bb8-498d-865d-943f96bf11fc" } },
      unitType: flat,
      unitName: "13",
      rows: [{name: "тестовая строка 3", count: 1, toPay:
"203", isMin: false}],
      paymentType: online,
      status: published
    }
  ) {
    id
```

```
{
  "data": {
    "obj": {
      "id": "1b3a4250-7e90-48c0-8266-e757edf599e9",
      "number": 30,
      "status": "published",
      "organization": {
        "id": "73b069e4-d807-4946-8725-88781e5d14ac",
        "name": "org1"
      },
      "toPay": "203.00000000",
      "rows": [
        {
          "name": "тестовая строка 3",
          "toPay": "203",
          "count": 1,
          "isMin": false,
          "currencyCode": "RUB",
          "sku": null
        }
```

```
    number
    status
    organization {
      id
      name
    }
    toPay
    rows {
      name
      toPay
      count
      isMin
      currencyCode
      sku
    }
    createdAt
    updatedAt
    createdBy {
      id
    }
    updatedBy {
      id
    }
    deletedAt
    publishedAt
    paidAt
  }
}
```

```
    ],
    "createdAt": "2025-03-11T04:42:00.863Z",
    "updatedAt": "2025-03-11T04:42:00.863Z",
    "createdBy": {
      "id": "810dc183-b9be-4216-aea5-f8237d576c8f"
    },
    "updatedBy": {
      "id": "810dc183-b9be-4216-aea5-f8237d576c8f"
    },
    "deletedAt": null,
    "publishedAt": "2025-03-11T04:42:00.856Z",
    "paidAt": null
  }
 }
}
```

Расщепление

Средства, поступившие в результате оплаты счета, обычно поступают на банковские реквизиты, указанные при настройке маркетплейса (если не было сделано иных корректировок). В случае, если нужно разделить полученные средства на разные счета (разные подрядчики, поставщики услуг и т.п.), используется поле **amountDistribution** при создании счета.

Поле amountDistribution - это json массив, состоящий из объектов следующего вида:

```
{
  recipient: {
    name: string,
    tin: string,
    bic: string,
    bankName: string,
    bankAccount: string,
  },
  amount: string,
```

```
  order: number,
  vor: boolean,
  isFeePayer: boolean,
  overpaymentPart: number,
}
```

recipient
Это реквизиты получателя средств.
Важно: реквизиты должны быть предварительно провалидированы через техподдержку.

amount
Сумма, которая будет перечислена.

order
Очередность распределения средств. По умолчанию равно нулю (0). Используется для тонкой настройки в случае частичной оплаты.

vor
Аббревиатура от **V**ictim **o**f **R**ounding. Указывает кто будет брать на себя результат округления. Например, если делить 100 на 3, то два получателя получат по 33.33, а тот, у кого указано vor: true, получит 33,34.

isFeePayer
Показывает кто из получателей средств будет платить комиссию домов.

overpaymentPart
Целое число. Задает долю от переплаты, которая отправится к получателю. Например, если только у одного получателя будет указано overpaymentPart: 1, то ему отправится вся переплата. Если у двоих получателей будет указано overpaymentPart: 1, то каждый из них получит по половине переплаты.

Все поля и их типы указаны в схеме данных, которые можно найти в песочнице: https://condo.d.doma.ai/admin/api.

Ограничения

Максимальное количество получателей: 5.

Реквизиты получателей должны быть предварительно добавлены в систему. Это делается через техподдержку.

Получатели должны быть уникальными. Нельзя задать два получателя в расщеплении с одинаковыми реквизитами.

Сумма расщепления должна совпадать с суммой счета. Нельзя распределить только часть средств.

В каждой группе (получатели с одинаковым order) должен быть строго один получатель с vor: true.

Получатель, у которого указано vor: true, обязательно должен быть среди плательщиков комиссии (isFeePayer: true) и получателей переплаты (overpaymentPart).

Примеры расщепления

Для упрощения вместо объекта recipient укажем строку.

Пример 1: базовый
Самое простое распределение. Нет частичной оплаты, нет переплаты. Просто оплата счета с двумя получателями.

Сумма счета: 1000

amountDistribution: [

{ recipient: “recipient1”, amount: “200”, vor: true, isFeePayer: true, overpaymentPart: 1 },

{ recipient: “recipient2”, amount: “800” }

]

Сумма платежа: 1000

Комиссия: 5%

Результат распределения:

- recipient1: 150
- recipient2: 800
- комиссия: 50

Пример 2: округление
Пример, показывающий как работает vor: true. Здесь три получателя и все они платят комиссию. Нет частичной оплаты, нет переплаты. Просто оплата счета с тремя получателями.

Сумма счета: 300

amountDistribution: [

{ recipient: “recipient1”, amount: “100”, isFeePayer: true },

{ recipient: “recipient2”, amount: “100”, vor: true, overpaymentPart: 1, isFeePayer: true },

{ recipient: “recipient3”, amount: “100”, isFeePayer: true },

]

Сумма платежа: 300

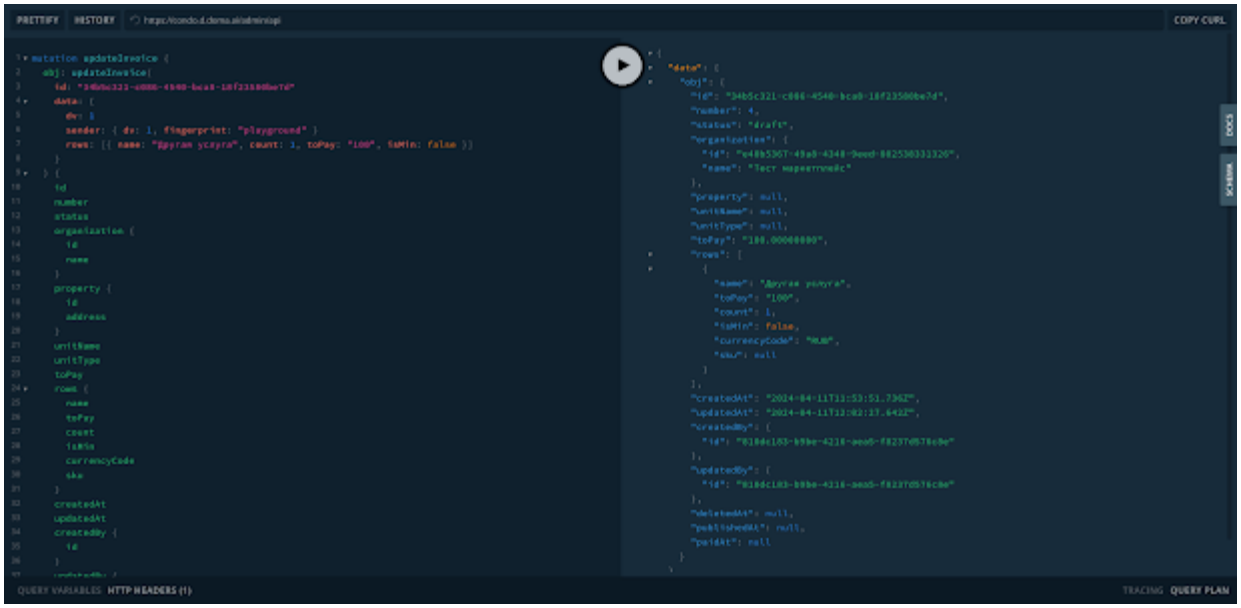
Комиссия: 10 (3,33333%)

Результат распределения:

- recipient1: 96,67
- recipient2: 96,66
- recipient3: 96,67
- комиссия: 10

Редактирование счета

Пример изменения списка услуг



QUERY	RESULT

mutation updateInvoice {
  obj: updateInvoice(
  id: "34b5c321-c086-4540-bca8-18f23580be7d"
  data: {
  dv: 1
  sender: { dv: 1, fingerprint: "playground" }
  rows: [{ name: "**Другая услуга**", count: 1, toPay: "100", isMin: false }]
  }
) {
  id
  number
  status
  organization {
  id
  name
  }
  property {
  id
  address
  }
  unitName
  unitType
  toPay
  rows {
  name
  toPay
  count
  isMin
  currencyCode
  sku
  }
	{
	  "data": {
	  "obj": {
	  "id": "34b5c321-c086-4540-bca8-18f23580be7d",
	  "number": 4,
	  "status": "draft",
	  "organization": {
	  "id": "e40b5367-49a8-4340-9eed-802538331326",
	  "name": "Тест маркетплейс"
	  },
	  "property": null,
	  "unitName": null,
	  "unitType": null,
	  "toPay": "100.00000000",
	  "rows": [
	  {
	  "name": "**Другая услуга**",
	  "toPay": "100",
	  "count": 1,
	  "isMin": false,
	  "currencyCode": "RUB",
	  "sku": null
	  }
	  ],
	  "createdAt": "2024-04-11T11:53:51.736Z",
	  "updatedAt": "2024-04-11T12:02:27.642Z",
	  "createdBy": {
	  "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
	  },
	  "updatedBy": {
	  "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
	  },
	  "deletedAt": null,

<table>
<tr>
<td>

```
    createdAt
    updatedAt
    createdBy {
      id
    }
    updatedBy {
      id
    }
    deletedAt
    publishedAt
    paidAt
  }
}
```

</td>
<td>

```
      "publishedAt": null,
      "paidAt": null
    }
  }
}
```

</td>
</tr>
</table>

Публикация счета

После публикации счет можно видеть в мобильном приложении и можно оплачивать.

Если в счете есть позиции с неявной ценой (isMin=true), то такой счет невозможно опубликовать.

АПИ запрос на публикацию - это то же самое что и редактирование, изменяется поле status на значение published.

Ссылка на оплату счета

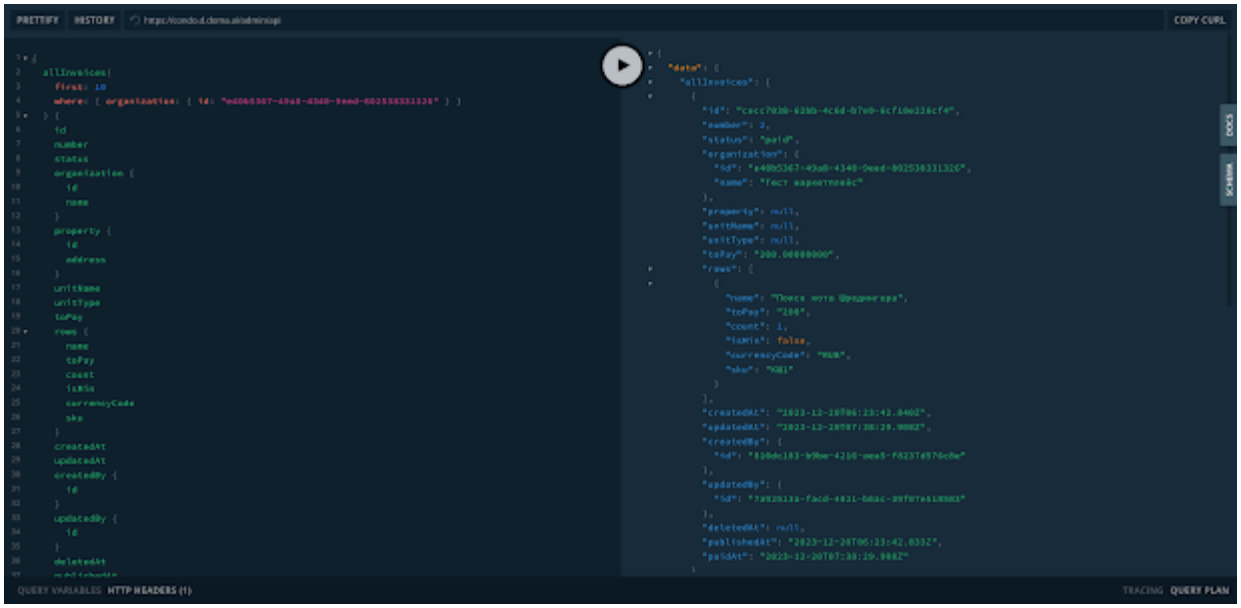
Шаблон ссылки на оплату выглядит вот так:

https://condo.d.doma.ai/payment-link?su=**{successUrl}**&fu=**{failureUrl}**&i=**{invoiceId}**

Поле	**Тип/формат**	**Пример**
successUrl	Строка, url encoded	https%3A%2F%2Fcondo.d.doma.ai%2Fmarketplace%2Finvoice%2Fpayment%2Fsuccess
failureUrl	Строка, url encoded	https%3A%2F%2Fcondo.d.doma.ai%2Fmarketplace%2Finvoice%2Fpayment%2Ffailure
invoiceId	Строка, UUID	14d2f5c9-de39-4302-9116-6abe501e54e1

При переходе по ссылке будут созданы эквайринговые сущности и пользователь будет перенаправлен на страницу ввода банковской карты. После оплаты пользователь будет перенаправлен на страницу результата (успех/ошибка)

Получение списка счетов (allInvoices)



QUERY	RESULT
{ allInvoices(first: 10 where: { organization: { id: "e40b5367-49a8-4340-9eed-802538331326" } }) { id number status organization { id name } property { id address } unitName unitType toPay rows { name toPay count isMin currencyCode sku } createdAt updatedAt createdBy { id	{ "data": { "allInvoices": [{ "id": "cecc7038-62bb-4c6d-b7e9-6cf10e226cf4", "number": 2, "status": "paid", "organization": { "id": "e40b5367-49a8-4340-9eed-802538331326", "name": "Тест маркетплейс" }, "property": null, "unitName": null, "unitType": null, "toPay": "200.00000000", "rows": [{ "name": "Поиск кота Шредингера", "toPay": "200", "count": 1, "isMin": false, "currencyCode": "RUB", "sku": "КШ1" }], "createdAt": "2023-12-20T06:23:42.840Z", "updatedAt": "2023-12-20T07:38:29.908Z", "createdBy": { "id": "810dc183-b9be-4216-aea5-f8237d576c8e" }, "updatedBy": { "id": "7a92513a-facd-4021-b6ac-39f07e518583" },

