1. ФИНАЛЬНЫЙ ЭТАП

Задача командного тура

1.1. Легенда

Стремительно развиваясь, деревня Финтехово переросла в полноценный город. В деревне начал развиваться бизнес, поэтому здесь решили построить торговый центр (далее — ТЦ).

У владельца ТЦ есть множество площадей, которые предприниматели арендуют под магазины, кафе и т.д. При этом каждый расчётный период (например, месяц) арендаторы платят за занимаемое ими место.

Однако часто происходит следующая ситуация: арендатор задерживает платеж за арендованное помещение, но продолжает им пользоваться. Несмотря на его обещания погасить долг, владелец ТЦ не получает своих денег. В результате стороны договора не могут урегулировать спор и обращаются в суд.

В связи с тем, что люди стали чаще покупать товары в онлайн-сервисах, количество покупателей в ТЦ резко сократилось. В результате этого владельцы ТЦ понесли дополнительные убытки.

Часто блокчейн-технологии применяются там, где важно, чтобы все стороны какого-либо договора гарантированно исполнили свои финансовые обязательства, и это тот самый случай.

В финальной задаче сезона 2021/22 годов вам предстоит разработать децентрализованное приложение для контроля доходов арендодателя.

Как это будет работать?

- Арендодатель устанавливает условия для каждой арендной площади, управляет договорами с арендаторами.
- Каждый арендатор регистрируется в системе и привязывает свой криптокошелек. Он заключает договор с арендодателем и обязуется принимать оплату от покупателей в криптовалюте (*для простоты представим, что криптовалюта Ethereum используется повсеместно*).
- Каждый кассир, работающий на точке арендатора, также имеет доступ к системе. Он оформляет покупки и предоставляет покупателю *QR-код* для оплаты.
- Все вырученные средства поступают на счёт контракта арендодателя (*владельца ТЦ*). Каждый месяц часть этих средств идёт на оплату аренды, а остаток переводится на счет арендатора.
- Ни арендатор, ни арендодатель не могут самостоятельно изменить условия аренды: все действия происходят после согласия обеих сторон.

1.2. Набор заданий

Решение командной задачи разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулирует необходимый функционал, который должен быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволяющих проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагается решать в отдельный соревновательный день (*итерацию*).

На каждой итерации решение подзадач проверяется с помощью системы автоматического тестирования, которая запускает решение участников с теми или иными параметрами в соответствии с приемочными тестами.

1.3. Условия проведения

- Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.
- Участники не ограничены в выборе языков программирования, если в задаче (подзадаче) не указано таких требований.
- Участники не могут пользоваться помощью тренера, наставника, сопровождающего лица или привлекать третьих лиц для решения задачи.
- Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются, баллы. Баллы за определенные подзадачи можно получить только в день, закрепленный за конкретным этапом.
- В начале первого дня состязаний участники каждой команды получают доступ к репозиторию на серверах gitlab.com. Каждая команда имеет свой собствен-

ный репозиторий. Члены других команд не имеют доступ к чужим репозиториям.

- В течение дня не ведется учет количества изменений, которые команды регистрируют в Git-репозитории.
- В конце каждого дня финального этапа жюри проверяет решение участников на соответствие приемочным тестам для каждой подзадачи, входящей в набор для соответствующего этапа.
- Баллы за все подзадачи, для которых прошло приемочное тестирование, определяют баллы, набранные командой в данный день соревнований. Для некоторых подзадач, на усмотрение жюри, может быть возможность получать баллы с дисконтом в последующие дни соревнований. Также есть подзадачи, которые позволяют набирать дополнительные баллы, если приемочные тесты для этих подзадач проходят в последующие дни (регрессионное тестирование).
- После завершения сорревновательного дня команды могут ознакомиться с логикой проверки решений и подать апелляцию не позже начала следующего сорревновательного дня, если не согласны с корректностью проведения тестов.
- После рассмотрения сути апелляции, жюри вправе провести тестирование еще раз и назначить команде баллы за соответствующие подзадачи.
- Описанные выше условия могут быть изменены членами жюри. Все изменения в условиях объявляются участникам перед началом каждого дня состязаний.

1.4. Процедура проведения приемочного тестирования и критерии оценки

Для каждого дня соревнований (для каждой итерации) справедлива следующая процедура приемочного тестирования:

- В конце каждого дня финального этапа команды должны сформировать запрос на слияние (*Merge Request*) из своей ветки исходного кода в основную ветку (*master*) в Git-репозитории.
- Команда ответственна за то, чтобы в запросе на слияние не было конфликтов. Запрос на слияние с конфликтами может не рассматриваться жюри для выполнения приемочного тестирования.
- После того, как все команды отправили запросы на слияние, жюри одобряет все запросы и приступает к приемочному тестированию, для тех подзадач, которые входят в соответствующую итерацию. Для этого исходный код приложения команды загружается в систему автоматического тестирования (поддержку которой осуществляет функциональность GitLab CI/CD), где запускаются автоматические тесты на соответствие решения участников требованиям к приемочным тестам (*acceptance criteria*).
- Если все приемочные тесты для данной подзадачи пройдены успешно, команда получает баллы за данную подзадачу. Если хотя бы один тест не проходит, то баллы за данное подзадачу не начисляются.

Приемочные тесты для каждой подзадачи описаны в разделе “Подробное описание подзадач”.

Дальше перечислены баллы, которые получает команда за решение подзадач в

каждой итерации.

Максимальное количество баллов, которое может набрать команда за решение всех подзадач — 400.

Первая итерация

user story	баллы
US-001 Развертывание контракта	1
US-002 Аренда помещения	12
US-101 Сборка и запуск решения	5
US-102 Аутентификация через MetaMask на стороне сервера	30
US-201 Аутентификация через MetaMask на стороне клиента	15

Максимальное количество баллов за итерацию по части смарт-контракта — 13.
Максимальное количество баллов за итерацию по серверной части приложения — 35.
Максимальное количество баллов за итерацию по пользовательскому интерфейсу приложения — 15.
Максимальное количество баллов за всю итерацию — 63.

Вторая итерация

user story	баллы
US-003 Управление списком кассиров	25
US-004 Приём платежей	21
US-103 Добавление помещений	10
US-104 Задание адреса контракта для помещения	12
US-105 Редактирование помещений	6
US-106 Удаление помещений	8
US-107 Просмотр списка помещений арендодателем	8
US-108 Просмотр списка арендованных и доступных для аренды помещений	12
US-109 Установка публичного названия арендуемого помещения	11
US-202 Добавление помещений	20
US-203 Просмотр помещений	15
US-204 Редактирование помещений	10
US-205 Установка публичного названия арендуемого помещения	9
US-206 Разрешение аренды помещения	20
US-207 Удаление помещений	6

Максимальное количество баллов за итерацию по части смарт-контракта — 46.
Максимальное количество баллов за итерацию по серверной части приложения — 67.
Максимальное количество баллов за итерацию по пользовательскому интерфейсу приложения — 80.
Максимальное количество баллов за всю итерацию — 193.

Третья итерация

user story	баллы
US-005 Вывод средств арендатора	21
US-006 Вывод средств арендодателя	17
US-007 Удаление контракта	36
US-110 Регистрация квитанции на оплату	25
US-111 Получение квитанции на оплату	5
US-208 Управление кассирами	20
US-209 Оплата квитанций	20

Максимальное количество баллов за итерацию по части смарт-контракта — 74.
Максимальное количество баллов за итерацию по серверной части приложения — 30.
Максимальное количество баллов за итерацию по пользовательскому интерфейсу приложения — 40.
Максимальное количество баллов за всю итерацию — 144.

Критерии определения команды-победителя командного тура

- Сумма баллов, набранных за решения подзадач командного тура финального этапа, определяет итоговую результативность команды (измеряемую в баллах).
- Команды ранжируются по результативности.
- Команда победитель определяется, как команда с максимальной результативностью.

1.5. Подробное описание подзадач

Участникам необходимо будет реализовать систему из трёх взаимосвязанных компонентов:

- смарт-контракта соглашения об аренде,
- серверной части,
- пользовательского интерфейса.

Контракт соглашения об аренде: высокоуровневое описание

Главная задача разрабатываемого нами смарт-контракта — сокращение количества судебных разбирательств между арендатором и арендодателем. Для предотвращения подобных разбирательств необходимо подготовить вещественные доказательства, например, историю транзакций, отображающую заработок арендатора. История транзакций должна быть проведена по заранее прописанным в смарт-контракте правилам и подтверждена криптографическими цифровыми подписями.

Смарт-контракт предоставляет следующие возможности его пользователям:

- интерфейс оплаты для клиентов арендатора,
- учёт доходов арендатора,
- автоматический вычет арендной платы.

Каждый смарт-контракт существует только на время аренды одного помещения одним арендатором. Когда срок аренды истекает, контракт должен быть удалён из сети блокчейн. При необходимости новый контракт может быть развёрнут в сети для того же помещения, а затем арендован тем же арендатором или другим человеком.

Таким образом жизненный цикл смарт-контракта состоит из трёх стадий:

- Создание свободного контракта, ожидающего начало аренды. Одновременно с этим создается юридический документ устанавливающий силу контракта с этим адресом на реальную арендную площадь.
- Помещение арендовано. Одновременно с этим арендатор подписывает упрощенный юридический документ, делегирующий всю работу на контракт.
- Аренда заканчивается по одной из трёх причин: преждевременный разрыв контракта, задолженность арендатора или штатное истечение срока.

(*) На протяжении жизни контракта у него может быть только один арендатор с момента начала аренды.

Серверная часть приложения: высокоуровневое описание

Задача серверной части — взять на себя работу с той частью данных, которая не может храниться на стороне смарт-контракта по экономическим и техническим соображениям.

Серверная часть приложения предоставляет следующие возможности её пользователям:

- аутентификация с помощью аккаунта в сети блокчейн,
- манипуляция со списком сдаваемых в аренду помещений для арендодателя,
- манипуляция арендованными помещениями для арендатора,
- выписывание квитанций для оплаты товаров или услуг арендатора для кассиров,
- получение квитанции из базы данных для клиентов арендатора.

Роли пользователей определяются следующим образом:

- Адрес аккаунта **арендодателя** передаётся приложению при запуске.
- Если в базе данных существует комната, арендованная пользователем, то он считается **арендатором**.
- Если в базе данных существует арендованная комната, в списке кассиров которой содержится адрес в сети блокчейн аккаунта пользователя, то этот пользователь — кассир.
- Аутентифицированный пользователь может не иметь роли, например, если он является потенциальным арендатором.
- Часть функций, например, получение квитанций, доступно и неаутентифицированным пользователям.

Пользовательский интерфейс приложения: высокоуровневое описание

Задача пользовательского интерфейса приложения — предоставить арендодателю и арендаторам удобный интерфейс для управления помещениями как на сервере, так и в смарт-контракте.

Первая итерация

Контракт соглашения об аренде: первая итерация

Цель итерации: реализация первого пункта жизненного цикла контракта.

Решение должно быть предоставлено в виде исходного кода на языке `Solidity`, компилируемым solc v0.8.11.

Контракт должен поддерживать следующий интерфейс:

- метод `nonpayable constructor(uint roomInternalId)` регистрирует контракт в сети блокчейн.
- метод `payable rent(uint deadline, address tenant, uint rentalRate, uint billingPeriodDuration, uint billingsCount, Sign landlordSign)` переводит контракт в арендованное состояние.
- метод `view getRoomInternalId() returns (uint)` возвращает id помещения, к которому привязан контракт.
- метод `view getLandlord() returns (address)` возвращает адрес владельца контракта.
- метод `view getTenant() returns (address)` возвращает адрес арендатора.
- метод `view getBillingPeriodDuration() returns (uint)` возвращает продолжительность расчётного периода.
- метод `view getRentStartTime() returns (uint)` возвращает временную метку начала аренды.
- метод `view getRentEndTime() returns (uint)` возвращает временную метку окончания аренды.
- метод `view getRentalRate() returns (uint)` возвращает арендную ставку.

Где Sign — следующая структура данных:

```
struct Sign
{
    uint8 v;
    bytes32 r;
    bytes32 s;
}
```

А параметр `landlordSign` сформирован из следующей EIP-712 сигнатуры:

`RentalPermit(uint256 deadline,address tenant,uint256 rentalRate,uint256 billingPeriodDuration,uint256 billingsCount)`

под доменом

`EIP712Domain(string name,string version,address verifyingContract)`

где

`name` = `"Rental Agreement"`

`version` = `"1.0"`

(*) Коды ошибок и примеры поведения методов в разных контекстах приведены в секции приемочного тестирования этой итерации.

Контракт соглашения об аренде: приемочное тестирование первой итерации

US-001 Развертывание контракта (1 балл)

Описание: Я, как арендодатель, могу развернуть смарт-контракт в сети блокчейн, указав ID помещения, арендную плату и период аренды.

Критерий оценивания (AC-001-01) : Развёртывание контракта (1 балл)

---

1. В директории `blockchain/contracts/` содержится контракт с именем `RentalAgreement`.
2. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
3. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `getRoomInternalId()`. Метод возвращает значение 2419200. Транзакция в блокчейн не отправляется.
4. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `getLandlord()`. Метод возвращает адрес `<landlord-address>`. Транзакция в блокчейн не отправляется.

US-002 Аренда помещения (12 баллов)

Зависит от успешного прохождения US-001

Описание: Я, как арендатор, могу арендовать помещение, имея подпись арендодателя и достаточное количество средств на счету для оплаты первого расчетного периода.

Критерий оценивания (AC-002-01) : Акт заключения сделки (1 балл)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 5, ...)` и передаёт

100 wei. Транзакция исполняется успешно. Block timestamp: 300.

3. Аккаунт с адресом <landlord-address> вызывает у контракта метод getTenant(). Метод возвращает адрес <tenant-address>. Транзакция в блокчейн не отправляется.
4. Аккаунт с адресом <landlord-address> вызывает у контракта метод getRentalRate(). Метод возвращает значение 100. Транзакция в блокчейн не отправляется.
5. Аккаунт с адресом <landlord-address> вызывает у контракта метод getBillingPeriodDuration(). Метод возвращает значение 1000. Транзакция в блокчейн не отправляется.
6. Аккаунт с адресом <landlord-address> вызывает у контракта метод getRentStartTime(). Метод возвращает значение 300. Транзакция в блокчейн не отправляется.
7. Аккаунт с адресом <landlord-address> вызывает у контракта метод getRentEndTime(). Метод возвращает значение 5300. Транзакция в блокчейн не отправляется.
8. Аккаунт с адресом <tenant-address> вызывает у контракта метод rent() с параметрами (410, '<tenant-address>', 100, 1000, 5, ...) и передаёт 100 wei. В ходе вызова метода контракта выполняется команда revert("The contract is being in not allowed state"). Транзакция в блокчейн не отправляется. Block timestamp: 400.

Критерий оценивания (AC-002-02) : Валидация параметров акта (11 баллов)

Зависит от успешного прохождения AC-002-01

1. Аккаунт с адресом <landlord-address> вызывает у контракта метод constructor с параметром (2419200). Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом <tenant-address> вызывает у контракта метод rent() с параметрами (1642676872, '<tenant-address>', 100, 1000, 5, A), где A - сообщение RentalPermit, подписанное приватным ключом аккаунта <address-of-somebody>. Транзакция снабжается суммой в 100 wei. В ходе вызова метода контракта выполняется команда revert("Invalid landlord sign"). Транзакция в блокчейн не отправляется.
3. Аккаунт с адресом <tenant-address> вызывает у контракта метод rent() с параметрами (100000, '<tenant-address>', 100, 1000, 5, A), где A - сообщение RentalPermit, подписанное приватным ключом аккаунта <landlord-address>. Транзакция снабжается суммой в 100 wei. В ходе вызова метода контракта выполняется команда revert("The operation is outdated"). Транзакция в блокчейн не отправляется.
4. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод rent() с параметрами (1642676872, '<tenant-address>', 100, 1000, 5, A), где A - сообщение RentalPermit, подписанное приватным ключом аккаунта <landlord-address>. Транзакция снабжается суммой в 100 wei. В ходе вызова метода контракта выполняется команда revert("The caller account and the account specified as a tenant do not match"). Транзакция в блокчейн

не отправляется.

5. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<landlord-address>', 100, 1000, 5, A)`, где `A` - сообщение `RentalPermit`, подписанное приватным ключом аккаунта `<landlord-address>`. Транзакция снабжается суммой в `100` wei. В ходе вызова метода контракта выполняется команда `revert("The landlord cannot become a tenant")`. Транзакция в блокчейн не отправляется.
6. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 0, 1000, 5, A)`, где `A` - сообщение `RentalPermit`, подписанное приватным ключом аккаунта `<landlord-address>`. В ходе вызова метода контракта выполняется команда `revert("Rent amount should be strictly greater than zero")`. Транзакция в блокчейн не отправляется.
7. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 100, 0, 5, A)`, где `A` - сообщение `RentalPermit`, подписанное приватным ключом аккаунта `<landlord-address>`. Транзакция снабжается суммой в `100` wei. В ходе вызова метода контракта выполняется команда `revert("Rent period should be strictly greater than zero")`. Транзакция в блокчейн не отправляется.
8. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 100, 1000, 0, A)`, где `A` - сообщение `RentalPermit`, подписанное приватным ключом аккаунта `<landlord-address>`. Транзакция снабжается суммой в `100` wei. В ходе вызова метода контракта выполняется команда `revert("Rent period repeats should be strictly greater than zero")`. Транзакция в блокчейн не отправляется.
9. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 100, 1000, 5, A)`, где `A` - сообщение `RentalPermit`, подписанное приватным ключом аккаунта `<landlord-address>`. Транзакция снабжается суммой в `99` wei. В ходе вызова метода контракта выполняется команда `revert("Incorrect deposit")`. Транзакция в блокчейн не отправляется.

Серверная часть приложения: первая итерация

Цель итерации: реализация первого пункта из списка функционала.

Серверная часть должна поддерживать запросы согласно следующей схеме `GraphQL`:

```
type Query {
    authentication: Authentication
}

type Mutation {
    requestAuthentication(address: String!): String!
    authenticate(
        address: String!
        signedMessage: InputSignature!
    ): Authentication!
}
```

```
type Authentication {
    address: String!
    isLandlord: Boolean!
}

input InputSignature {
    v: String!
    r: String!
    s: String!
}
```

Серверная часть приложения: приемочное тестирование первой итерации

US-101 Сборка и запуск решения (5 баллов)

Описание: Я, как системный администратор, могу собрать и запустить приложение с помощью `docker-compose`.

Критерий оценивания (AC-101-01) : Сборка и запуск приложения через docker-compose (3 балла)

---

1. Файл `docker-compose.yaml` написан участниками и находится в корне директории с решением. Решение участников принимает на вход переменные окружения `$LANDLORD_ADDRESS` в качестве адреса аккаунта арендодателя и `$RPC_URL` в качестве URL-адреса узла RPC блокчейна.
   Команда `docker-compose up --detach --build --force-recreate` в директории с исходным кодом исполняется успешно.
2. Команда `docker-compose ps | grep solution-web` выводит строку с информацией о собранном контейнере.

Критерий оценивания (AC-101-02) : Проверка доступности сервера (2 балла)

---

1. GET-запрос к серверу по адресу `http://solution-web/graphql` выполняется успешно (код ответа 200).

US-102 Аутентификация через MetaMask на стороне сервера (30 баллов)

Зависит от успешного прохождения US-101

Описание: Я, как пользователь приложения, могу аутентифицироваться в нём, используя свой кошелёк MetaMask.

Критерий оценивания (AC-102-01) : Генерация сообщений для аутентификации (5 баллов)

---

1. Здесь и далее запросы к серверу отправляются по адресу `http://solution-web/graphql`. На сервер отправляется следующий запрос:

```
mutation {
    message: requestAuthentication(
        address: "<address-0>"
    )
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "message": "<message-0>"
    }
}
```

2. На сервер отправляется следующий запрос:

```
mutation {
    message: requestAuthentication(
        address: "<address-0>"
    )
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "message": "<message-1>"
    }
}
```

3. На сервер отправляется следующий запрос:

```
mutation {
    message: requestAuthentication(
        address: "<address-1>"
    )
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "message": "<message-2>"
    }
}
```

4. `<message-0>`, `<message-1>` и `<message-2>` попарно отличаются друг от друга.

Критерий оценивания (AC-102-02) : Успешная аутентификация (10 баллов)

---

1. На сервер отправляется следующий запрос:

   ```
   query {
       authentication { address, isLandlord }
   }
   ```

   Запрос успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "authentication": null
       }
   }
   ```

2. На сервер отправляется следующий запрос:

   ```
   mutation {
       message: requestAuthentication(
           address: "<address-0>"
       )
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "message": "<message-0>"
       }
   }
   ```

3. Владелец аккаунта `<address-0>` подписывает сообщение `<message-0>` своим приватным ключом и получает объект подписи `<signature-0>`, состоящий из трёх компонентов: `<v-0>`, `<r-0>` и `<s-0>`, здесь и далее представляющие из себя числа в шестнадцатеричной системе, начинающиеся с `0x`.
4. На сервер отправляется следующий запрос:

   ```
   mutation {
       authentication: authenticate(
           address: "<address-0>"
           signedMessage: {
               v: "<v-0>",
               r: "<r-0>",
               s: "<s-0>"
           }
       ) {
           address
           isLandlord
       }
   }
   ```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "authentication": {
            "address": "<address-0>",
            "isLandlord": false
        }
    }
}
```

Переданные в ответе куки могут использоваться при совершении действий от имени владельца аккаунта `<address-0>`.