```
    }
    updatedBy {
      id
    }
    deletedAt
    publishedAt
    paidAt
  }
}
```


```
      "deletedAt": null,
      "publishedAt": "2023-12-20T06:23:42.833Z",
      "paidAt": "2023-12-20T07:38:29.908Z"
    },
    {
      "id": "54dfecdd-4478-4a0c-9bc3-bc8bbb79b383",
      "number": 1,
      "status": "canceled",
      "organization": {
        "id": "e40b5367-49a8-4340-9eed-802538331326",
        "name": "Тест маркетплейс"
      },
      "property": null,
      "unitName": null,
      "unitType": null,
      "toPay": "100.00000000",
      "rows": [
        {
          "name": "Тест услуги",
          "toPay": "100",
          "count": 1,
          "isMin": false,
          "currencyCode": "RUB",
          "sku": null
        }
      ],
      "createdAt": "2023-12-20T04:56:13.136Z",
      "updatedAt": "2024-02-18T06:11:00.023Z",
      "createdBy": {
        "id": "810dc183-b9be-4216-aea5-f8237d576c8e"
      },
      "updatedBy": null,
      "deletedAt": null,
      "publishedAt": "2023-12-20T04:56:24.693Z",
      "paidAt": null
    }
  ]
 }
}
```


В фильтрации запроса (where) можно указывать различные поля и их значения. Например, для фильтрации по дате укажите:

```
{
  AND: [{ createdAt_gte: "2024-03-31T19:00:00.000Z" }, { createdAt_lte: "2024-04-28T18:59:59.999Z" }]
}
```

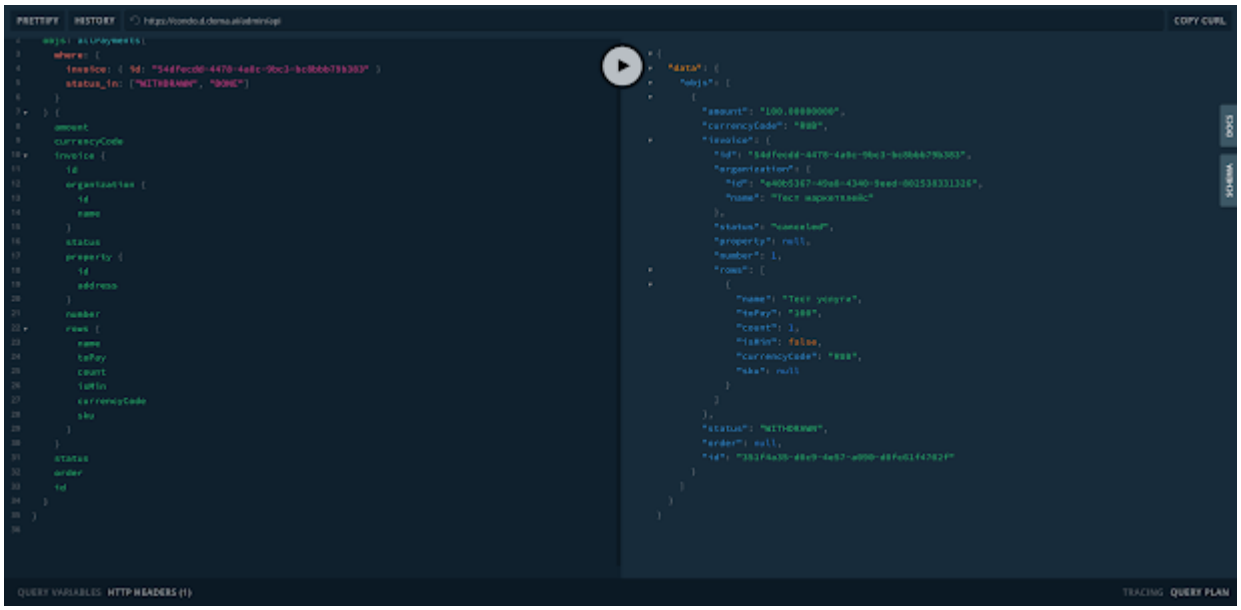
Проверка оплаты счета

Счет оплачен считается оплаченным, если существует платеж (payment) в статусе **WITHDRAWN** или **DONE** и указывающий на этот счет.

WITHDRAWN означает что деньги списаны с карты жителя

DONE означает что деньги отправлены банком на счет УК

Проверить наличие платежа для конкретного счета можно так:

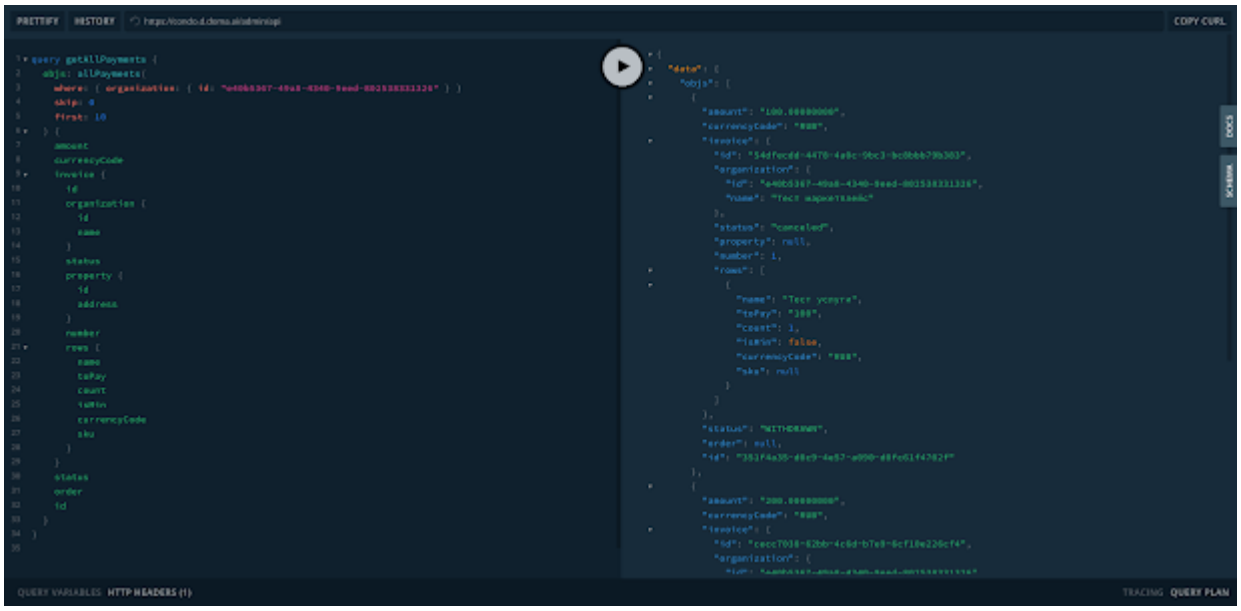


QUERY	RESULT

query getAllPayments {
objs: allPayments(
where: {
invoice: { id:
"54dfecdd-4478-4a0c-9bc3-bc8bbb79b383" }
status_in: ["**WITHDRAWN**", "**DONE**"]
}
) {
amount
currencyCode
invoice {
id
organization {
id
name
}
status
property {
id
address
}
number
rows {
name
toPay
count
isMin
currencyCode
sku
}
}
status
order
	{
	"data": {
	"objs": [
	{
	"amount": "100.00000000",
	"currencyCode": "RUB",
	"invoice": {
	"id": "54dfecdd-4478-4a0c-9bc3-bc8bbb79b383",
	"organization": {
	"id": "e40b5367-49a8-4340-9eed-802538331326",
	"name": "Тест маркетплейс"
	},
	"status": "canceled",
	"property": null,
	"number": 1,
	"rows": [
	{
	"name": "Тест услуги",
	"toPay": "100",
	"count": 1,
	"isMin": false,
	"currencyCode": "RUB",
	"sku": null
	}
	]
	},
	"status": "WITHDRAWN",
	"order": null,
	"id": "351f4a35-d0c9-4e57-a090-d0fc61f4702f"
	}
	]
	}
	}

id } }	

Если нужно получить все счета организации:



skip и first - это пагинация

QUERY	RESULT

query getAllPayments {
objs: allPayments(
where: { organization: { id: "e40b5367-49a8-4340-9eed-802538331326" } }
skip: 0
first: 10
) {
amount
currencyCode
invoice {
id
organization {
id
name
}
status
property {
id
address
}
number
rows {
name
toPay
	{
	"data": {
	"objs": [
	{
	"amount": "100.00000000",
	"currencyCode": "RUB",
	"invoice": {
	"id": "54dfecdd-4478-4a0c-9bc3-bc8bbb79b383",
	"organization": {
	"id": "e40b5367-49a8-4340-9eed-802538331326",
	"name": "Тест маркетплейс"
	},
	"status": "canceled",
	"property": null,
	"number": 1,
	"rows": [
	{
	"name": "Тест услуги",
	"toPay": "100",
	"count": 1,
	"isMin": false,
	"currencyCode": "RUB",
	"sku": null
	}

```
      count
      isMin
      currencyCode
      sku
    }
  }
  status
  order
  id
 }
}
```

```
      ]
     },
     "status": "WITHDRAWN",
     "order": null,
     "id": "351f4a35-d0c9-4e57-a090-d0fc61f4702f"
    },
    {
     "amount": "200.00000000",
     "currencyCode": "RUB",
     "invoice": {
      "id": "cecc7038-62bb-4c6d-b7e9-6cf10e226cf4",
      "organization": {
       "id": "e40b5367-49a8-4340-9eed-802538331326",
       "name": "Тест маркетплейс"
      },
      "status": "paid",
      "property": null,
      "number": 2,
      "rows": [
       {
        "name": "Поиск кота Шредингера",
        "toPay": "200",
        "count": 1,
        "isMin": false,
        "currencyCode": "RUB",
        "sku": "КШ1"
       }
      ]
     },
     "status": "WITHDRAWN",
     "order": null,
     "id": "3e1722df-aac7-4115-876b-17e2ce17300c"
    }
   ]
  }
}
```

🔐 Аутентификация и авторизация (#auth)

Данный раздел переехал: https://developers.doma.ai/ru/docs/api/examples/auth

🆔 Open ID Connect и мини-приложения

Если вы реализуете свое mini-приложение, то вам понадобится OIDC (OpenID Connect – authentication протокол, расширяющий OAuth 2.0 authorization framework)

Подробнее почитать про mini-приложения можно почитать в отдельной доке.

В примерах ниже, мы будем считать, что **mini.app.com** – адреса, где развернут ваше мини приложение, а **condo.d.doma.ai** – адрес, где развернут Condo OpenID Connect API.

Вам нужно реализовать на стороне вашего **mini.app.com** поддержку OIDC протокола. Для этого, вам нужно реализовать два API обработчика.

- mini.app.com**/oidc/auth** – адрес для начала аутентификации. Он будет формировать URL и делать редирект на наш OIDC сервер.
- mini.app.com**/oidc/callback** – адрес для завершения аутентификации. Наш OIDC сервер будет перенаправлять туда пользователей после успешной авторизации и аутентификации.

Вы должны передать нам адрес для начала OIDC аутентификации и, в примере выше, это mini.app.com**/oidc/callback**. Мы сгенерируем для вас **client_id** и **client_secret** необходимые для OIDC протокола.

Как это работает?

Пользователь, находясь на нашем сайте или в нашем мобильном приложении, может перейти в раздел с mini-приложениям. Он может зайти в mini-приложение, после чего, приложение может начать процесс аутентификации и авторизации.

- приложение или сайт делает запрос на mini.app.com**/oidc/auth**, где формируется перенаправление на наш OIDC сервер. Что-то вроде такого:
  https://condo.d.doma.ai**/oidc/auth**?**client_id=**<выданный-вам-client_id>**&scope**=openid&**response_type**=code%20id_token&**redirect_uri**=https://mini.app.com/oidc/callback&**nonce=**<уникальный-идентификатор>

  На этом шаге можно получить дополнительные данные о выбранной в интерфейсе пользователя организации, если мини-приложение открыто внутри интерфейса платформы, то мы добавим **condoOrganizationId** и **condoUserId**. Пример:
  mini.app.com**/oidc/auth**?**condoOrganizationId=**81cfe-3fcb-495f6cee&**condoUserId**=9b1cffc5-a26e-4a8a-a2b0-31ed9c0a5487

- наш OIDC сервер проверяет полученный запрос и формирует перенаправление на back-end часть вашего mini-приложения, передавая туда необходимые для OIDC протокола параметры. Что-то вроде такого:
  https://mini.app.com**/oidc/callback**#**code**=<oidc-code>&**id_token**=<oidc-id_token>

- ваш back-end mini-приложения использует полученный code, для получения access_token и refresh_token, делая запрос с использованием client_secret. Для этого, нужно отправить POST запрос на https://condo.d.doma.ai**/oidc/token** и передать туда
  grant_type=authorization_code&**code**=<oidc-code>&redirect_uri=https://mini.app.com/oidc/callback

  В ответ от OIDC сервера, будет получен access-token

Попробуем пройти этот процесс руками!

В качестве mini-приложения, мы будем использовать тестовый набор данных:

- **client_id**=oidc-condo-api-example
- **client_secret**=1b1b1b47043cd01d22a6febeedd14411

- **redirect_uri**=https://jwt.io/

1) Открываем в браузере https://condo.d.doma.ai/ и авторизуемся.
2) Переходим по ссылке:
   https://condo.d.doma.ai/oidc/auth?client_id=oidc-condo-api-example&scope=openid&response_type=code%20id_token&redirect_uri=https://jwt.io/&nonce=random

   Вас должно было перенаправить на:
   https://jwt.io/#code=9VMlcZABc2yE_-yFae_f4BAvIftC-nGrV4eKAPQshKm&id_token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImtleXN0b3JlLUNIQU5HRS1NRSJ9.eyJzdWIiOiI3ODgzYjEwMi01YzA4LTRiYzYtYTMxOC1jODVkYmEyMTkyZDYiLCJ2Ijo3LCJkdiI6MSwidHlwZSI6InN0YWZmIiwibmFtZSI6ItCR0LXQu9GL0Lkg0JzQsNC60YHQuNC8INCh0LXRgNCz0LXQtdCy0LjRhyIsImlzQWRtaW4iOmZhbHNlLCJpc1N1cHBvcnQiOmZhbHNlLCJub25jZSI6InJhbmRvbSIsImF0X2hhc2giOiJHX0FVMi1HR3dMTi16bHlPSmUyR3ZBIiwiYXVkIjoib2lkYy1jb25kby1hcGktZXhhbXBsZSIsImV4cCI6MTY1NDE2MTIxOSwiaWF0IjoxNjU0MTU3NjE5LCJpc3MiOiJodHRwczovL2NvbmRvLmQuZG9tYS5haSJ9.XAVsR_fESOnVbtMIaroT6PTXQSHShMCY1yBBCXHMIb7iFmKEk5NJ1QzmYcfvxzTeluNUXXAoHpIF4ksvvKmI03YXkvwysutI21vhtjrXnzaGmE6-9PvP3CSu5jog50OiaBEcAb98-LOQDaLsVcQrgCrZxnhhk5r0qBo2sD83vtTv_tsDIX3mv6HpGpNHFy4NopB7CmaeoUNxQOu5t-cyxsMfeIoPXHvXb_txUmYK4yk6ltorWaovcA3pVfSj0BNV1v4o-Ww-aA5RCmAAdZ4zZph89qO52UTgSK7hpQSS0Lz_fQil-A56kkgubCq_SUWcEiLWqS9dKK4XJNoSBLU4oA

3) Используем полученный code для обмена его на **access_token** через curl запрос:

```
curl https://condo.d.doma.ai/oidc/token -d
'code=9VMlcZABc2yE_-yFae_f4BAvIftC-nGrV4eKAPQshKm' -d
'client_id=oidc-condo-api-example' -d
'client_secret=1b1b1b47043cd01d22a6febeedd14411' -d
'redirect_uri=https://jwt.io/' -d
'grant_type=authorization_code' -X POST
```

   Пример ответа от сервера:


```
{"access_token":"b8pQyoMuoLfqq1nxUpfW0zWVYSZPZ9BRavKo7SAtyRj","e
xpires_in":31536000,"id_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpX
VCIsImtpZCI6ImtleXN0b3JlLUNIQU5HRS1NRSJ9.eyJzdWIiOiI4MTBkYzE4My1
iOWJlLTQyMTYtYWVhNS1mODIzN2Q1NzZjOGUiLCJ2Ijo0LCJkdiI6MSwidHlwZSI
6InN0YWZmIiwibmFtZSI6ItCQ0LvQtdC60YHQsNC90LTRgCDQndCw0LnQtNGR0L3
QvtCyIiwiaXNBZG1pbiI6dHJ1ZSwiaXNTdXBwb3J0Ijp0cnVlLCJub25jZSI6InJ
hbmRvbSIsImF0X2hhc2giOiJKV0ZvZlUxVEtQcWxlR2wzSmdxaEhRIiwiYXVkIjo
ib2lkYy1jb25kby1hcGktZXhhbXBsZSIsImV4cCI6MTY3MzU4OTQ4NSwiaWF0Ijo
xNjczNTg1ODg1LCJpc3MiOiJodHRwczovL2NvbmRvLmQuZG9tYS5haSJ9.QxR2Zn
4umvvZpy3bffBax8kVyDZkd7CpCWSZsUDq13m5p66NHBAuW0Jmt_WdPAUyAOJ7Be
```

ckU_0lY-Wu1g4CpGnB6a6saLXxTwnij2tnwV3q2HnvZUFFIKhyCvpRoUfDvpPZy6
P8euZfI859jOZ_BAsCVo19bvZvK8P5KOXDwyIA43EZBIeA4ynZYexjDRr6E7NefG
CBoVK_ubceUTPwMUQBUmrJ8KRDCORt-0-Dc31fSdRzqXMmw-8fX2SmFg_KVji-Kh
amqmUUlODLTrSkp8NMK5v9DYo8GGkAzLJ4YYpPzTnoOgP8HJo87EH7D5haM4gNjY
ct3SsETOnLCH2xBQ","refresh_token":"ojzGUmkHLI3eUe0xBDSiuTJdRaaFa
vhDhazTVu8VxiM","scope":"openid","token_type":"Bearer"}

4) Теперь мы можем выполнять запросы от имени пользователя взаимодействующего с mini-приложением используя **access_token**

```
curl 'https://condo.d.doma.ai/admin/api' -H 'Authorization:
Bearer b8pQyoMuoLfqq1nxUpfW0zWVYSZPZ9BRavKo7SAtyRj' -H
'Content-Type: application/json' --data-raw
'{"variables":{},"query":"query { authenticatedUser { id name
avatar { publicUrl __typename } phone email isAdmin __typename
}}"}'
```

Пример ответа:

{"data":{"authenticatedUser":{"id":"7883b102-5c08-4bc6-a318-c85d
ba2192d6","name":"Белый Максим
Сергеевич","avatar":null,"phone":"+79068088888","email":"no-repl
ay@doma.ai","isAdmin":false,"__typename":"User"}}}

5) access_token имеет ограниченный срок жизни. Ориентировочно 1 час. Если попытаться сделать запрос с истекшим токеном, то в ответ вернется код **401**. Сэмулировать это можно с помощью заранее неверного access-токена

```
curl 'https://condo.d.doma.ai/admin/api' -H 'Authorization:
Bearer WRONGoMuoLfqq1nxUpfW0zWVYSZPZ9BRavKo7SAtyRj' -H
'Content-Type: application/json' -I
```

В команду специально добавлен ключ -I чтобы увидеть заголовки ответа:

```
HTTP/2 401
date: Fri, 13 Jan 2023 05:18:38 GMT
content-type: text/plain; charset=utf-8
content-length: 12
x-powered-by: Express
x-request-id: 4fda96ab741cdf4bf9b45b0fe23c6b67
vary: Origin
access-control-allow-credentials: true
etag: W/"c-dZuWFQkdtS3hezqxDTNgH8AJlYk"
strict-transport-security: max-age=31536000; preload
```

Для обновления токена нужно использовать refresh_token:

```
curl 'https://condo.d.doma.ai/oidc/token' -H 'Authorization:
Basic
b2lkYy1jb25kby1hcGktZXhhbXBsZToxYjFiMWI0NzA0M2NkMDFkMjJhNmZlYmVl
ZGQxNDQxMQ==' -H 'Content-Type:
application/x-www-form-urlencoded' -d
'grant_type=refresh_token&refresh_token=ojzGUmkHLI3eUe0xBDSiuTJd
RaaFavhDhazTVu8VxiM'
```

Пример ответа, содержащего новые токены:

```
{"access_token":"hxCH6csLn3ZFH0_c9_3SrkkfCJWOQL7zfw0cRvO0fIB","e
xpires_in":31536000,"id_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpX
VCIsImtpZCI6ImtleXN0b3JlLUNIQU5HRS1NRSJ9.eyJzdWIiOiI4MTBkYzE4My1
iOWJlLTQyMTYtYWVhNS1mODIzN2Q1NzZjOGUiLCJ2Ijo0LCJkdiI6MSwidHlwZSI
6InN0YWZmIiwibmFtZSI6ItCQ0LvQtdC60YHQsNC90LTRgCDQndCw0LnQtNGR0L3
QvtCyIiwiaXNBZG1pbiI6dHJ1ZSwiaXNTdXBwb3J0Ijp0cnVlLCJub25jZSI6InJ
hbmRvbSIsImF0X2hhc2giOiJhYUFmUG1Fd0l4R3RjRUwyVm5lTXNRIiwiYXVkIjo
ib2lkYy1jb25kby1hcGktZXhhbXBsZSIsImV4cCI6MTY3MzU5Mzk1MiwiaWF0Ijo
xNjczNTkwMzUyLCJpc3MiOiJodHRwczovL2NvbmRvLmQuZG9tYS5haSJ9.XJG3oA
ao3nrwoj2cBOIcmzbtIftx-0WMUardWiGLvB9dgndrZS0mSoohlyt-RQBkF8oyrH
2NYpooEcFda7Sj4iGy0acvCg6juuVKFTGQZ_EdcaP19asS97S6ccnSLSObBlmU9x
ZcVNqO_oETgje4wIHZ5_4ohnmf-3SDc_oIRIFhgtOYH9dnVNBtoPiM17L3zetoR_
A6hq1rN6GkKha56utr1fPrshDUWMhk1R2cuMlpuYAO8gjw8zxRvsnVx_XChcbYUD
9daxlXDJ4A91bYPi7fr-lCxcPhE5ybymAyEdHqqv3DZGhL-umdowd7mkY04NkDTI
lTFNEmLaAz5gxpMg","refresh_token":"ojzGUmkHLI3eUe0xBDSiuTJdRaaFa
vhDhazTVu8VxiM","scope":"openid","token_type":"Bearer"}
```

Обратите внимание на заголовок Authorization. Его значение - это base64-encoded строка “<client_id>:<client_secret>”. Для нашего примера это будет так:

```
echo -n
'oidc-condo-api-example:1b1b1b47043cd01d22a6febeedd14411' |
base64
```

Получится значение для basic authorization:

```
b2lkYy1jb25kby1hcGktZXhhbXBsZToxYjFiMWI0NzA0M2NkMDFkMjJhNmZlYmVl
ZGQxNDQxMQ==
```

Тестовые client_id, client_secret

На тестовом сервере https://condo.d.doma.ai/ созданы OIDC клиенты для разработки локальной разработки или тестирования протокола.

Набор с редиректом на jwt.io из примеров выше

ClientID: oidc-condo-api-example

ClientSecret: 1b1b1b47043cd01d22a6febeedd14411

CallbackUri: https://jwt.io/

Набор с редиректом на localhost порт 3000 по https

Согласно общей политике безопасности, редирект не может быть на домен localhost! Поэтому, мы добавили редирект на домен doma.local. Если вы тестируете авторизацию локально, то можете прописать на своем компьютере в hosts домен doma.local указывающий на 127.0.0.1. Настроить локальные сертификаты SSL и использовать следующие настройки.

ClientID: oidc-https-localhost3000-example

ClientSecret: nAG5uDcu1eyIkqajRTEpxYcsGUcdkl30

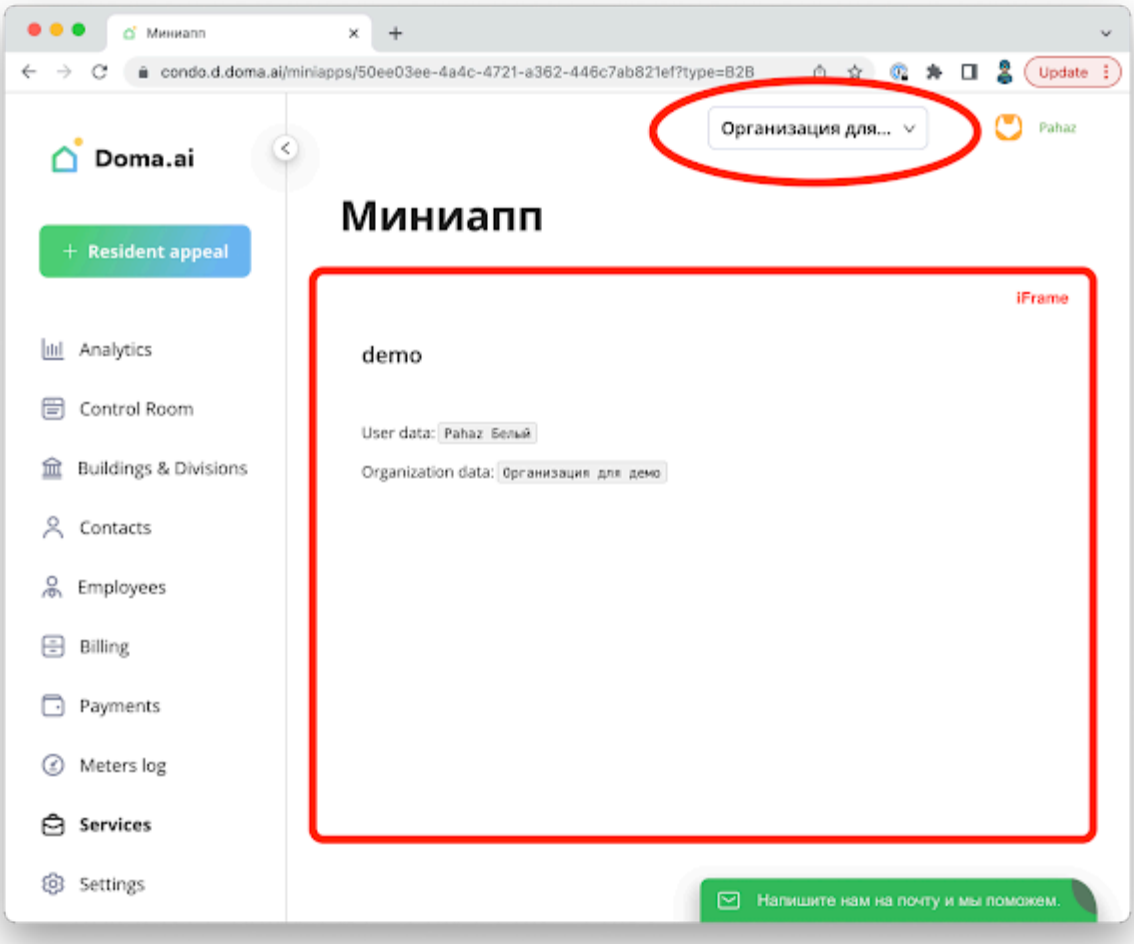
CallbackUri: https://doma.local:3000/oidc/callback, https://doma.local:8000/oidc/callback, https://doma.local:8001/oidc/callback, https://doma.local:8002/oidc/callback, https://doma.local:8003/oidc/callback, https://doma.local:8004/oidc/callback, https://doma.local:8005/oidc/callback, https://doma.local:3000/api/oidc/callback, https://doma.local:8000/api/oidc/callback, https://doma.local:3000/api/oidc, https://doma.local:3000/api/callback, https://doma.local:3000/api/auth/doma, https://doma.local:3000/api/auth/condo

Примеры реализации

Если вы реализуете открытый пример, то напишите о нем нам и мы включим его в этот список. Пример на node.js + express + openid-client.

Получение информации о текущей организации для B2B приложений

Если вы реализуете mini-приложения, вам может понадобится информация о выбранной пользователем организации.



Эту информацию можно получить из get параметров при открытии iFrame мини-приложения. В момент открытия приложения, мы добавляем к адресу condoOrganizationId и condoUserId. Пример:
mini.app.com**/oidc/auth?condoOrganizationId=**81cfe-3fcb-495f6cee&**condoUserId=**9b1cffc5-a26e-4a8a-a2b0-31ed9c0a5487

Отправка пушей для B2C приложений

Иногда, B2C приложениям необходимо отправлять пуш с данными. Например, домофония отправляет пуш в момент звонка в домофон.

Для отправки пуша нам понадобится авторизационный токен, который вы получите в момент открытия пользователем мини-приложения после выполнения OpenID авторизации. Чтобы отправить пуш в мини-приложение, пользователь должен его открыть.

Для отправки пушей необходимо использовать мутацию sendB2CAppPushMessage, передав в нее идентификатор приложения, от имени которого будет отправляется пуш и идентификатор пользователя которому этот пуш будет отправлен.

С точки зрения системы прав, получается, что пользователь сам себе отправляет пуш-уведомление в мини-приложение.

Мы даем отправлять три типа пушей (type):

- VOIP_INCOMING_CALL_MESSAGE – voip-пуши для звонков
- CANCELED_CALL_MESSAGE_PUSH – пуш для отмены voip-пуша
- B2C_APP_MESSAGE_PUSH – обычные сообщения с данными

Вы не сможете отправлять пуши слишком часто. Если вы пытаетесь отправлять пуши слишком часто, то отправка пушей для вашего приложения может быть заблокирована. Вы можете отправить voip-пуш и пуш для отмены voip-пуша - спустя 2 секунды после предыдущего, а сообщение с данными только спустя 3600 секунд.

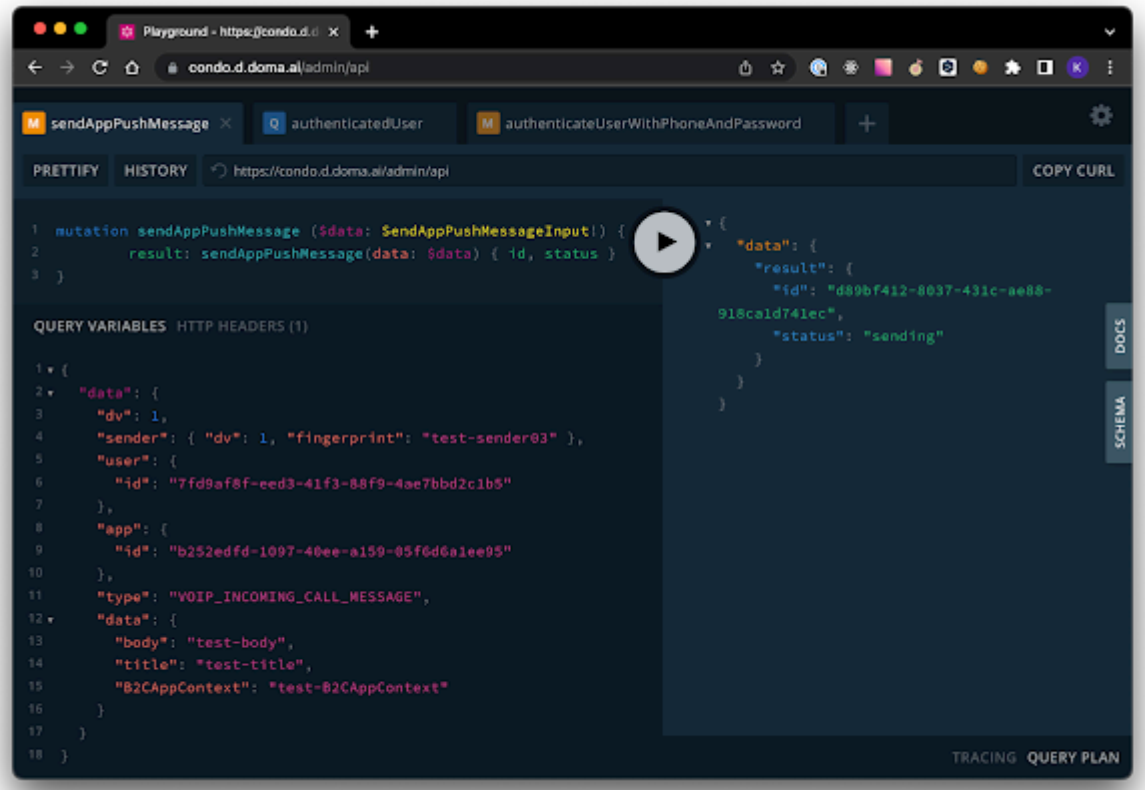
QUERY

```
mutation sendB2CAppPushMessage ($data: SendB2CAppPushMessageInput!) {
   result: sendB2CAppPushMessage(data: $data) { id, status }
}
```

```
{
  "data": {
    "dv": 1,
    "sender": { "dv": 1, "fingerprint": "test-sender" },
    "user": {
            "id": "2de1d71b-5366-4330-870d-ed3204a53d51"
    },
    "app": {
            "id": "4dbd7c48-a066-4482-8415-c10500f8a0c8"
    },
    "type": "CANCELED_CALL_MESSAGE_PUSH",
    "data": { "body": "test-body", "title": "test-title", "B2CAppContext": "test-B2CAppContext", "voipIncomingCallId": "e4a1fa8d-eec1-4b9b-b394-b668b4808df3" }
  }
}
```

RESULT