5. На сервер отправляется следующий запрос:

```
query {
    authentication { address, isLandlord }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "authentication": {
            "address": "<address-0>",
            "isLandlord": false
        }
    }
}
```

6. На сервер отправляется следующий запрос:

```
mutation {
    message: requestAuthentication(
        address: "<landlord-address>"
    )
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "message": "<message-1>"
    }
}
```

7. Арендодатель с аккаунта `<landlord-address>` подписывает сообщение `<message-1>` своим приватным ключом и получает объект подписи `<signature-1>`, состоящий из трёх компонентов: `<v-1>`, `<r-1>` и `<s-1>`.
8. На сервер отправляется следующий запрос:

```
mutation {
    authentication: authenticate(
        address: "<landlord-address>"
        signedMessage: {
```

```
            v: "<v-1>",
            r: "<r-1>",
            s: "<s-1>"
        }
    ) {
        address
        isLandlord
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "authentication": {
            "address": "<landlord-address>",
            "isLandlord": true
        }
    }
}
```

Переданные в ответе куки могут использоваться при совершении действий от имени владельца аккаунта `<landlord-address>`.

9. На сервер отправляется следующий запрос:

```
query {
    authentication { address, isLandlord }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "authentication": {
            "address": "<landlord-address>",
            "isLandlord": true
        }
    }
}
```

Критерий оценивания (AC-102-03) : Неудачная аутентификация (15 баллов)

---

1. На сервер отправляется следующий запрос:

```
mutation {
    authentication: authenticate(
        address: "<landlord-address>"
        signedMessage: {
            v: "<v-1>",
            r: "<r-1>",
            s: "<s-1>"
```

```
        }
    ) {
        address
        isLandlord
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "errors": [
        {
            "message": "Authentication failed"
        }
    ]
}
```

2. На сервер отправляется следующий запрос:

```
mutation {
    message: requestAuthentication(
        address: "<address-0>"
    )
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "message": "<message-2>"
    }
}
```

3. На сервер отправляется следующий запрос:

```
mutation {
    authentication: authenticate(
        address: "<address-0>"
        signedMessage: {
            v: "<invalid-v>",
            r: "<invalid-r>",
            s: "<invalid-s>",
         }
    ) {
        address
        isLandlord
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "errors": [
        {
            "message": "Authentication failed"
```

```
        }
    ]
}
```

4. Пользователь с аккаунта `<address-1>` подписывает сообщение `<message-2>` своим приватным ключом и получает объект подписи `<signature-2>`, состоящий из трёх компонентов: `<v-2>`, `<r-2>` и `<s-2>`. На сервер отправляется следующий запрос:

```
mutation {
    authentication: authenticate(
        address: "<address-0>"
        signedMessage: {
            v: "<v-2>",
            r: "<r-2>",
            s: "<s-2>"
        }
    ) {
        address
        isLandlord
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "errors": [
        {
            "message": "Authentication failed"
        }
    ]
}
```

Пользовательский интерфейс приложения: первая итерация

Цель итерации: реализация аутентификации.

Пользовательский интерфейс приложения: приемочное тестирование первой итерации

US-201 Аутентификация через MetaMask на стороне клиента (15 баллов)

Описание: Я, как пользователь приложения, могу аутентифицироваться в нём, используя свой кошелёк MetaMask.

Критерий оценивания (AC-201-01) : Аутентификация (11 баллов)

---

1. В браузере открывается страница по адресу `http://solution-web/`.

2. На странице нажимается кнопка `.authentication__authenticate`.
3. MetaMask показывает всплывающее окно, в котором просит пользователя выбрать аккаунт для подключения. Пользователь выбирает аккаунт с адресом `<address-0>` и предоставляет необходимые разрешения.
4. MetaMask показывает новое всплывающее окно, в котором просит пользователя подтвердить подписание сообщения. Сообщение отображается в человекочитаемом виде. Пользователь подтверждает подписание сообщения, нажав кнопку *Sign*.
5. Кнопка `.authentication__authenticate` исчезает со страницы, и появляется новый элемент `.account__address` с текстом `<address-0>`. Теперь пользователь может совершать действия от имени владельца аккаунта `<address-0>` при запросах к серверу.

Критерий оценивания (AC-201-02) : Проверка актуальности аутентификации (4 балла)

---

1. В браузере открывается страница по адресу `http://solution-web/`.
2. Пользователь проходит аутентификацию, используя аккаунт с адресом `<address-0>`.
3. Пользователь закрывает страницу, меняет активный аккаунт в MetaMask на `<address-1>`, самостоятельно подключает его к сайту, и снова открывает `http://solution-web/`.
4. На странице появляется кнопка `.authentication__authenticate`, а также элемент `.authentication__warning` с текстом `Your MetaMask account is different from the one you authenticated with before`.

Вторая итерация

Контракт соглашения об аренде: вторая итерация

Цель итерации: реализация второго пункта жизненного цикла контракта.

Контракт должен поддерживать следующий интерфейс:

- метод `nonpayable addCashier(address addr)` добавляет аккаунт в список кассиров.
- метод `nonpayable removeCashier(address cashierAddr)` удаляет аккаунт из списка кассиров.
- метод `view getCashierNonce(address cashierAddr) returns (uint)` возвращает число, используемое кассиром для подписи.
- метод `view getCashiersList() returns (address[])` возвращает список кассиров.
- метод `payable pay(uint deadline, uint nonce, uint value, Sign cashierSign)` используется для приема контрактом внешних платежей.

(*) Включая интерфейс предыдущей итерации.

Где параметр `cashierSign`, сформирован из следующей EIP-712 сигнатуры:

`Ticket(uint256 deadline,uint256 nonce,uint256 value)`

под доменом

`EIP712Domain(string name,string version,address verifyingContract)`

где

`name` = `"Rental Agreement"`

`version` = `"1.0"`

(*) Коды ошибок и примеры процедуры проведения платежей приведены в секции приемочного тестирования этой итерации.

Контракт соглашения об аренде: приемочное тестирование второй итерации

US-003 Управление списком кассиров (25 баллов)

Зависит от успешного прохождения US-002

Описание: Я, как арендатор, могу манипулировать списком кассиров арендованного мною помещения.

Критерий оценивания (AC-003-01) : Добавление кассиров (5 баллов)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 27152431086164306, 1234567, 5, ...)` и передаёт 27152431086164306 wei. Транзакция исполняется успешно.
3. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `addCashier()` с параметром `('<address-of-somebody>')`. В ходе вызова метода контракта выполняется команда `revert("You are not a tenant")`.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<landlord-address>')`. В ходе вызова метода контракта выполняется команда `revert("The landlord cannot become a cashier")`.
5. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('0x0')`. В ходе вызова метода контракта выполняется команда `revert("Zero address cannot become a cashier")`.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
7. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
8. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.

9. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.
10. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
11. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-2-address>')`. Транзакция исполняется успешно.
12. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.
13. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.
14. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.

Критерий оценивания (AC-003-02) : Удаление кассиров (3 балла)

Зависит от успешного прохождения AC-003-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 27152431086164306, 1234567, 5, ...)` и передаёт `27152431086164306` wei. Транзакция исполняется успешно.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-2-address>')`. Транзакция исполняется успешно.
5. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-3-address>')`. Транзакция исполняется успешно.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `removeCashier()` с параметром `('<cashier-1-address>')`. В ходе вызова метода контракта выполняется команда `revert("You are not a tenant")`.
7. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `removeCashier()` с параметром `('<address-of-somebody>')`. В ходе вызова метода контракта выполняется команда `revert("Unknown cashier")`.
8. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.
9. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.
10. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод

`getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.

11. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `removeCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
12. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
13. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 9. Транзакция в блокчейн не отправляется.
14. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 10. Транзакция в блокчейн не отправляется.
15. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `removeCashier()` с параметром `('<cashier-2-address>')`. Транзакция исполняется успешно.
16. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
17. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
18. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 10. Транзакция в блокчейн не отправляется.
19. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `removeCashier()` с параметром `('<cashier-1-address>')`. В ходе вызова метода контракта выполняется команда `revert("Unknown cashier")`.
20. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
21. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение, отличное от такового в пункте 8. Транзакция в блокчейн не отправляется.
22. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
23. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 10. Транзакция в блокчейн не отправляется.
24. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-4-address>')`. Транзакция исполняется успешно.

25. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 21. Транзакция в блокчейн не отправляется.
26. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-2-address>')`. Метод возвращает значение `0`. Транзакция в блокчейн не отправляется.
27. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-3-address>')`. Метод возвращает значение, совпадающее с таковым в пункте 10. Транзакция в блокчейн не отправляется.
28. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-4-address>')`. Метод возвращает значение отличное от нуля. Транзакция в блокчейн не отправляется.

Критерий оценивания (AC-003-03) : Получение списка кассиров (7 баллов)

Зависит от успешного прохождения AC-003-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 27152431086164306, 1234567, 5, ...)` и передаёт `27152431086164306` wei. Транзакция исполняется успешно.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getCashiersList()`. Метод возвращает список ['<cashier-1-address>']. Транзакция в блокчейн не отправляется.

Критерий оценивания (AC-003-04) : Получение списка кассиров после удаления (10 баллов)

Зависит от успешного прохождения AC-003-02

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(1642676872, '<tenant-address>', 27152431086164306, 1234567, 5, ...)` и передаёт `27152431086164306` wei. Транзакция исполняется успешно.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `removeCashier()` с параметром `('<cashier-1-address>')`. Транзакция испол-

няется успешно.

5. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getCashiersList()`. Метод возвращает пустой список. Транзакция в блокчейн не отправляется.

US-004 Приём платежей (21 баллов)

Зависит от успешного прохождения AC-003-01

Описание: Я, как клиент арендатора, могу расплатиться с ним через интерфейс контракта аренды.

Критерий оценивания (AC-004-01) : Приём платежа (5 баллов)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 5, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение N, отличное от нуля. Транзакция в блокчейн не отправляется.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(400, N, 120, ...)` и передаёт `120` wei. В ходе исполнения транзакции испускается событие `PurchasePayment(120)`. Транзакция исполняется успешно. Block timestamp: 390.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение `M`, отличное от `N` и нуля. Транзакция в блокчейн не отправляется.
7. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(1410, M, 30, ...)` и передаёт `30` wei. В ходе исполнения транзакции испускается событие `PurchasePayment(30)`. Транзакция исполняется успешно. Block timestamp: 1400.

Критерий оценивания (AC-004-02) : Валидация данных о платеже (2 балла)

Зависит от успешного прохождения AC-004-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 5, ...)` и передаёт `100`

wei. Транзакция исполняется успешно. Block timestamp: 300.

3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `getCashierNonce()` с параметром `('<cashier-1-address>')`. Метод возвращает значение `N`, отличное от нуля. Транзакция в блокчейн не отправляется.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(410, N, 100, [0,0,0])` и передаёт `100` wei. В ходе вызова метода контракта выполняется команда `revert("Unknown cashier")`. Block timestamp: 400.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(390, N, 100, ...)` и передаёт `100` wei. В ходе вызова метода контракта выполняется команда `revert("The operation is outdated")`. Block timestamp: 400.
7. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(410, N+1, 100, ...)` и передаёт `100` wei. В ходе вызова метода контракта выполняется команда `revert("Invalid nonce")`. Block timestamp: 400.
8. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(410, N, 100, ...)` и передаёт `1013` wei. В ходе вызова метода контракта выполняется команда `revert("Invalid value")`. Block timestamp: 400.

Критерий оценивания (AC-004-03) : Отклонение платежей по истечении срока действия контракта (1 балл)

Зависит от успешного прохождения AC-004-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 2, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(400, ..., 100, ...)` и передаёт `100` wei. В ходе исполнения транзакции испускается событие `PurchasePayment(100)`. Транзакция исполняется успешно. Block timestamp: 390.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(2500, ..., 100, ...)` и передаёт `100` wei. В ходе вызова метода контракта выполняется команда `revert("The contract is being in not allowed state")`. Block timestamp: 2490.

Критерий оценивания (AC-004-04) : Отклонение платежей при появлении задолженности (13 баллов)

Зависит от успешного прохождения AC-004-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 2, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` с параметрами `(1500, ..., 100, ...)` и передаёт `100` wei. В ходе вызова метода контракта выполняется команда `revert("The contract is being in not allowed state")`. Block timestamp: 1490.

Серверная часть приложения: вторая итерация

Цель итерации: реализация второго и третьего пунктов из списка функционала.

Серверная часть должна поддерживать запросы согласно следующей схеме `GraphQL`:

```
type Query {
    authentication: Authentication

    rooms: [Room!]!
    room(id: ID!): Room!
}

type Mutation {
    requestAuthentication(address: String!): String!
    authenticate(
        address: String!
        signedMessage: InputSignature!
    ): Authentication!

    createRoom(room: InputRoom!): Room!
    editRoom(id: ID!, room: InputRoom!): Room!
    setRoomContractAddress(id: ID!, contractAddress: String): Room!
    setRoomPublicName(id: ID!, publicName: String): Room!
    removeRoom(id: ID!): Room!
}

# Authentication

type Authentication {
    address: String!
    isLandlord: Boolean!
}
```

```
# Rooms

type Room {
    id: ID!
    internalName: String!
    area: Float!
    location: String!

    contractAddress: String
    publicName: String
}

input InputRoom {
    internalName: String!
    area: Float!
    location: String!
}

input InputSignature {
    v: String!
    r: String!
    s: String!
}
```

Серверная часть приложения: приемочное тестирование второй итерации

US-103 Добавление помещений (10 баллов)

Зависит от успешного прохождения US-102

Описание: Я, как арендодатель, могу добавлять новые помещения в базу данных.

Критерий оценивания (AC-103-01) : Нормальное создание помещения (7 баллов)

---

1. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    createRoom(room: {
            internalName: "some-name",
            area: 100.5,
            location: "some location"
    }) {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "createRoom": {
            "id": "<room-id>",
            "internalName": "some-name",
            "area": 100.5,
            "location": "some location"
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "some-name",
            "area": 100.5,
            "location": "some location"
        }
    }
}
```

Критерий оценивания (AC-103-02) : Валидация данных помещения при создании (1 балл)

---

1. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    createRoom(room: {
            internalName: "some-name",
            area: -1,
            location: "some location"
    }) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "The room area must be greater than zero", }
    ]
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    createRoom(room: {
            internalName: "some-name",
            area: 0,
            location: "some location"
    }) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "The room area must be greater than zero" }
    ]
}
```

Критерий оценивания (AC-103-03) : Проверка роли при создании помещения (2 балла)

---

1. На сервер отправляется следующий запрос от имени неаутентифицированного пользователя:

```
mutation {
    createRoom(room: {
            internalName: "some-name",
            area: 100.5,
            location: "some location"
    }) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Authentication required" }
    ]
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
mutation {
    createRoom(room: {
            internalName: "some-name",
            area: 100.5,
            location: "some location"
    }) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the landlord" }
    ]
}
```

US-104 Задание адреса контракта для помещения (12 баллов)

Зависит от успешного прохождения US-103

Описание: Я, как арендодатель, могу устанавливать адрес контракта для помещений.

Критерий оценивания (AC-104-01) : Нормальная установка адреса контракта для помещения (4 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<contract-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "setRoomContractAddress": {
            "id": "<room-id>",
```

```
            "contractAddress": "<contract-address>"
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "contractAddress": "<contract-address>"
        }
    }
}
```

Критерий оценивания (AC-104-02) : Валидация ID помещения при установке адреса контракта (1 балл)

---

1. В базе данных не существует комнаты с ID `<nonexisting-room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(
        id: "<nonexisting-room-id>",
        contractAddress: "<contract-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room with such ID not found" }
    ]
}
```

Критерий оценивания (AC-104-03) : Проверка роли при установке адреса контракта (2 балла)

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<contract-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the landlord" }
    ]
}
```

2. На сервер отправляется следующий запрос от имени неаутентифицированного пользователя:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<contract-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Authentication required" }
    ]
}
```

Критерий оценивания (AC-104-04) : Редактирование адреса контракта помещения (1 балл)

1. В базе данных предварительно создана комната с ID `<room-id>` и ей установлен адрес контракта `<old-contract-address>`. В блокчейне развёрнут контракт с адресом `<new-contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<new-contract-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "setRoomContractAddress": {
            "id": "<room-id>",
            "contractAddress": "<new-contract-address>"
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "contractAddress": "<new-contract-address>"
        }
    }
}
```

Критерий оценивания (AC-104-05) : Удаление адреса контракта помещения (1 балл)

---

1. В базе данных предварительно создана комната с ID `<room-id>` и ей установлен адрес контракта `<old-contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(id: "<room-id>") {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "setRoomContractAddress": {
            "id": "<room-id>",
            "contractAddress": null
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, contractAddress
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "contractAddress": null
        }
    }
}
```

Критерий оценивания (AC-104-06) : Валидация адреса контракта при установке его для помещения (3 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<invalid-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Contract with such address not found" }
```

```
    ]
}
```

2. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне не существует развёрнутого контракта с адресом `<valid-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    setRoomContractAddress(
        id: "<room-id>",
        contractAddress: "<valid-address>"
    ) {
        id, contractAddress
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Contract with such address not found" }
    ]
}
```

US-105 Редактирование помещений (6 баллов)

Зависит от успешного прохождения US-103

Описание: Я, как арендодатель, могу редактировать свои помещения в базе данных.

Критерий оценивания (AC-105-01) : Нормальное редактирование помещения (2 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<old-name>`, `123` и `<old-location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    editRoom(id: "<room-id>",
        room: {
            internalName: "<new-name>",
            area: 42,
            location: "<new-location>"
        }
    ) {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "editRoom": {
            "id": "<room-id>",
            "internalName": "<new-name>",
            "area": 42,
            "location": "<new-location>"
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<new-name>",
            "area": 42,
            "location": "<new-location>"
        }
    }
}
```

Критерий оценивания (AC-105-02) : Валидация ID помещения при редактировании (1 балл)

---

1. В базе **не** существует комнаты с ID `<room-id>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    editRoom(id: "<room-id>",
        room: {
            internalName: "<new-name>",
            area: 42,
            location: "<new-location>"
        }
    ) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room with such ID not found" }
    ]
}
```

Критерий оценивания (AC-105-03) : Валидация данных помещения при редактировании (1 балл)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<old-name>`, `123` и `<old-location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

   ```
   mutation {
       editRoom(id: "<room-id>",
           room: {
               internalName: "<new-name>",
               area: -1.1,
               location: "<new-location>"
           }
       ) {
           id, internalName, area, location
       }
   }
   ```

   Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

   ```
   {
       "data": null,
       "errors": [
           { "message": "The room area must be greater than zero" }
       ]
   }
   ```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

   ```
   mutation {
       editRoom(id: "<room-id>",
           room: {
               internalName: "<new-name>",
               area: 0,
               location: "<new-location>"
           }
       ) {
           id, internalName, area, location
       }
   }
   ```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "The room area must be greater than zero" }
    ]
}
```

3. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<old-name>",
            "area": 123,
            "location": "<old-location>"
        }
    }
}
```

Критерий оценивания (AC-105-04) : Проверка роли при редактировании помещения (2 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<old-name>`, `123` и `<old-location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
mutation {
    editRoom(id: "<room-id>",
        room: {
            internalName: "<new-name>",
            area: 42,
            location: "<new-location>"
        }
    ) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the landlord" }
    ]
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<old-name>",
            "area": 123,
            "location": "<old-location>"
        }
    }
}
```

3. На сервер отправляется следующий запрос от имени неаутентифицированного пользователя:

```
mutation {
    editRoom(id: "<room-id>",
        room: {
            internalName: "<new-name>",
            area: 42,
            location: "<new-location>"
        }
    ) {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Authentication required" }
    ]
}
```

4. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<old-name>",
            "area": 123,
            "location": "<old-location>"
        }
    }
}
```

US-106 Удаление помещений (8 баллов)

Зависит от успешного прохождения US-103 и US-104

Описание: Я, как арендодатель, могу удалять свои помещения из базы данных.

Критерий оценивания (AC-106-01) : Нормальное удаление помещения (3 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<name>`, `123` и `<location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно, адрес контракта не задан. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

   ```
   mutation {
       removeRoom(id: "<room-id>") {
           id, internalName, area, location
       }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "removeRoom": {
               "id": "<room-id>",
               "internalName": "<name>",
               "area": 123,
               "location": "<location>"
           }
       }
   }
   ```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room with such ID not found" }
    ]
}
```

Критерий оценивания (AC-106-02) : Валидация данных помещения при удалении (3 балла)

---

1. В базе **не** существует комнаты с ID `<room-id>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    removeRoom(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room with such ID not found" }
    ]
}
```

2. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<name>`, 123 и `<location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом аккаунта `<tenant-address>` и с подписью арендодателя в качестве параметров. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
mutation {
    removeRoom(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room has rented contract in progress" }
    ]
}
```

3. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<name>",
            "area": 123,
            "location": "<location>"
        }
    }
}
```

Критерий оценивания (AC-106-03) : Проверка роли при удалении помещения (2 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с `<name>`, `123` и `<location>` в качестве значений атрибутов `internalName`, `area` и `location` соответственно. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
mutation {
    removeRoom(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the landlord" }
    ]
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<name>",
            "area": 123,
            "location": "<location>"
        }
    }
}
```

3. На сервер отправляется следующий запрос от имени неаутентифицированного пользователя:

```
mutation {
    removeRoom(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Authentication required" }
    ]
}
```

4. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    room(id: "<room-id>") {
        id, internalName, area, location
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "internalName": "<name>",
```

```
            "area": 123,
            "location": "<location>"
        }
    }
}
```

US-107 Просмотр списка помещений арендодателем (8 баллов)

Зависит от успешного прохождения US-105 и US-106

Описание: Я, как арендодатель, могу просматривать актуальный список своих помещений в базе данных.