```
{
  "data": {
    "result": {
      "id": "e4a1fa8d-eec1-4b9b-b394-b668b4808df3",
      "status": "sending"
    }
  }
}
```



QUERY	RESULT

mutation sendAppPushMessage ($data: SendAppPushMessageInput!) {
result: sendAppPushMessage(data: $data) { id, status }
}
	{
	"data": {
	"result": {
	"id": "e4a1fa8d-eec1-4b9b-b394-b668b4808df3",
	"status": "sending"
	}
	}
	}

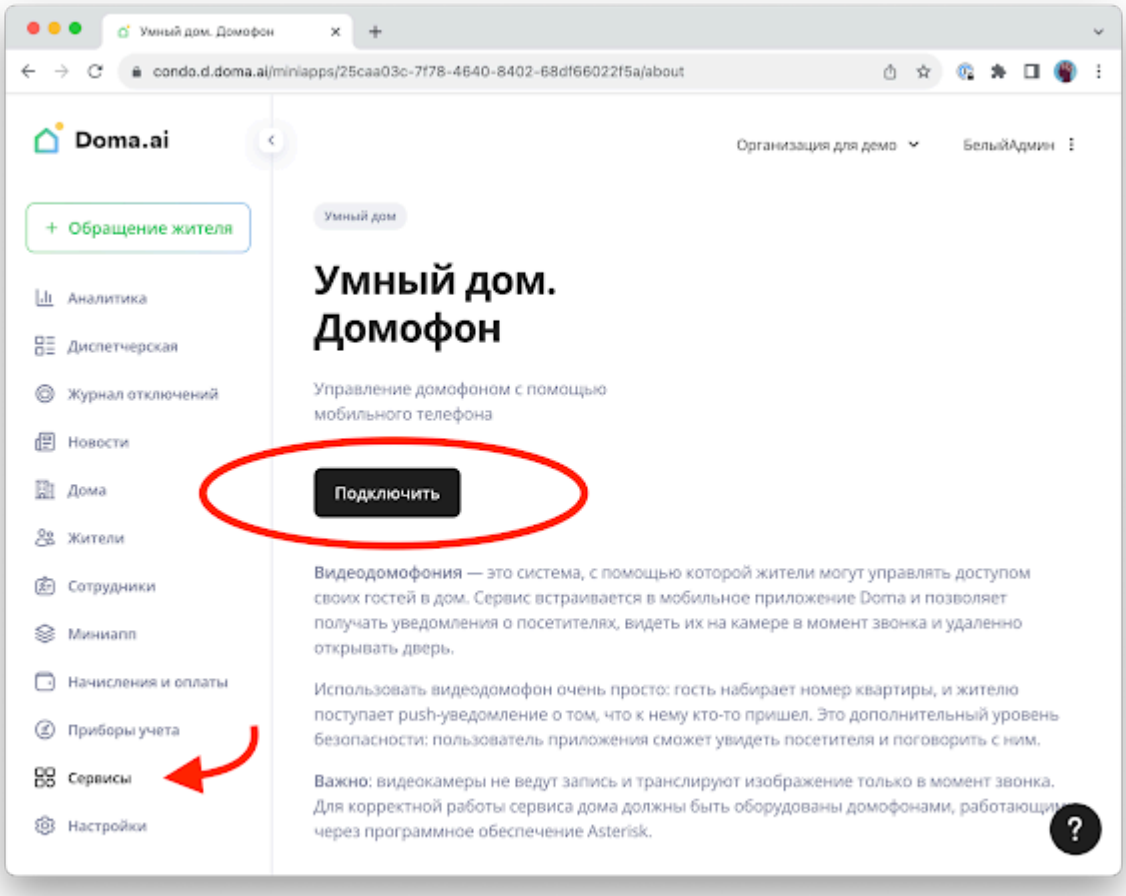
{
"data": {
"dv": 1,
"sender": { "dv": 1, "fingerprint": "test-sender" },
"user": {
"id": "2de1d71b-5366-4330-870d-ed3204a53d51"
},
"app": {
"id": "4dbd7c48-a066-4482-8415-c10500f8a0c8"
},
"type": "VOIP_INCOMING_CALL_MESSAGE",
"data": { "body": "test-body", "title": "test-title", "B2CAppContext": "test-B2CAppContext" }
}	

}	

Отображение B2C приложения на адресах

Если вы разрабатываете мини-приложение для мобильного приложения жителя (B2C мини-приложение), то вам важно понимать, на каких адресах оно будет доступно.

Рассмотрим на примере кейса в домофонией: есть B2B приложение “Домофон”, которое подключает администратор УК в разделе сервисы.



Данное приложение состоит из двух частей: B2B-часть (которую подключает УК) и B2C-часть (которую открывает житель в мобильном приложении)

Таким образом в платформе дома должно существовать B2B-часть и B2C-часть приложения “домофон”.

Оценка заявки (контроль качества)

Сотрудники организации могут выставлять оценки заявкам после обзвона/опроса жителей о выполненных работах.

Для этого у тикета (Ticket) есть поля:

- qualityControlValue – оценка по 2-балльной шкале
- qualityControlComment – текстовый комментарий к оценке
- qualityControlAdditionalOptions – дополнительный набор заготовленных уточнения к оценке

Также есть поля только для чтения, которые заполняются автоматически при обновлении любого из вышеуказанных полей:

- qualityControlUpdatedAt – последнее время обновления оценки
- qualityControlUpdatedBy – последний сотрудник организации, который обновил оценку

Какие бывают оценки (qualityControlValue):

- good – хорошо
- bad – плохо

Какие бывают заготовленные уточнения к оценке (qualityControlAdditionalOptions):

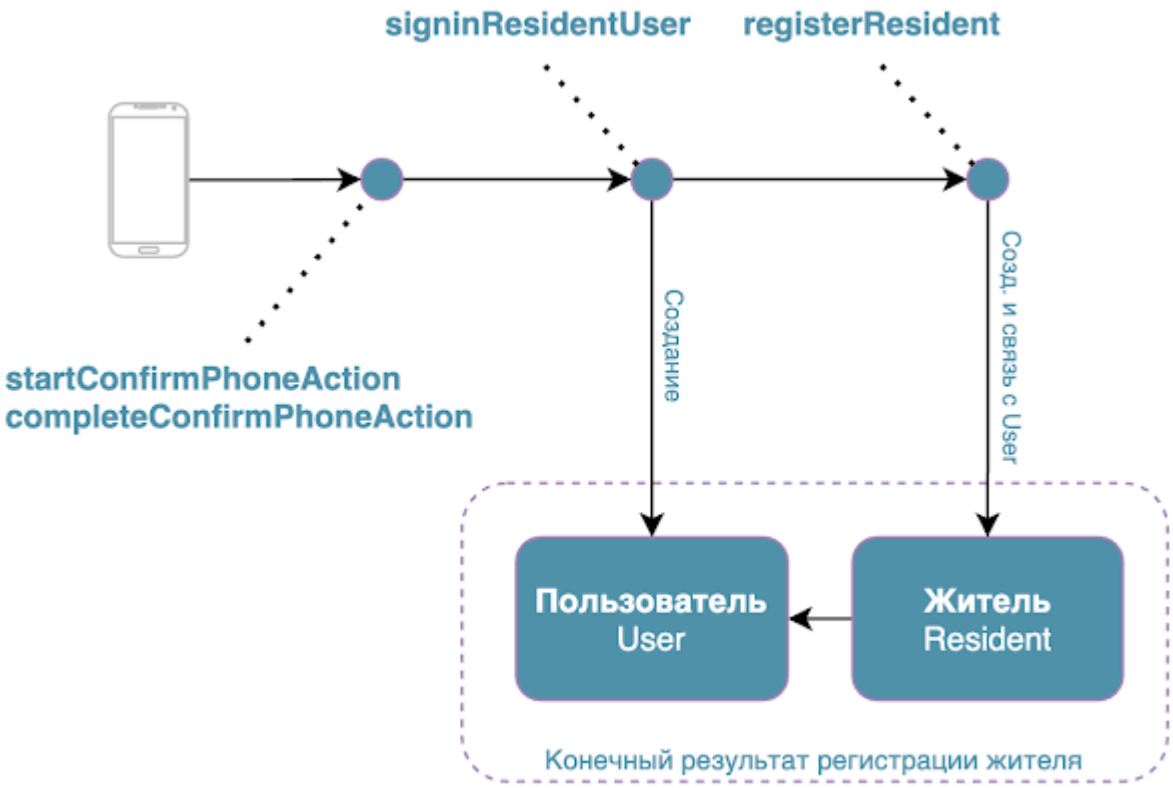
- lowQuality – Низкое качество
- highQuality – Высокое качество

- slowly – Выполнено медленно
- quickly – Выполнено быстро

📲 Мобильное приложение жителя

В этом блоке будут описаны особенности работы с API со стороны мобильного приложения жителя. Жители имеют свой собственный механизм аутентификации и авторизации. Данный раздел подходит только для пользователей (User) с типом (type) житель (‘resident’).

Общий сценарий регистраций работы приложения жителя:



1) Проходим процедуру верификации номера телефона (startConfirmPhoneAction и completeConfirmPhoneAction)
2) Создаем пользователя или входим, если такой пользователь уже зарегистрирован используя signinResidentUser

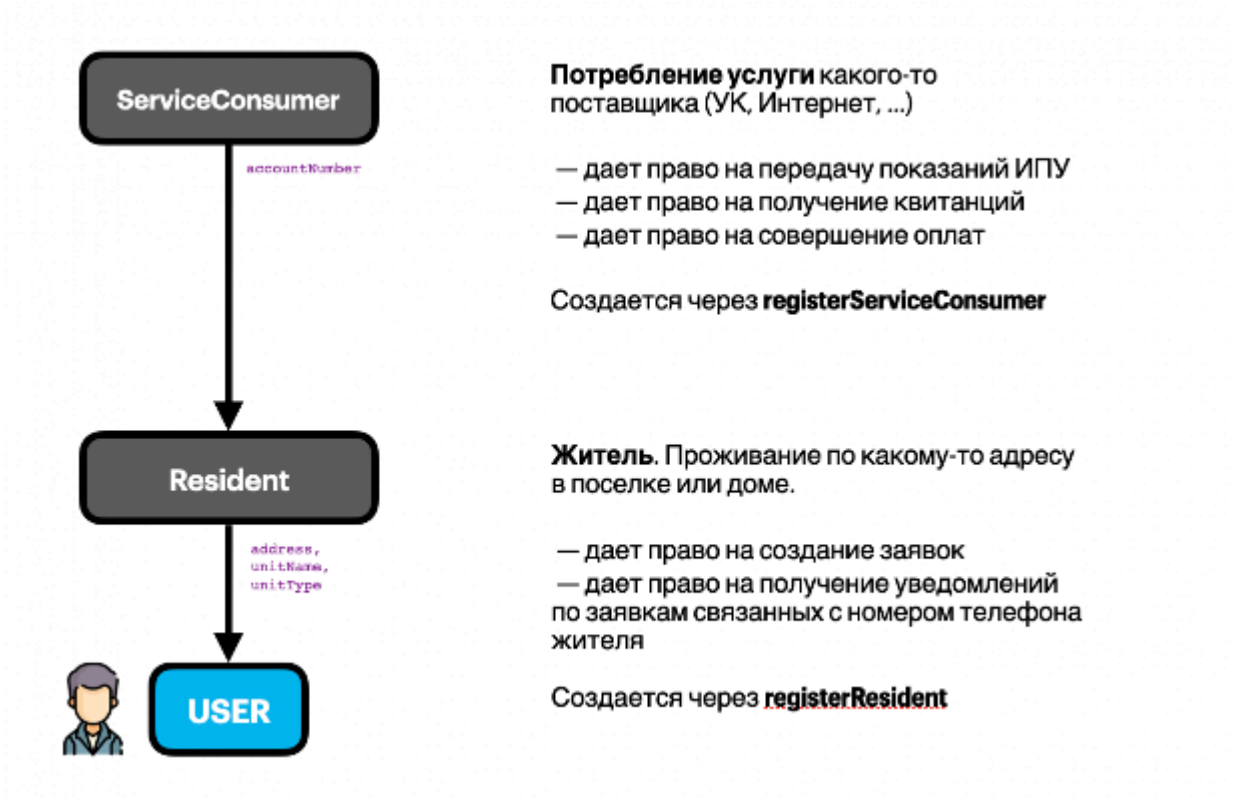
На втором шаге, мы получим User и сможем выполнять от его имени запросы. Можно проверить, что мы прошли аутентификацию, используя authenticatedUser

3) Следующим шагом, пользователь мобильного приложения, должен ввести адрес объекта (address) общей собственности, где он живет, это делается через registerResident. На этом шаге, необходимо указать адрес, номер помещения и его тип

На этом шаге, мы создали сущность Resident, она показывает, что данный пользователь является жителем и связываем его с конкретным адресом, номером (unitName) и типом помещения (unitType). Это дает ему возможность создавать заявки, если по данному адресу в нашем сервисе есть организация (Organization) и она добавила этот адрес в список своих объектов управления (Property)

4) Следующим шагом, нам необходимо передать информацию о лицевом счете, это открывает возможность для данного жителя получать квитанции и совершать оплаты, это делается через registerServiceConsumer. Для этого, необходимо передать номер лицевого счета (accountNumber)

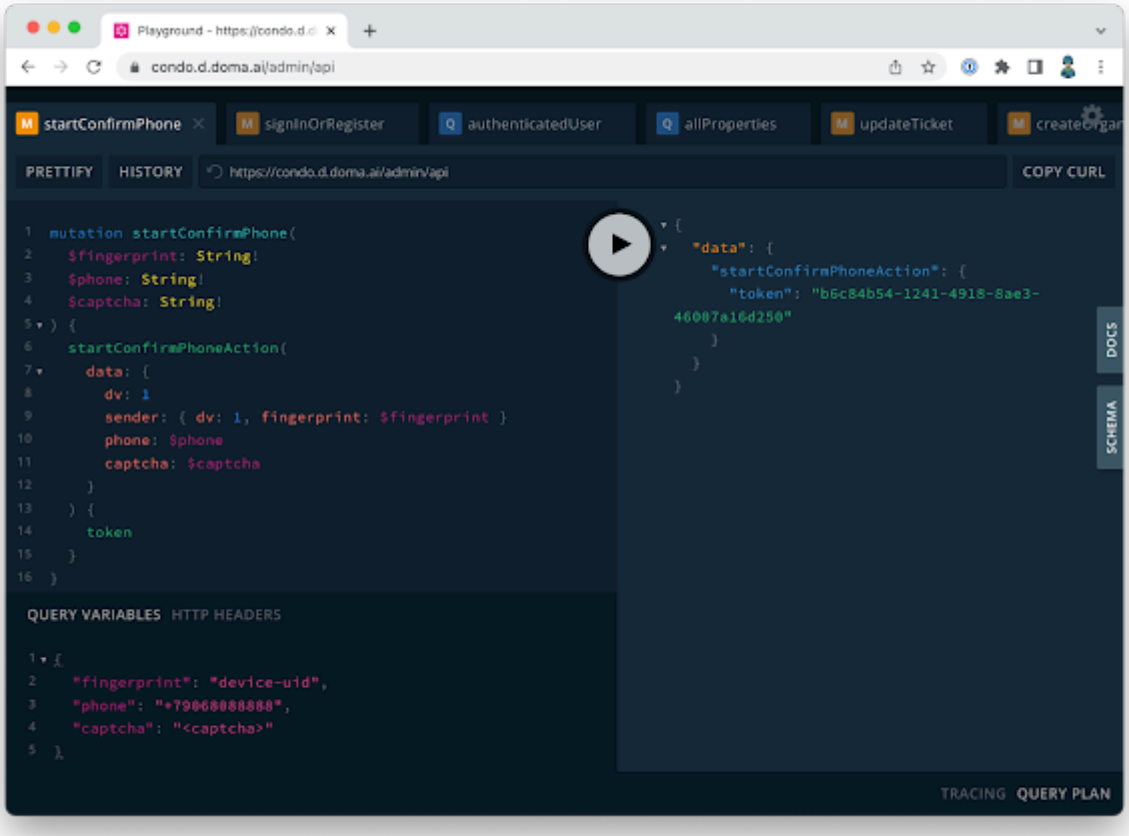
Подведем итог:



startConfirmPhone

Используйте мутацию startCofirmPhone для подтверждения номера телефона. В ответе приходит токен, а на номер телефона отправляется смс сообщение с кодом.

- $fingerprint – уникальный идентификатор устройства, подробнее в DOMA API
- $phone – номер телефона в E.164 формате
- $captcha – текстовая капча, мы заведем для вас уникальный идентификатор



QUERY	RESULT

mutation startConfirmPhone(
$fingerprint: String!
$phone: String!
$captcha: String!
) {
startConfirmPhoneAction(
data: {
dv: 1
sender: { dv: 1, fingerprint: $fingerprint }
phone: $phone
captcha: $captcha
}
) {
token
}
}
	{
	"data": {
	"startConfirmPhoneAction": {
	"token": "b6c84b54-1241-4918-8ae3-46007a16d250"
	}
	}
	}

{ "fingerprint": "<device-uid>", "phone": "+79068088888", "captcha": "<captcha>"	

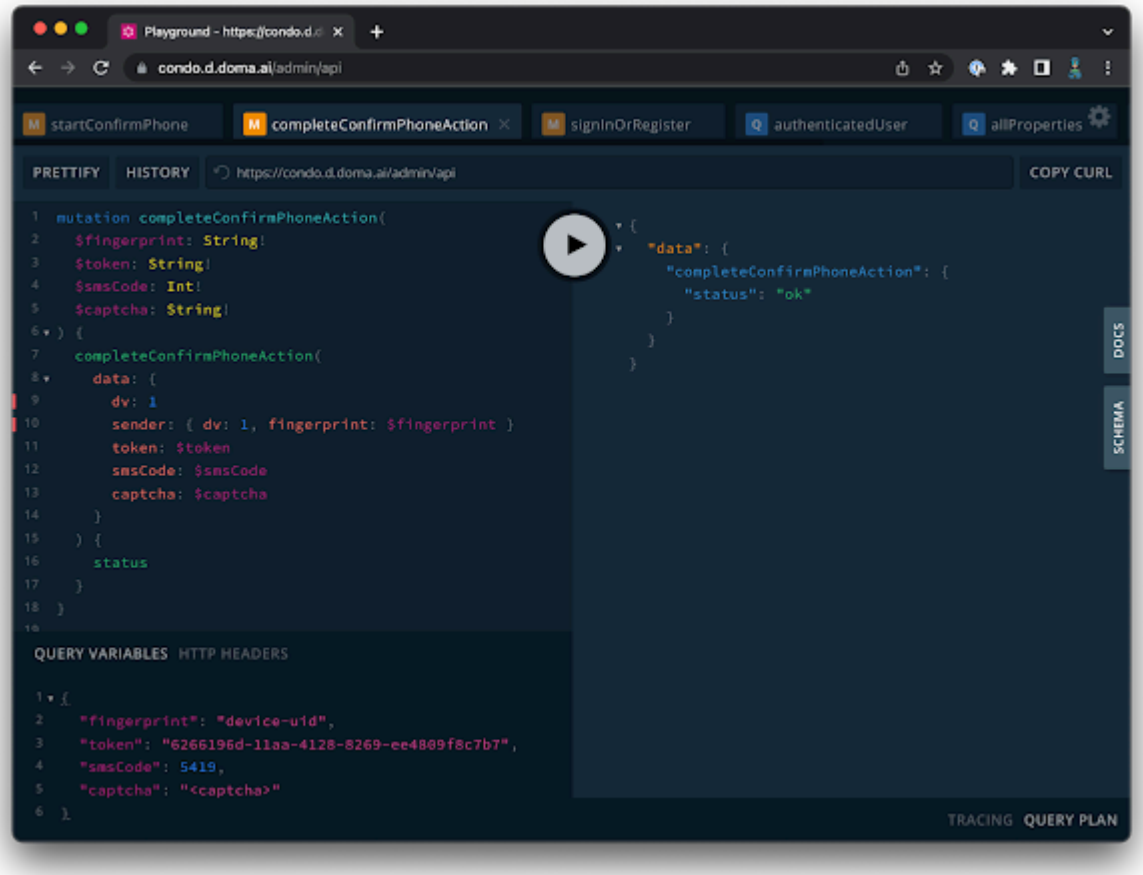
}	

Код, отправленный через смс нужно использовать в completeConfirmPhoneAction, чтобы завершить процедуру подтверждения номера телефона.

completeConfirmPhoneAction

Пользователь получает СМС сообщение с кодом, вводит его в интерфейс, после этого, необходимо вызвать completeConfirmPhoneAction и передать в него код из СМС.

- $fingerprint – уникальный идентификатор устройства, подробнее в DOMA API
- $token – токен, полученный на шаге startConfirmPhone
- $smsCode – код, четырехзначное число из СМС сообщения
- $captcha – текстовая капча, мы заведем для вас уникальный идентификатор



QUERY	RESULT