Критерий оценивания (AC-107-01) : Получение начального списка помещений (5 баллов)

---

1. В базе данных предварительно созданы комнаты с ID `<room-1>` и `<room-2>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               { "id": "<room-1>" },
               { "id": "<room-2>" }
           ]
       }
   }
   ```

Критерий оценивания (AC-107-02) : Получения списка после создания помещений (1 балл)

---

1. В базе данных предварительно созданы комнаты с ID `<room-1>` и `<room-2>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
   ```

```
            { "id": "<room-1>" },
            { "id": "<room-2>" }
        ]
    }
}
```

2. Арендодатель создаёт помещение с ID `<room-3>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-2>" },
            { "id": "<room-3>" }
        ]
    }
}
```

Критерий оценивания (AC-107-03) : Получение списка после редактирования помещений (1 балл))

---

1. В базе данных предварительно созданы комнаты с ID `<room-1>` и `<room-2>` и названиями `<name-1>` и `<name-2>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    rooms { id, internalName }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            {
                "id": "<room-1>",
                "internalName": "<name-1>"
            },
            {
                "id": "<room-2>",
                "internalName": "<name-2>"
            }
        ]
    }
}
```

2. Арендодатель редактирует помещение с ID `<room-2>` и меняет название на `<new-name>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    rooms { id, internalName }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            {
                "id": "<room-1>",
                "internalName": "<name-1>"
            },
            {
                "id": "<room-2>",
                "internalName": "<new-name>"
            }
        ]
    }
```

Критерий оценивания (AC-107-04) : Получение списка после удаления помещений (1 балл)

---

1. В базе данных предварительно созданы комнаты с ID `<room-1>` и `<room-2>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-2>" }
        ]
    }
}
```

2. Арендодатель удаляет помещение с ID `<room-2>`. На сервер отправляется следующий запрос от имени аутентифицированного арендодателя:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" }
        ]
    }
}
```

US-108 Просмотр списка арендованных и доступных для аренды помещений (12 баллов)

Зависит от успешного прохождения US-104 и US-107

Описание: Я, как арендатор, могу просматривать список арендованных мною и доступных для аренды помещений.

Критерий оценивания (AC-108-01) : Получение пустого списка помещений (2 балла)

---

1. В базе данных предварительно создана комната с ID `<room-1>`, адреса контракта для неё **не задан**. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": []
       }
   }
   ```

2. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": []
       }
   }
   ```

Критерий оценивания (AC-108-02) : Получение списка помещений после изменения его арендодателем (4 балла)

1. В базе данных предварительно созданы комнаты с ID `<room-1>` и `<room-2>` и с `<name-1>` и `<name-2>` в качестве значений атрибута `internalName`. Арендодатель задаёт для них адреса контрактов, находящихся в готовом к аренде состоянии. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   query {
       rooms { id, internalName }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               {
                   "id": "<room-1>",
                   "internalName": "<name-1>"
               },
               {
                   "id": "<room-2>",
                   "internalName": "<name-2>"
               }
           ]
       }
   }
   ```

2. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

   ```
   query {
       rooms { id, internalName }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               {
                   "id": "<room-1>",
                   "internalName": "<name-1>"
               },
               {
                   "id": "<room-2>",
                   "internalName": "<name-2>"
               }
           ]
       }
   }
   ```

3. **Арендодатель** создаёт помещение с ID `<room-3>` и названием `<name-3>`, задаёт адрес контракта в состоянии, готовом к аренде. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-2>" },
            { "id": "<room-3>" }
        ]
    }
}
```

4. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-2>" },
            { "id": "<room-3>" }
        ]
    }
}
```

5. **Арендодатель** удаляет помещение с ID `<room-2>`. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-3>" }
        ]
    }
}
```

6. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

```
query {
    rooms { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" },
            { "id": "<room-3>" }
        ]
    }
}
```

7. **Арендодатель** задаёт помещению с ID `<room-1>` название `<new-name>`. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
query {
    rooms { id, internalName }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            {
                "id": "<room-1>",
                "internalName": "<new-name>"
            },
            {
                "id": "<room-3>",
                "internalName": "<name-3>"
            }
        ]
    }
}
```

8. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

```
query {
    rooms { id, internalName }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            {
                "id": "<room-1>",
                "internalName": "<new-name>"
            },
```

```
            {
                "id": "<room-3>",
                "internalName": "<name-3>"
            }
        ]
    }
}
```

9. **Арендодатель** удаляет адрес контракта у `<room-3>`. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               { "id": "<room-1>" }
           ]
       }
   }
   ```

10. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

    ```
    query {
        rooms { id }
    }
    ```

    Запрос выполняется успешно и возвращает следующий JSON-объект:

    ```
    {
        "data": {
            "rooms": [
                { "id": "<room-1>" }
            ]
        }
    }
    ```

Критерий оценивания (AC-108-03) : Получение списка после заключения договора об аренде (6 баллов)

---

1. В базе данных предварительно создана комната с ID `<room-1>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. **Арендодатель** устанавливает для комнаты с ID `<room-1>` адрес контракта `<contract-address>`. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-address>`:

   ```
   query {
       rooms { id }
   }
   ```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            { "id": "<room-1>" }
        ]
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-2-address>`:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               { "id": "<room-1>" }
           ]
       }
   }
   ```

3. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": [
               { "id": "<room-1>" }
           ]
       }
   }
   ```

4. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом аккаунта `<tenant-address>` и с подписью арендодателя в качестве параметров. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-address>`:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
   ```

```
            "rooms": [
                { "id": "<room-1>" }
            ]
        }
    }
```

5. На сервер отправляется следующий запрос от имени **другого** аутентифицированного арендатора с адресом аккаунта `<tenant-2-address>`:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": []
       }
   }
   ```

6. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": []
       }
   }
   ```

7. В блокчейне появляется блок с временной меткой, которая больше возвращаемой методом `getRentEndTime()` контракта комнаты с ID `<room-1>`. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-2-address>`:

   ```
   query {
       rooms { id }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "rooms": []
       }
   }
   ```

8. На сервер отправляется следующий запрос от имени аутентифицированного пользователя, не являющегося ни арендатором, ни арендодателем:

   ```
   query {
       rooms { id }
   }
   ```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": []
    }
}
```

US-109 Установка публичного названия арендуемого помещения (11 баллов)

Зависит от успешного прохождения US-104

Описание: Я, как арендатор, могу управлять арендованными мною помещениями и задавать им свои названия.

Критерий оценивания (AC-109-01) : Нормальная установка названия помещения (3 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом аккаунта `<tenant-address>` и с подписью арендодателя в качестве параметров. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-address>`:

   ```
   mutation {
       setRoomPublicName(id: "<room-id>", publicName: "<name>") {
           id, publicName
       }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "setRoomPublicName": {
               "id": "<room-id>",
               "publicName": "<name>"
           }
       }
   }
   ```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   query {
       room(id: "<room-id>") {
           id, publicName
       }
   }
   ```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "publicName": "<name>"
        }
    }
}
```

Критерий оценивания (AC-109-02) : Валидация ID помещения при установке названия (1 балл)

---

1. В базе данных не существует комнаты с ID `<nonexistent-id>`. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   mutation {
       setRoomPublicName(id: "<nonexistent-id>", publicName: "<name>") {
           id, publicName
       }
   }
   ```

   Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

   ```
   {
       "data": { "setRoomPublicName": null },
       "errors": [
           { "message": "Room with such ID not found" }
       ]
   }
   ```

Критерий оценивания (AC-109-03) : Проверка факта аренды помещения при установке названия(3 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` **без** адреса контракта. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   mutation {
       setRoomPublicName(id: "<room-id>", publicName: "<name>") {
           id, publicName
       }
   }
   ```

   Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

   ```
   {
       "data": { "setRoomPublicName": null },
       "errors": [
           { "message": "This room is not rented by you" }
   ```

```
    ]
}
```

2. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>` и с пустым публичным названием. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом аккаунта `<tenant-address>` и с подписью арендодателя в качестве параметров. На сервер отправляется следующий запрос от имени **другого** аутентифицированного арендатора с адресом аккаунта `<tenant-2-address>`:

   ```
   mutation {
       setRoomPublicName(id: "<room-id>", publicName: "<name>") {
           id, publicName
       }
   }
   ```

   Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

   ```
   {
       "data": { "setRoomPublicName": null },
       "errors": [
           { "message": "This room is not rented by you" }
       ]
   }
   ```

3. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

   ```
   query {
       room(id: "<room-id>") {
           id, publicName
       }
   }
   ```

   Запрос выполняется успешно и возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "room": {
               "id": "<room-id>",
               "publicName": null
           }
       }
   }
   ```

Критерий оценивания (AC-109-04) : Сброс публичного названия помещения(1 балл)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом

<contract-address>. Сервер успешно обработал запрос, содержащий setRoomContractAddress(<room-id>, <contract-address>). На стороне контракта успешно вызывается метод rent() от имени арендатора с адресом аккаунта <tenant-address> и с подписью арендодателя в качестве параметров. Сервер успешно обработал запрос, содержащий setRoomPublicName(<room-id>, <public-name>). На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта <tenant-address>:

```
mutation {
    setRoomPublicName(id: "<room-id>") {
        id, publicName
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "setRoomPublicName": {
            "id": "<room-id>",
            "publicName": null
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

```
query {
    room(id: "<room-id>") {
        id, publicName
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "publicName": null
        }
    }
}
```

Критерий оценивания (AC-109-05) : Редактирование публичного названия помещения (1 балл)

---

1. Сервер успешно обработал запрос, содержащий createRoom(<room>), создав комнату с ID <room-id>. В блокчейне развёрнут контракт с адресом <contract-address>. Сервер успешно обработал запрос, содержащий

`setRoomContractAddress(<room-id>, <contract-address>)`. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом аккаунта `<tenant-address>` и с подписью арендодателя в качестве параметров. Сервер успешно обработал запрос, содержащий `setRoomPublicName(<room-id>, <public-name>)`. На сервер отправляется следующий запрос от имени аутентифицированного арендатора с адресом аккаунта `<tenant-address>`:

```
mutation {
    setRoomPublicName(id: "<room-id>", publicName: "<new-name>") {
        id, publicName
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "setRoomPublicName": {
            "id": "<room-id>",
            "publicName": "<new-name>"
        }
    }
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

```
query {
    room(id: "<room-id>") {
        id, publicName
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "publicName": "<new-name>"
        }
    }
}
```

Критерий оценивания (AC-109-06) : Проверка роли при установке публичного названия (2 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На стороне контракта успешно вызывается метод `rent()` от имени арендатора с адресом акка-

унта `<tenant-address>` и с подписью арендодателя в качестве параметров. Сервер успешно обработал запрос, содержащий `setRoomPublicName(<room-id>, <old-name>)`. На сервер отправляется следующий запрос от имени **неаутентифицированного** пользователя:

```
mutation {
    setRoomPublicName(id: "<room-id>", publicName: "<name>") {
        id, publicName
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": { "setRoomPublicName": null },
    "errors": [
        { "message": "Authentication required" }
    ]
}
```

2. На сервер отправляется следующий запрос от имени аутентифицированного арендатора:

```
query {
    room(id: "<room-id>") {
        id, publicName
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "id": "<room-id>",
            "publicName": "<old-name>"
        }
    }
}
```

Пользовательский интерфейс приложения: вторая итерация

Цель итерации: реализация функций, связанных с помещениями.