```
mutation completeConfirmPhoneAction(
  $fingerprint: String!
  $token: String!
  $smsCode: Int!
  $captcha: String!
) {
  completeConfirmPhoneAction(
    data: {
      dv: 1
      sender: { dv: 1, fingerprint: $fingerprint }
      token: $token
      smsCode: $smsCode
      captcha: $captcha
    }
  ) {
    status
  }
}
```

```
{
  "fingerprint": "<device-uid>",
  "token": "b6c84b54-1241-4918-8ae3-46007a16d250",
  "smsCode": 5419,
  "captcha": "<captcha>"
}
```

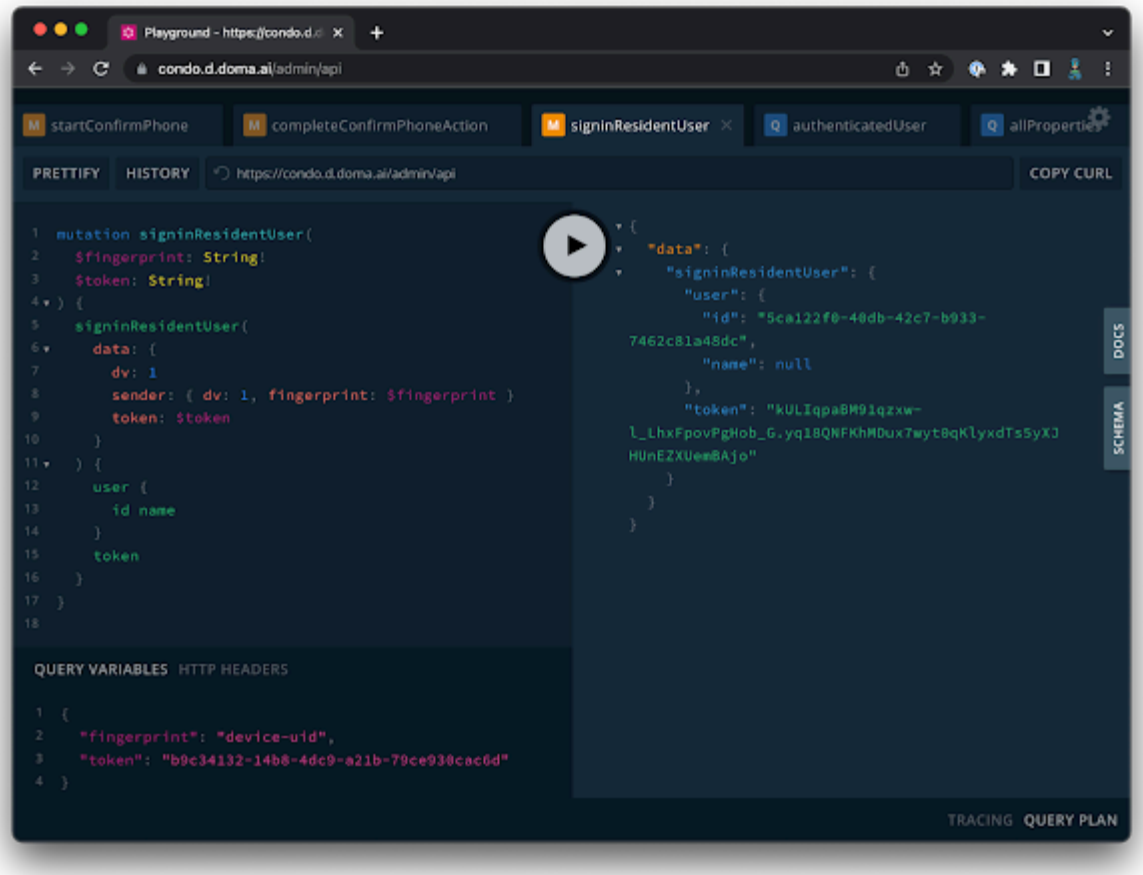
```
{
  "data": {
    "completeConfirmPhoneAction": {
      "status": "ok"
    }
  }
}
```

signinResidentUser

Имея подтвержденный токен, можно выполнить процедуру входа жителя по номеру телефона. Для этого, нужно использовать signinResidentUser и передать туда токен, который уже прошел через completeConfirmPhoneAction.

Данная мутация создаст жителя (User), если это необходимо, и осуществит вход. В ответе будут данные пользователи и его токен. Мутация signinResidentUser работает аналогично мутации authenticateUserWithPhoneAndPassword.

- $fingerprint – уникальный идентификатор устройства, подробнее в DOMA API
- $token – токен, полученный на шаге startConfirmPhone и подтвержденный через completeConfirmPhoneAction



QUERY	RESULT

mutation signinResidentUser(
$fingerprint: String!
$token: String!
) {
signinResidentUser(
data: {
dv: 1
sender: { dv: 1, fingerprint: $fingerprint }
token: $token
}
) {
user {
id name
}
token
}
}
	{
	"data": {
	"signinResidentUser": {
	"user": {
	"id": "5ca122f0-40db-42c7-b933-7462c81a48dc",
	"name": null
	},
	"token": "kULIqpaBM91qzxw-l_LhxFpovPgHob_G.yq18QNFKhMDux7wyt0qKlyxdTs5yXJHUnEZXUemBAjo"
	}
	}
	}

{
"fingerprint": "<device-uid>",
"token": "b6c84b54-1241-4918-8ae3-46007a16d250"
}	

После успешного выполнения мутации, мы установим вам **cookie** сессию. Из этой же мутации вы сможете получить **bearer token** и использовать его для выполнения запросов от имени жителя. Пример использования токен можно найти тут

API Квоты для партнеров

Частота: не более **1 запроса в секунду** с одного API-ключа.

Объём данных: внутри любого запроса допускается обрабатывать **до 100 объектов** (создание, обновление или выборка).

Размер GraphQL запроса:

- поле `query` — **до 1 024 символов**,
- поле `variables` — **до 128 КБ**.

При превышении лимитов платформа вернёт HTTP 429 или соответствующую ошибку GraphQL.

Служебные атрибуты (#1)

Набор служебных атрибутов используемых для всех объектов:

- id -- уникальный идентификатор объекта
- createdAt -- дата создания объекта, нельзя подменить, задается автоматически
- updatedAt -- дата обновления объекта, нельзя подменить, задается автоматически
- createdBy -- привязка к пользователю создавшему объект, нельзя подменить
- updatedBy -- привязка к пользователю обновившему объект, нельзя подменить
- deletedAt -- привязка к пользователю удалившему объект, нельзя подменить
- newId -- для склейки объектов (смотри в разделе Склейка/Миграция (#5))
- dv -- версия структуры данных (смотри в разделе Версия структуры данных (#6))
- v -- версия данных. Счетчик, который увеличивается при любых изменениях
- importId -- идентификатор внешней системы, используется при импорте данных в наш сервис из других систем

Параметры фильтрации и сортировки (#2)

Работая со списками объектов, необходимо использовать параметры where, sortBy.

where

Данный параметр используется для фильтрации объектов по некоторым атрибутам, например, мы хотим отфильтровать пользователей у которых имя начинается с буквы “A” без учета регистра:

QUERY	RESULT

```
query {
  allUsers (where: {name_starts_with_i: "A"}) {
    id name __typename
  }
}
```