Пользовательский интерфейс приложения: приемочное тестирование второй итерации

US-202 Добавление помещений (20 баллов)

Описание: Я, как арендодатель, могу добавлять новые помещения в базу данных.

Критерий оценивания (AC-202-01) : Добавление помещения (16 баллов)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/rooms`.
3. На странице нажимается ссылка `.rooms__create`.
4. Адрес страницы изменился на `http://solution-web/rooms/create`, а на самой странице появилась форма `.room-form` со следующими элементами:
   - `input[type="text"].room-form__internal-name:required` с пустым значением;
   - `input[type="number"].room-form__area:required` с пустым значением;
   - `input[type="text"].room-form__location:required` с пустым значением;
   - `button[type="submit"].room-form__submit`.
5. Внутри этой формы происходят следующие действия:
   5.1. В `.room-form__internal-name` указывается `Big grocery store`;
   5.2. В `.room-form__area` указывается `5000`;
   5.3. В `.room-form__location` указывается `To the right of the main entrance`;
   5.4. Нажимается кнопка `.room-form__submit`.
6. Адрес страницы изменился на `http://solution-web/room/<room-0>`.
7. В ответ на запрос

```
query {
    room(id: "<room-0>") {
        internalName
        area
        location
        publicName
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "internalName": "Big grocery store",
            "area": 5000,
            "location": "To the right of the main entrance",
            "publicName": null,
            "contractAddress": null
        }
    }
}
```

Критерий оценивания (AC-202-02) : Проверка права на добавление по-

мещений (4 балла)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/rooms`.
3. На странице присутствует ссылка `.rooms__create`.
4. Пользователь с аккаунтом `<address-0>` проходит аутентификацию на сайте.
5. В браузере открывается страница по адресу `http://solution-web/rooms`.
6. На странице **отсутствует** ссылка `.rooms__create`.

US-203 Просмотр помещений (15 баллов)

Описание: Я, как арендатор или арендодатель, могу просматривать информацию о помещениях.

Примечание: В примерах этого US предполагается, что в ответ на запрос

```
query {
    rooms {
        id
        internalName
        area
        location
        publicName
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
            {
                "id": "<room-0>",
                "internalName": "Big grocery store",
                "area": 5000,
                "location": "To the right of the main entrance",
                "publicName": null,
                "contractAddress": null
            },
            {
                "id": "<room-1>",
                "internalName": "Cinema small store",
                "area": 100,
                "location": "On the fifth floor near the cinema",
                "publicName": "Coffee & Coffee",
                "contractAddress": "<room-1-contract>"
            },
            {
```

```
                "id": "<room-2>",
                "internalName": "Medium store",
                "area": 400,
                "location": "On the third floor near the elevator",
                "publicName": null,
                "contractAddress": "<room-2-contract>"
            },
            {
                "id": "<room-3>",
                "internalName": "2nd floor small store",
                "area": 150,
                "location": "On the second floor in the north part",
                "publicName": "NFT toys",
                "contractAddress": "<room-3-contract>"
            }
        ]
    }
}
```

Кроме того,

- при вызове арендодателем метода `getTenant()` у контракта `<room-1-contract>` возвращается адрес `<tenant-1>`;
- при вызове метода `getRentStartTime()` у контракта `<room-1-contract>` возвращается значение `1647324000`;
- при вызове метода `getRentEndTime()` у контракта `<room-1-contract>` возвращается значение `1647583200`;
- при вызове метода `getBillingPeriodDuration()` у контракта `<room-1-contract>` возвращается значение `86400`;
- при вызове метода `getRentalRate()` у контракта `<room-1-contract>` возвращается значение `100`;
- у контракта `<room-2-contract>` ещё не вызывался метод `rent()`;
- при вызове арендодателем метода `getTenant()` у контракта `<room-3-contract>` возвращается адрес `<tenant-3>`;
- при вызове метода `getRentStartTime()` у контракта `<room-3-contract>` возвращается значение `1644575370`;
- при вызове метода `getRentEndTime()` у контракта `<room-3-contract>` возвращается значение `1647172800`;
- при вызове метода `getBillingPeriodDuration()` у контракта `<room-3-contract>` возвращается значение `2597430`;
- при вызове метода `getRentalRate()` у контракта `<room-3-contract>` возвращается значение `100`;

Критерий оценивания (AC-203-01) : Просмотр общей информации о помещении (8 баллов)

---

1. Арендодатель проходит аутентификацию на сайте.

2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице присутствуют следующие элементы:
   - `.room__name` с текстом `Big grocery store`;
   - `.room__area` с текстом `5000 sq.m.`;
   - `.room__location` с текстом `To the right of the main entrance`;
   - `.room__status` с текстом `Unavailable for renting`.
4. В браузере открывается страница по адресу `http://solution-web/room/<room-1>`.
5. На странице присутствуют следующие элементы:
   - `.room__name` с текстом `Coffee & Coffee`;
   - `.room__internal-name` с текстом `Cinema small store`;
   - `.room__area` с текстом `100 sq.m.`;
   - `.room__location` с текстом `On the fifth floor near the cinema`;
   - `.room__status` с текстом `Rented`;
   - `.room__contract-address` с текстом `<room-1-contract>`;
   - `.room__tenant` с текстом `<tenant-1>`;
   - `.room__rent-start` с текстом `Tue, 15 Mar 2022 06:00:00 GMT`;
   - `.room__rent-end` с текстом `Fri, 18 Mar 2022 06:00:00 GMT`;
   - `.room__billing-period` с текстом `1 day`;
   - `.room__rental-rate` с текстом `100 wei`.
6. В браузере открывается страница по адресу `http://solution-web/room/<room-2>`.
7. На странице присутствуют следующие элементы:
   - `.room__name` с текстом `Medium store`;
   - `.room__area` с текстом `400 sq.m.`;
   - `.room__location` с текстом `On the third floor near the elevator`;
   - `.room__status` с текстом `Available for renting`;
   - `.room__contract-address` с текстом `<room-2-contract>`.
8. В браузере открывается страница по адресу `http://solution-web/room/<room-3>`.
9. На странице присутствуют следующие элементы:
   - `.room__name` с текстом `NFT toys`;
   - `.room__internal-name` с текстом `2nd floor small store`;
   - `.room__area` с текстом `150 sq.m.`;
   - `.room__location` с текстом `On the second floor in the north part`;
   - `.room__status` с текстом `Rent ended`;
   - `.room__contract-address` с текстом `<room-3-contract>`;
   - `.room__tenant` с текстом `<tenant-3>`;
   - `.room__rent-start` с текстом `Fri, 11 Feb 2022 10:29:30 GMT`;
   - `.room__rent-end` с текстом `Sun, 13 Mar 2022 12:00:00 GMT`;
   - `.room__billing-period` с текстом `30 days 1 hour 30 minutes 30 seconds`;
   - `.room__rental-rate` с текстом `100 wei`.
10. Арендатор с аккаунтом `<tenant-1>` проходит аутентификацию на сайте.

11. В браузере открывается страница по адресу `http://solution-web/room/<room-1>`.
12. На странице присутствуют те же элементы, что и в пункте 5.

Критерий оценивания (AC-203-02) : Просмотр всех помещений арендодателем (5 баллов)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/rooms`.
3. На странице присутствуют следующие элементы:
   - `.room-card#room-<room-0>` с дочерними элементами:
     - `.room-card__name` с текстом `Big grocery store`;
     - `.room-card__status` с текстом `Unavailable for renting`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-0>`.
   - `.room-card#room-<room-1>` с дочерними элементами:
     - `.room-card__name` с текстом `Coffee & Coffee`;
     - `.room-card__status` с текстом `Rented`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-1>`.
   - `.room-card#room-<room-2>` с дочерними элементами:
     - `.room-card__name` с текстом `Medium store`;
     - `.room-card__status` с текстом `Available for renting`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-2>`.
   - `.room-card#room-<room-3>` с дочерними элементами:
     - `.room-card__name` с текстом `NFT toys`;
     - `.room-card__status` с текстом `Rent ended`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-3>`.
4. На странице нет других элементов, соответствующих селектору `.room-card`.

Критерий оценивания (AC-203-03) : Просмотр помещений арендатором (1 балл)

---

1. Арендатор с аккаунтом `<tenant-1>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/rooms`.
3. На странице присутствуют следующие элементы:
   - `.room-card#room-<room-1>` с дочерними элементами:
     - `.room-card__name` с текстом `Coffee & Coffee`;
     - `.room-card__status` с текстом `Rented`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-1>`.
   - `.room-card#room-<room-2>` с дочерними элементами:
     - `.room-card__name` с текстом `Medium store`;
     - `.room-card__status` с текстом `Available for renting`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-2>`.
4. На странице нет других элементов, соответствующих селектору `.room-card`.

5. Арендатор с аккаунтом `<tenant-3>` проходит аутентификацию на сайте.
6. В браузере открывается страница по адресу `http://solution-web/rooms`.
7. На странице присутствуют следующие элементы:
   - `.room-card#room-<room-3>` с дочерними элементами:
     - `.room-card__name` с текстом `NFT toys`;
     - `.room-card__status` с текстом `Rent ended`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-3>`.
   - `.room-card#room-<room-2>` с дочерними элементами:
     - `.room-card__name` с текстом `Medium store`;
     - `.room-card__status` с текстом `Available for renting`;
     - `a.room-card__details` с ссылкой на страницу `/room/<room-2>`.
8. На странице нет других элементов, соответствующих селектору `.room-card`.

Критерий оценивания (AC-203-04) : Просмотр помещений обычным пользователем (1 балл)

---

1. Пользователь, не арендовавший помещения, проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/rooms`.
3. На странице присутствует элемент `.room-card#room-<room-2>` с дочерними элементами:
   - `.room-card__name` с текстом `Medium store`;
   - `.room-card__status` с текстом `Available for renting`;
   - `a.room-card__details` с ссылкой на страницу `/room/<room-2>`.
4. На странице нет других элементов, соответствующих селектору `.room-card`.

US-204 Редактирование помещений (10 баллов)

Описание: Я, как арендодатель, могу редактировать свои помещения в базе данных.

Примечание: В примерах этого US предполагается, что в ответ на запрос

```
query {
    room(id: "<room-0>") {
        internalName
        area
        location
        publicName
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
```

```
        "room": {
            "internalName": "Big grocery store",
            "area": 5000,
            "location": "To the right of the main entrance",
            "publicName": "Electret",
            "contractAddress": "<room-0-contract>"
        }
    }
}
```

Кроме того, при вызове арендодателем метода `getTenant()` у контракта `<room-0-contract>` возвращается адрес `<tenant-0>`.

Критерий оценивания (AC-204-01) : Редактирование помещения (8 баллов)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается ссылка `.room__edit`.
4. Адрес страницы изменился на `http://solution-web/room/<room-0>/edit`, а на самой странице появилась форма `.room-form` со следующими элементами:
   - `input[type="text"].room-form__internal-name:required` со значением `Big grocery store`;
   - `input[type="number"].room-form__area:required` со значением `5000`;
   - `input[type="text"].room-form__location:required` со значением `To the right of the main entrance`;
   - `button[type="submit"].room-form__submit`.
5. Внутри этой формы происходят следующие действия:
   5.1. В `.room-form__internal-name` указывается `Large grocery store`;
   5.2. В `.room-form__area` указывается `5042.10`;
   5.3. Нажимается кнопка `.room-form__submit`.
6. Адрес страницы изменился на `http://solution-web/room/<room-0>`.
7. На странице присутствуют следующие элементы:
   - `.room__internal-name` с текстом `Large grocery store`;
   - `.room__area` с текстом `5042.1 sq.m.`;
   - `.room__location` с текстом `To the right of the main entrance`.
8. В ответ на запрос

   ```
   query {
       room(id: "<room-0>") {
           internalName
           area
           location
       }
   }
   ```

   сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "internalName": "Large grocery store",
            "area": 5042.1,
            "location": "To the right of the main entrance"
        }
    }
}
```

Критерий оценивания (AC-204-02) : Проверка права на редактирование помещений (2 балла)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице присутствует ссылка `.room__edit`.
4. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
5. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
6. На странице **отсутствует** ссылка `.room__edit`.

US-205 Установка публичного названия арендуемого помещения (9 баллов)

Описание: Я, как арендатор, могу задавать и редактировать публичное название арендованного мною помещения.

Примечание: В примерах этого US предполагается, что изначально в ответ на запрос

```
query {
    room(id: "<room-0>") {
        internalName
        publicName
        contractAddress
    }
}
```

сервер возвращал следующий JSON-объект:

```
{
    "data": {
        "room": {
            "internalName": "Medium store",
            "publicName": null,
            "contractAddress": "<room-0-contract>"
        }
```

```
    }
}
```

Кроме того, при вызове арендодателем метода `getTenant()` у контракта `<room-0-contract>` возвращается адрес `<tenant-0>`.

Критерий оценивания (AC-205-01) : Установка публичного названия (3 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается кнопка `.room__edit-public-name`.
4. На странице исчез элемент `.room__name`, появился элемент `.room__internal-name` с текстом `Medium store`, и появилась форма `.public-name-edit` со следующими элементами:
   - `input[type="text"].public-name-edit__name` с пустым значением;
   - `button[type="submit"].public-name-edit__submit`.
5. Внутри этой формы происходят следующие действия:
   5.1. В `.public-name-edit__name` указывается `63 Journals`;
   5.2. Нажимается кнопка `.public-name-edit__submit`.
6. На странице появился элемент `.room__name` с текстом `63 Journals`. Текст элемента `.room__internal-name` не изменился, а форма `.public-name-edit` исчезла.
7. В ответ на запрос

   ```
   query {
       room(id: "<room-0>") {
           publicName
       }
   }
   ```

   сервер возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "room": {
               "publicName": "63 Journals"
           }
       }
   }
   ```

Критерий оценивания (AC-205-02) : Редактирование публичного названия (2 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.

3. На странице нажимается кнопка `.room__edit-public-name`.
4. На странице исчез элемент `.room__name` и появилась форма `.public-name-edit` со следующими элементами:
   - `input[type="text"].public-name-edit__name` со значением `63 Journals`;
   - `button[type="submit"].public-name-edit__submit`.
5. Внутри этой формы происходят следующие действия:
   5.1. В `.public-name-edit__name` указывается `Books & Journals`;
   5.2. Нажимается кнопка `.public-name-edit__submit`.
6. На странице появился элемент `.room__name` с текстом `Books & Journals`.
7. В ответ на запрос

```
query {
    room(id: "<room-0>") {
        publicName
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "publicName": "Books & Journals"
        }
    }
}
```

Критерий оценивания (AC-205-03) : Удаление публичного названия (2 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается кнопка `.room__edit-public-name`.
4. На странице исчез элемент `.room__name` и появилась форма `.public-name-edit` со следующими элементами:
   - `input[type="text"].public-name-edit__name` со значением `Books & Journals`;
   - `button[type="submit"].public-name-edit__submit`.
5. Внутри этой формы происходят следующие действия:
   5.1. Значение `.public-name-edit__name` удаляется;
   5.2. Нажимается кнопка `.public-name-edit__submit`.
6. На странице появился элемент `.room__name` с текстом `Medium store`, а элемент `.room__internal-name` исчез со страницы.
7. В ответ на запрос

```
query {
    room(id: "<room-0>") {
        publicName
```

```
        }
    }
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "publicName": null
        }
    }
}
```

Критерий оценивания (AC-205-04) : Проверка права на изменение публичного названия (2 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице присутствует кнопка `.room__edit-public-name`.
4. Арендодатель проходит аутентификацию на сайте.
5. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
6. На странице **отсутствует** кнопка `.room__edit-public-name`.

US-206 Разрешение аренды помещения (20 баллов)

Описание: Я, как арендодатель, могу разрешать аренду помещения.

Примечание: В примерах этого US подразумевается, что в ответ на запрос

```
query {
    room(id: "<room-0>") {
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "contractAddress": null
        }
    }
}
```

Критерий оценивания (AC-206-01) : Разрешение аренды помещения (20 баллов)

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается кнопка `.room__allow-renting`.
4. MetaMask показывает новое всплывающее окно, в котором просит пользователя подтвердить развёртывание контракта. Пользователь подтверждает действие, нажав кнопку *Confirm*.
5. Текст элемента `.room__status` изменился на `Available for renting`. Также на странице появился элемент `.room__contract-address` с текстом `<room-0-contract>`, а кнопка `.room__allow-renting` исчезла.
6. При вызове метода `getRoomInternalId()` у контракта `<room-0-contract>` возвращается значение `<room-0>`.
7. При вызове метода `getLandlord()` у контракта `<room-0-contract>` возвращается значение `<landlord-address>`.
8. В ответ на запрос

   ```
   query {
       room(id: "<room-0>") {
           contractAddress
       }
   }
   ```

   сервер возвращает следующий JSON-объект:

   ```
   {
       "data": {
           "room": {
               "contractAddress": "<room-0-contract>"
           }
       }
   }
   ```

US-207 Удаление помещений (6 баллов)

Описание: Я, как арендодатель, могу удалять свои помещения из базы данных.

Примечание: В примерах этого US предполагается, что в ответ на запрос

```
query {
    rooms {
        id
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "rooms": [
```

```
            {
                "id": "<room-0>",
                "contractAddress": null,
            },
            {
                "id": "<room-1>",
                "contractAddress": "<room-1-contract>",
            }
        ]
    }
}
```

Критерий оценивания (AC-207-01) : Проверка возможности удаления (2 балла)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице присутствует кнопка `.room__remove`.
4. В браузере открывается страница по адресу `http://solution-web/room/<room-1>`.
5. На странице **отсутствует** кнопка `.room__remove`.

Критерий оценивания (AC-207-02) : Удаление помещений (4 балла)

---

1. Арендодатель проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается кнопка `.room__remove`.
4. Адрес страницы изменился на `http://solution-web/rooms`. На странице отсутствует элемент `.room-card#room-<room-0>`.
5. В ответ на запрос

   ```
   query {
       room(id: "<room-0>") {
           id
       }
   }
   ```

   сервер возвращает следующий JSON-объект:

   ```
   {
       "data": null,
       "errors": [
           { "message": "Room with such ID not found" }
       ]
   }
   ```

Третья итерация

Контракт соглашения об аренде: третья итерация

Цель итерации: реализация третьего пункта жизненного цикла контракта.

Контракт должен поддерживать следующий интерфейс:

- метод `view getTenantProfit() returns (uint)` возвращает не снятую сумму заработка арендатора за вычетом арендной платы.
- метод `nonpayable withdrawTenantProfit()` снимает заработок арендатора за вычетом арендной платы.
- метод `view getLandlordProfit() returns (uint)` возвращает не снятую сумму собранной арендной платы.
- метод `nonpayable withdrawLandlordProfit()` снимает сумму собранной арендной платы.
- метод `nonpayable endAgreement()` удаляет контракт из сети блокчейн.
- метод `nonpayable endAgreementManually(uint deadline, Sign landlordSign, Sign tenantSign)` удаляет контракт из сети блокчейн.

(*) Включая интерфейс предыдущих итераций.

Где параметры `landlordSign` и `tenantSign` сформированы из EIP-712 сигнатуры:

`EndConsent(uint256 deadline)`

под доменом

`EIP712Domain(string name,string version,address verifyingContract)`

где

`name` = `"Rental Agreement"`

`version` = `"1.0"`

(*) Коды ошибок и примеры поведения методов в разных контекстах приведены в секции приемочного тестирования этой итерации.

Контракт соглашения об аренде: приемочное тестирование третьей итерации

US-005 Вывод средств арендатора (21 баллов)

Зависит от успешного прохождения US-004

Описание: Я, как арендатор, могу вывести заработанные средства за вычетом арендной платы.