```
{
  "data": {
    "allUsers": [
      {
        "id": "121fe-3f3b-495f6cee",
        "__typename": "User",
        "name": "Anna"
      }
    ]
  }
}
```

В зависимости от типа данных атрибута, будут доступны разные возможности по фильтрации данных, давайте посмотрим на примеры.

Фильтрация по связанным объектам (relationship)

- {fieldName}_every: whereInput
- {fieldName}_some: whereInput
- {fieldName}_none: whereInput
- {fieldName}_is_null: Boolean

Фильтры по строковым типам

- {Field}: String
- {Field}_not: String
- {Field}_contains: String
- {Field}_not_contains: String
- {Field}_starts_with: String
- {Field}_not_starts_with: String
- {Field}_ends_with: String
- {Field}_not_ends_with: String
- {Field}_i: String
- {Field}_not_i: String
- {Field}_contains_i: String
- {Field}_not_contains_i: String
- {Field}_starts_with_i: String
- {Field}_not_starts_with_i: String
- {Field}_ends_with_i: String
- {Field}_not_ends_with_i: String
- {Field}_in: [String]
- {Field}_not_in: [String]

Фильтр по полю ID

- {Field}: ID
- {Field}_not: ID
- {Field}_in: [ID!]

- {Field}_not_in: [ID!]

Фильтры по числовым полям

- {Field}: Int
- {Field}_not: Int
- {Field}_lt: Int
- {Field}_lte: Int
- {Field}_gt: Int
- {Field}_gte: Int
- {Field}_in: [Int]
- {Field}_not_in: [Int]

Использование логических операций AND и OR

- AND: [whereInput]
- OR: [whereInput]

Вы можете комбинировать несколько условий:

QUERY	RESULT

query {
  allUsers (where: {**OR**: [{ name_starts_with_i: "A" }, { email_starts_with_i: "A" }]}) {
  id name __typename
  }
}
	{
	  "data": {
	  "allUsers": [
	  {
	  "id": "121fe-3f3b-495f6cee",
	  "__typename": "User",
	  "name": "Anna"
	  }
	  ]
	  }
	}

Черновики

Формат ошибок (#3)

error:

- path
- message
- locations
- extensions
- name
- source -- gql query
- position --

Внутренние соглашения (#4)

- API является продуктом для внутренних и внешних разработчиков
- Мы стараемся сохранять обратную совместимость при внесении изменений
- Мы следим за консистентностью и следим за общим стилем в нашем API
- Мы используем camelCase для имен атрибутов
- Мы используем <action><Object> или <action><Objects> для именования мутаций выполняющих действия над объектом или объектами. Примеры: sendMessage, updateOrganization.

Склейка/Миграция (#5)

Данный раздел переехал: https://developers.doma.ai/ru/docs/api/merge

Версия структуры данных (#6)

Данный раздел переехал: https://developers.doma.ai/ru/docs/api/dv

Удаление объектов (#7)

Данный раздел переехал: https://developers.doma.ai/ru/docs/api/soft-delete

Создание и удаление объектов (#8)

API для мобильного приложения жителя (#mobile)

Полное описание сущностей и мутаций со списком всех полей можно посмотреть в playground-странице GraphiQL по адресу https://condo.d.doma.ai/admin/api. В описании ниже будут даны пояснения к некоторым параметрам.

Следующие поля сущностей и входных данных мутаций имеют единый формат и назначение.

Название	Тип	Описание	Пример значения
dv	Number	Версия структуры данных, пока для всех сущностей имеет значение 1	1
sender	JSON	{dv: Number, fingerprint: String) dv: 1, fingerprint: идентификатор клиента, который должен быть уникальный для комбинации сессии и устройства	{"dv": 1, "fingerprint": "7pvZzj2ik8Ff"}
address	String	Адрес в нормализованном формате	г Москва, ул Тверская, д 2

addressMeta	JSON	Сырой ответ DaData на указанный адрес[1]	—
unitName	Number	Номер квартиры/помещения	12

[1] Элемент ручного указания адреса рекомендуется реализовать с помощью Select, который для набранного текста выдаёт список вариантов, которые ему соответствуют. Список вариантов адресов для набранного текста получается из API DaData.

resendConfirmPhoneActionSms

Повторная отправка СМС с кодом подтверждения номера телефона.

Название	Тип	Описание
token	String	Токен, полученный в результате вызова мутации startConfirmPhoneAction
sender	String	см. “Общие параметры”
captcha	String	Значение для

registerResident

Для регистрации текущего пользователя в качестве жителя, необходима сессия авторизованного пользователя. Для регистрации или авторизации пользователя имеются следующие мутации:

- registerNewUser
- authenticateUserWithPhoneAndPassword.

Регистрация жителя, в результате которой создаётся экземпляр сущности Resident.

Если переданному адресу соответствует дом (Property) то происходит привязка жителя к данному дому, а также к УК (Property.organization), которая управляет данным домом.

Данная мутация используется на экране регистрации жителя, где пользователь указывает свой адрес и квартиру.

Смена телефона

Порядок вызова мутаций:

- startConfirmPhoneAction
- completeConfirmPhoneAction
- changePhoneNumberResidentUser

changePhoneNumberResidentUser

Меняет телефон жителя.

Название	Тип	Описание
token	String	Токен, полученный в результате вызова мутации startConfirmPhoneAction

Восстановление доступа

Явного сценария восстановления доступа нет, вместо него используется Аутентификация.

Property

checkPropertyWithAddressExist

Проверяет, существует ли в системе дом (Property), адрес которого полностью совпадает с указанным.

Ticket

allResidentTickets

Все заявки данного жителя.

createResidentTicket

Создание заявки от данного жителя.

updateResidentTicket

Обновление заявки данного жителя.

Должно быть ещё

1. Создание файла аттача createTicketFile
2. Получение классификаторов категорий и мест getAllTicketCategoryClassifiers, getAllTicketPlaceClassifiers (3 уровень скорее всего не нужен)

По добавлению файлов:

Дальше по тексту:

Ticket = Заявка,

OBS = аналог huawei obs в сберклауде

softDelete = update { deletedAt: true }

1. Каждый файл к заявке - это отдельная запись в таблице TicketFile

2. Для того чтобы добавить файл к заявке нужно создать запись в таблице TicketFile с ссылкой на Ticket (ну и организацию - там в API можно поля посмотреть)

Тут нужно подробнее описать сценарий добавления файлов на стадии создания заявки

1. Пользователь заходит на страницу создания заявки и прикрепляет файл

2. В этот момент файл ушел в OBS и создалась запись в таблицу TicketFile с заполненным полем Organization, но с пустым Ticket

3. Если пользователь выйдет с экрана добавления заявки - то добавленный файл потом когда-нибудь почистится на OBS

4. Если пользователь хочет удалить файл который он добавил к заявке то вызываем softDelete на файле (и убираем его из списка)

5. После того как он выбрал, поудалял, снова выбрал файлы и нажал сохранить заявку мы:

6. Сохраняем заявку и получаем её id

7. Пробегаем по оставшимся после действия 5 файлам и вызываем на них update, устанавливая поле Ticket

Теперь как это всё реализуется в GraphQl

У apollo есть специально обученный тип называется Upload

Когда вы отправляете данные в мутацию на создание (или обновления) TicketFile то в поле file передаете объект file из File Input

(в мобильном приложении не знаю как, скорее всего так же)

В браузере он выглядит так:

```ts
interface UploadFile<T = any> {
  uid: string;
  size?: number;
  name: string;
  fileName?: string;
  lastModified?: number;
  lastModifiedDate?: Date;
  url?: string;
  status?: UploadFileStatus;
  percent?: number;
  thumbUrl?: string;
  originFileObj?: RcFile;
  response?: T;
  error?: any;
```

```
    linkProps?: any;
    type?: string;
    xhr?: T;
    preview?: string;
}
```

В базе файл будет храниться так:

```json
{
  "id":"ckr4thdss0000w05fgzoe4tbu",
  "_meta":{ // Это ответ от OBS
    "CommonMsg":{
"Id2":"32AAAUgAIAABAAAQAAEAABAAAQAAEAABCSR3ujgFlCHZUxJdxsj7pGiVknp3WXTW",
      "Code":"",
      "HostId":"",
      "Status":200,
      "Message":"",
      "RequestId":"0000017AA9E1DE189011FC7BEFBD26E6",
      "InterfaceResult":null
    },
    "InterfaceResult":{
"Id2":"32AAAUgAIAABAAAQAAEAABAAAQAAEAABCSR3ujgFlCHZUxJdxsj7pGiVknp3WXTW",
      "Date":"Thu, 15 Jul 2021 11:16:29 GMT",
      "ETag":"\"a05f313c199d33f1e7c8b8448a1b1654\"",
      "RequestId":"0000017AA9E1DE189011FC7BEFBD26E6",
      "ContentLength":"0"


```

```
    }
  },
  "encoding":"7bit",
  "filename":"ckr4thdss0000w05fgzoe4tbu.xlsx",
  "mimetype":"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
  "originalFilename":"exportTicketsExample.xlsx"
}
```

А apollo его вернёт - так

```ts
type File = {
  __typename?: 'File';
  id?: Maybe<Scalars['ID']>;
  path?: Maybe<Scalars['String']>;
  filename?: Maybe<Scalars['String']>;
  originalFilename?: Maybe<Scalars['String']>;
  mimetype?: Maybe<Scalars['String']>;
  encoding?: Maybe<Scalars['String']>;
  publicUrl?: Maybe<Scalars['String']>;
};

```

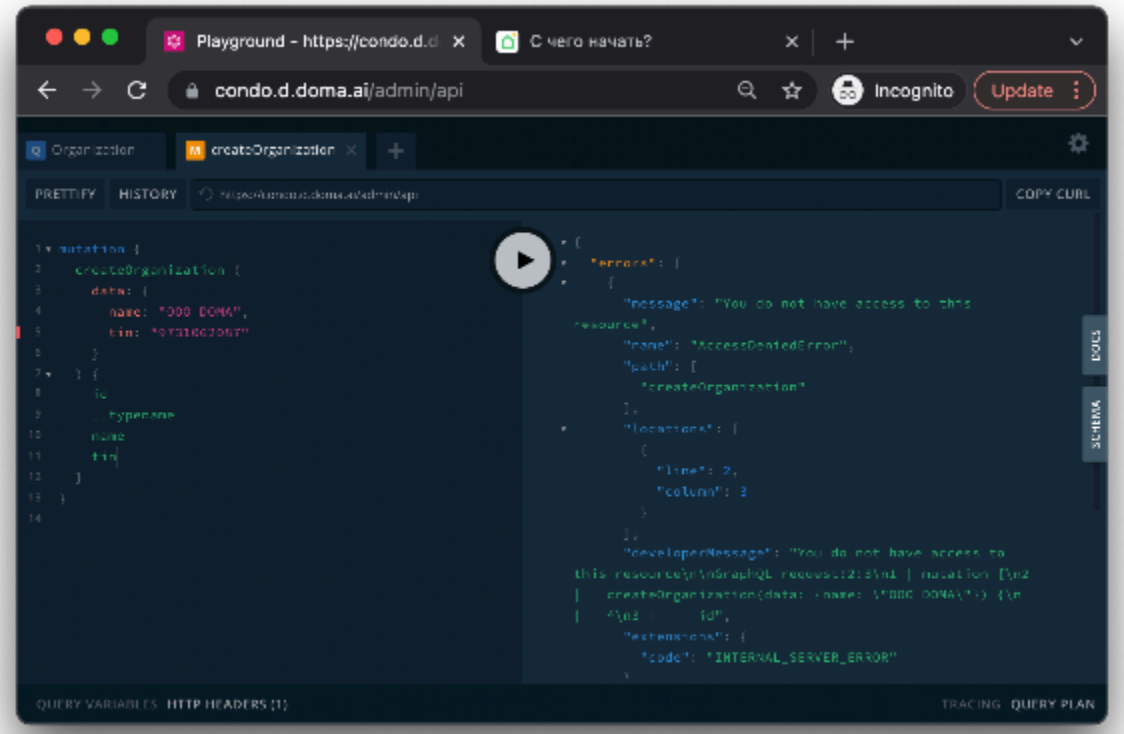
publicUrl - это ссылка на наш сервер

Когда запрос попадет на наш сервер мы:
1. Проверим права
2. Выпишем токен доступа (5 минут)
3. Перенаправим на OBS с токеном

Вот тут может возникнуть проблема с отображением фотографий (а может не возникнуть). Если возникнет, то мы вместо redirect будем делать pipe и никто не будет догадываться что мы с OBS работаем, но трафик через наш сервер вырастет.

Создание организации через createOrnanization

Например, мы хотим создать организацию (у вас должны быть соответствующие права):

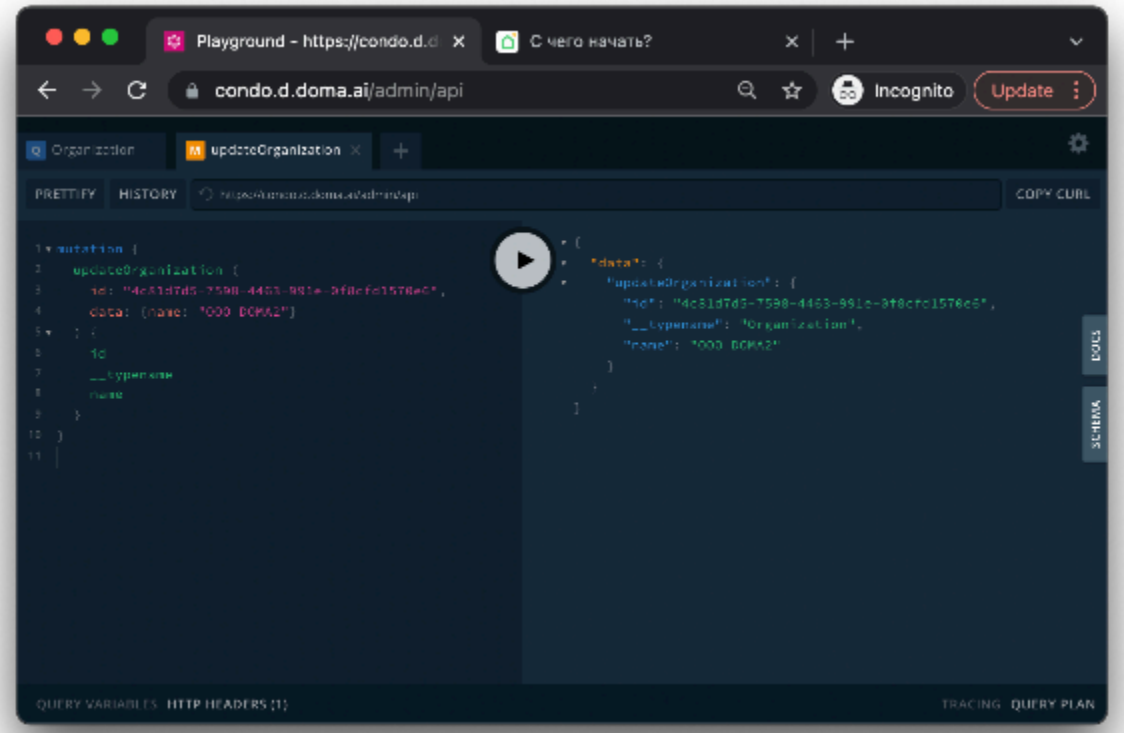


QUERY	RESULT

mutation {
createOrganization (
data: {
name: "OOO DOMA",
tin: "9731062087"
}
) {
id
__typename
name
tin
}
}
	{
	"data": {
	"createOrganization": {
	"id": "81cfe-3fcb-495f6cee",
	"__typename": "Organization",
	"name": "OOO DOMA"
	}
	}
	}

Обновление организации через updateOrganization

Например, мы хотим обновить название организации (если у вас есть права):



QUERY	RESULT

mutation {
  updateOrganization (id: "81cfe-3fcb-495f6cee", data: {name: "OOO DOMA2"}) {
  id
  __typename
  name
  }
}
	{
	  "data": {
	  "updateOrganization": {
	  "id": "81cfe-3fcb-495f6cee",
	  "__typename": "Organization",
	  "name": "OOO DOMA2"
	  }
	  }
	}

Подробнее про удаление объектов можно посмотреть в разделе #7

Подробнее про создание и обновление объектов можно посмотреть в разделе #8

Тут надо написать про fingerprint!

Работа с доменом Billing

Помимо заявок от жителей управляющая компания выставляет квитанции и собирает оплаты по ним через мобильное приложение. Список выставленных квитанций доступен в разделе https://condo.d.doma.ai/billing. В этом разделе дано описание того, как выставить квитанции для определенной управляющей компании.

Домен биллинга состоит из нескольких списков:

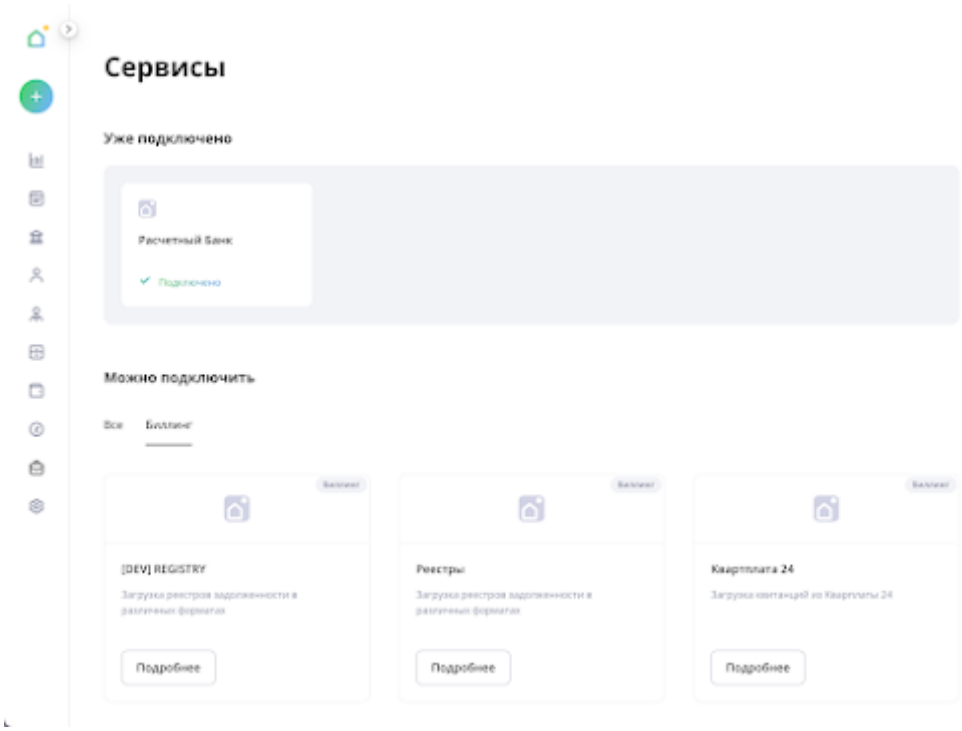
1. **BIllingIntegration** – название биллинговой интеграции. Например интеграция с “ГИС-ЖКХ”
2. **BillingIntegrationAccessRight** – служебный объект, описание прав доступа определенного пользователя к определенной интеграции (например пользователя “Интеграция с ГИС-ЖКХ” к интеграции “ГИС-ЖКХ”
3. **BillingIntegrationOrganizationContext** – служебный объект, настройки определенной интеграции для определенной компании. (например подключение организации “УК-Ромашка” к интеграции “ГИС-ЖКХ”
4. **BillingProperty** – адрес для выставления начислений (не путать с Property)
5. **BillingAccount** – лицевой счет для выставления начислений
6. **BillingReceipt** – объект квитанции

Подготовка

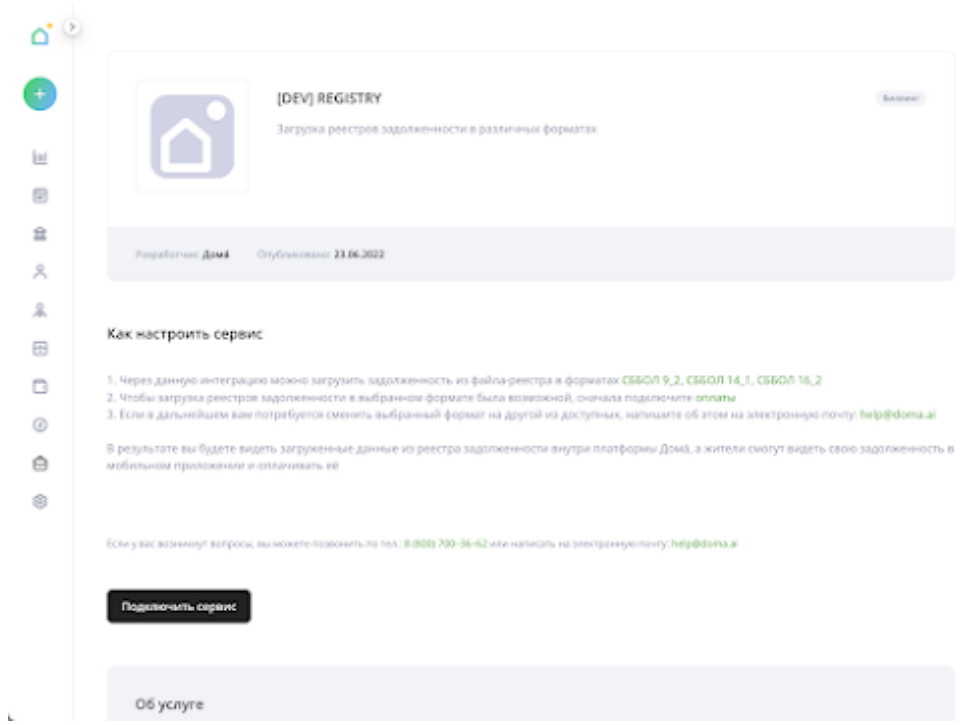
Для того, чтобы выставлять или просматривать квитанции и другие сущности домена для какой-либо УК необходимо сделать несколько вещей:

1. Зарегистрировать интеграцию в CRM (создать объект **BillingIntegration**)
2. Зарегистрировать права доступа (создать объект **BIllingIntegrationAccessRight**)
3. Подключить УК к интеграции (создать объект **BillingIntegrationOrganizationContext**)

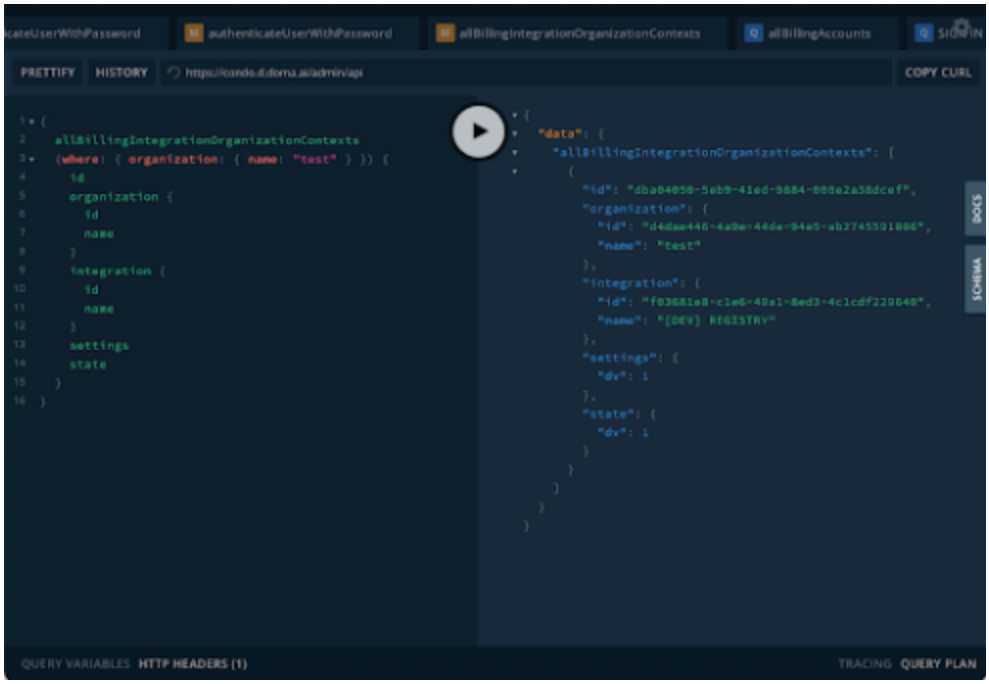
Шаги 1-2 выполняются поддержкой Doma.ai. Шаг 3 выполняется пользователем, который желает подключить свою организацию к интеграции. В интерфейсе это выглядит так:



Нажмем на “Подробнее”



Нажмем на “Подключить сервис”



Найдем созданный billingIntegrationOrganizationContext

Выставление новых квитанций (registerBillingReceipts)

Как только выполнены все подготовительные шаги можно выставлять квитанции. Для выставления квитанции используем мутацию **registerBillingReceipts**

Мутация принимает набор данных:

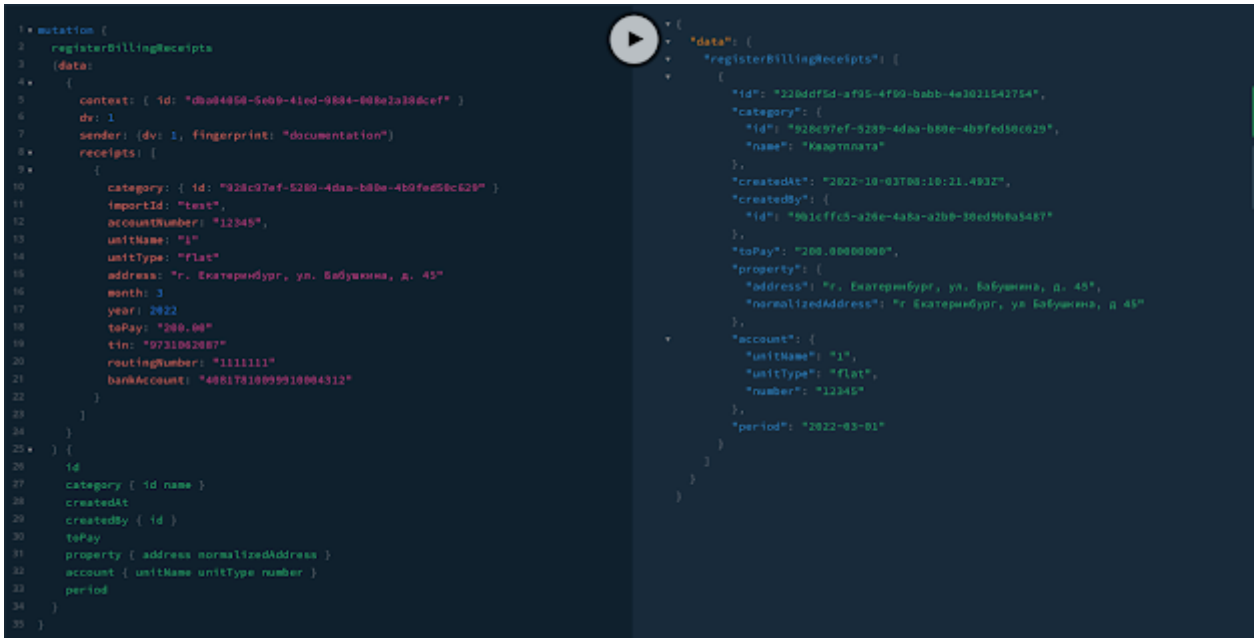
1. **context** – это ID контекста, куда необходимо выгрузить квитанции
2. **dv** – см. Версия структуры данных. Обычно равна 1
3. **sender** – см. Sender.
4. **receipts** – набор квитанций.
    a. **category** – категория. Например Квартплата или Капремонт. Посмотреть все доступные категории можно через запрос allBillingCategories
    b. **importId** – идентификатор квитанции. Обычно при разработке интеграций в это поле ставят идентификатор квитанции в исходном биллинге. Например мы делаем интеграцию ГИС-ЖКХ и Doma.ai. Некоторая квитанция имеет в ГИС-ЖКХ внутренний ID, равный “2”. Тогда для этой квитанции importId будет равен “2”
    c. **accountNumber** – Номер лицевого счета
    d. **unitName** – Номер квартиры
    e. **unitType** – Тип помещения
    f. **address** – Ненормализованный адрес, без квартиры.
    g. **normalizedAddress** (Необязательное) Кастомная нормализация адреса. Пригодится, если, ваши адреса не распознаются сервисом DaData.
    h. **month** – Месяц
    i. **year** – Год
    j. **toPay** – Сколько необходимо заплатить за квитанцию. Указывайте строкой, с двумя символами после точки.
    k. **toPayDetails** – Детализация квитанции. см. тип **toPayDetails**
    l. **services** – Детализация квитанции по услугам. см. тип **services**
    m. **tin** – ИНН получателя платежа
    n. **routingNumber** – БИК получателя платежа
    o. **bankAccount** – Номер счета получателя платежа

Результат мутации

Результатом выполнения мутации является набор созданных квитанций (BillingReceipt) Если есть мобильное приложение жителя, то выставленные квитанции мгновенно начнут в нем отображаться

Ограничения мутации:

- Мутация позволяет создать не более чем 50 объектов за раз.
- Мутация не позволит создать объекты в “чужую” организацию.
- Мутация не позволяет создавать множество одинаковых квитанций.



Получение списка адресов (allBIllingProperties)

Получить созданные адреса можно с помощью запроса **allBillingProperties**

Получение списка лицевых счетов (allBillingAccounts)

Получить созданные ЛС можно с помощью запроса **allBillingAccounts**

Получение списка адресов (allBillingReceipts)

Получить созданные квитанции можно с помощью запроса **allBillingReceipts**