Критерий оценивания (AC-005-01) : Вывод средств арендатора (10 баллов)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт реги-

стрируется в сети блокчейн.

2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 4, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 0. Транзакция в блокчейн не отправляется. Block timestamp: 320.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `40` wei. Транзакция исполняется успешно. Block timestamp: 400.
6. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 0. Транзакция в блокчейн не отправляется. Block timestamp: 420.
7. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `210` wei. Транзакция исполняется успешно. Block timestamp: 500.
8. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 150. Транзакция в блокчейн не отправляется. Block timestamp: 600.
9. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. В результате исполнения транзакции на баланс аккаунта `<tenant-address>` должно перечислиться 150 wei со счета контракта. Транзакция исполняется успешно, но в блокчейн не отправляется. Block timestamp: 600.
10. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 150. Транзакция в блокчейн не отправляется. Block timestamp: 1400.
11. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. На баланс аккаунта `<tenant-address>` перечисляется 150 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 1400.

Критерий оценивания (AC-005-02) : Вывод средств арендатора в последний расчетный период (4 балла)

Зависит от успешного прохождения AC-005-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 2, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `250` wei. Транзакция исполняется успешно. Block timestamp: 400.

5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `160` wei. Транзакция исполняется успешно. Block timestamp: 1400.
6. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 310. Транзакция в блокчейн не отправляется. Block timestamp: 1450.
7. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. На баланс аккаунта `<tenant-address>` перечисляется 310 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 1450.

Критерий оценивания (AC-005-03) : Вывод средств арендатора с задолженностью (4 балла)

Зависит от успешного прохождения AC-005-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 5, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `330` wei. Транзакция исполняется успешно. Block timestamp: 400.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `40` wei. Транзакция исполняется успешно. Block timestamp: 1400.
6. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `getTenantProfit()`. Метод возвращает значение 230. Транзакция в блокчейн не отправляется. Block timestamp: 2400.
7. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. На баланс аккаунта `<tenant-address>` перечисляется 230 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 2400.

Критерий оценивания (AC-005-04) : Вывод средств арендатора по истечении срока действия контракта (3 балла)

Зависит от успешного прохождения AC-005-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 2, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()`

с параметром (’<cashier-1-address>’). Транзакция исполняется успешно.

4. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 330 wei. Транзакция исполняется успешно. Block timestamp: 400.
5. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 120 wei. Транзакция исполняется успешно. Block timestamp: 1400.
6. Аккаунт с адресом <tenant-address> вызывает у контракта метод getTenantProfit(). Метод возвращает значение 350. Транзакция в блокчейн не отправляется. Block timestamp: 2400.
7. Аккаунт с адресом <tenant-address> вызывает у контракта метод withdrawTenantProfit(). На баланс аккаунта <tenant-address> перечисляется 350 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 2400.

US-006 Вывод средств арендодателя (17 баллов)

Зависит от успешного прохождения US-004

Описание: Я, как арендодатель, могу вывести заработанные за счёт аренды средства.

Критерий оценивания (AC-006-01) : Вывод средств арендодателя (14 баллов)

---

1. Аккаунт с адресом <landlord-address> вызывает у контракта метод constructor с параметром (2419200). Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом <tenant-address> вызывает у контракта метод rent() с параметрами (310, ’<tenant-address>’, 100, 1000, 1, ...) и передаёт 100 wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом <tenant-address> вызывает у контракта метод addCashier() с параметром (’<cashier-1-address>’). Транзакция исполняется успешно.
4. Аккаунт с адресом <landlord-address> вызывает у контракта метод getLandlordProfit(). Метод возвращает значение 100. Транзакция в блокчейн не отправляется. Block timestamp: 400.
5. Аккаунт с адресом <landlord-address> вызывает у контракта метод withdrawLandlordProfit(). На баланс аккаунта <landlord-address> перечисляется 100 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 400.
6. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 50 wei. Транзакция исполняется успешно. Block timestamp: 500.
7. Аккаунт с адресом <landlord-address> вызывает у контракта метод getLandlordProfit(). Метод возвращает значение 0. Транзакция в блокчейн не отправляется. Block timestamp: 500.
8. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 160 wei. Транзакция исполняется успешно. Block timestamp: 600.
9. Аккаунт с адресом <landlord-address> вызывает у контракта метод getLandlordProfit(). Метод возвращает значение 0. Транзакция в блокчейн

не отправляется. Block timestamp: 700.

10. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `getLandlordProfit()`. Метод возвращает значение 0. Транзакция в блокчейн не отправляется. Block timestamp: 1400.

Критерий оценивания (AC-006-02) : Вывод средств арендодателя при появлении задолженности (3 балла)

Зависит от успешного прохождения AC-006-01

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 5, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()` с параметром `('<cashier-1-address>')`. Транзакция исполняется успешно.
4. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `330` wei. Транзакция исполняется успешно. Block timestamp: 400.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `40` wei. Транзакция исполняется успешно. Block timestamp: 1400.
6. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `getLandlordProfit()`. Метод возвращает значение 240. Транзакция в блокчейн не отправляется. Block timestamp: 2400.
7. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `withdrawLandlordProfit()`. На баланс аккаунта `<landlord-address>` перечисляется 240 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 2400.

US-007 Удаление контракта (36 баллов)

Зависит от успешного прохождения US-005 и US-006

Описание: Я, как пользователь (оракул), имеющий необходимые подписи, могу инициировать завершение контракта с сопутствующим возвратом средств.

Критерий оценивания (AC-007-01) : Удаление контракта по соглашению двух сторон (8 баллов)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()`. В ходе вызова метода контракта выполняется команда `revert("The contract is being in not allowed state")`. Транзакция в блокчейн не отправляется.

3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(110, '<tenant-address>', 100, 1000, 5, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 100.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()`. Транзакция исполняется успешно.
5. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()` с параметрами `(210, A, B)`, где `A` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<landlord-address>`, а `B` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<address-of-somebody>`. В ходе вызова метода контракта выполняется команда `revert("Invalid tenant sign")`. Транзакция в блокчейн не отправляется. Block timestamp: 200.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()` с параметрами `(220, A, B)`, где `A` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<address-of-somebody>`, а `B` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<tenant-address>`. В ходе вызова метода контракта выполняется команда `revert("Invalid landlord sign")`. Транзакция в блокчейн не отправляется. Block timestamp: 210.
7. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()` с параметрами `(230, A, B)`, где `A` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<landlord-address>`, а `B` - сообщение `EndConsent`, подписанное приватным ключом аккаунта `<tenant-address>`. В ходе вызова метода контракта выполняется команда `revert("The operation is outdated")`. Транзакция в блокчейн не отправляется. Block timestamp: 220.
8. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `30` wei. Транзакция исполняется успешно. Block timestamp: 230.
9. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()`. Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что контракт будет удалён из сети блокчейн, а на аккаунты `<tenant-address>` и `<landlord-address>` будет перечислено `30` и `100` wei соответственно. Block timestamp: 240.
10. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `90` wei. Транзакция исполняется успешно. Block timestamp: 250.
11. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()`. Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что контракт будет удалён из сети блокчейн, а на аккаунты `<tenant-address>` и `<landlord-address>` будет перечислено `120` и `100` wei соответственно. Block timestamp: 260.
12. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. На баланс аккаунта `<tenant-address>` перечисляется `20` wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 270.
13. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreementManually()`. Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что контракт будет удалён из сети блокчейн, а на аккаунты `<tenant-address>` и `<landlord-address>` будет перечислено `100` и `100` wei соответственно. Block timestamp: 280.
14. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `withdrawLandlordProfit()`. На баланс аккаунта `<landlord-address>` перечис-

ляется 100 wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 290.

15. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод endAgreementManually(). Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что контракт будет удалён из сети блокчейн, а на аккаунт <tenant-address> будет перечислено 100 wei. Block timestamp: 300.
16. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 40 wei. Транзакция исполняется успешно. Block timestamp: 310.
17. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод endAgreementManually(). Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что контракт будет удалён из сети блокчейн, а на аккаунт <tenant-address> будет перечислено 140 wei. Block timestamp: 320.
18. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод endAgreementManually(). Транзакция исполняется успешно. В результате исполнения транзакции контракт будет удалён из сети блокчейн, а на аккаунты <tenant-address> и <landlord-address> будет перечислено 40 и 100 wei соответственно. Block timestamp: 1300.

Критерий оценивания (AC-007-02) : Удаление контракта при появлении задолженности (14 баллов)

---

1. Аккаунт с адресом <landlord-address> вызывает у контракта метод constructor с параметром (2419200). Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом <landlord-address> вызывает у контракта метод endAgreement(). В ходе вызова метода контракта выполняется команда revert("The contract is being in not allowed state"). Транзакция в блокчейн не отправляется.
3. Аккаунт с адресом <tenant-address> вызывает у контракта метод rent() с параметрами (310, '<tenant-address>', 100, 1000, 3, ...) и передаёт 100 wei. Транзакция исполняется успешно. Block timestamp: 300.
4. Аккаунт с адресом <tenant-address> вызывает у контракта метод addCashier(). Транзакция исполняется успешно.
5. Аккаунт с адресом <landlord-address> вызывает у контракта метод endAgreement(). В ходе вызова метода контракта выполняется команда revert("The contract is being in not allowed state"). Транзакция в блокчейн не отправляется. Block timestamp: 400.
6. Аккаунт с адресом <address-of-somebody> вызывает у контракта метод pay() и передаёт 30 wei. Транзакция исполняется успешно. Block timestamp: 400.
7. Аккаунт с адресом <tenant-address> вызывает у контракта метод getTenantProfit(). Метод возвращает значение 0. Транзакция в блокчейн не отправляется. Block timestamp: 500.
8. Аккаунт с адресом <landlord-address> вызывает у контракта метод withdrawLandlordProfit(). Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что на баланс

аккаунта `<landlord-address>` будет перечислено `100` wei со счета контракта. Block timestamp: 600.

9. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `endAgreement()`. Транзакция исполняется успешно. Контракт удаляется из сети блокчейн, а на аккаунт `<landlord-address>` перечисляется `130` wei. Block timestamp: 3000.

Критерий оценивания (AC-007-03) : Удаление контракта по истечении срока действия соглашения (14 баллов)

---

1. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `constructor` с параметром `(2419200)`. Транзакция исполняется успешно. Контракт регистрируется в сети блокчейн.
2. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `endAgreement()`. В ходе вызова метода контракта выполняется команда `revert("The contract is being in not allowed state")`. Транзакция в блокчейн не отправляется.
3. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `rent()` с параметрами `(310, '<tenant-address>', 100, 1000, 2, ...)` и передаёт `100` wei. Транзакция исполняется успешно. Block timestamp: 300.
4. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `addCashier()`. Транзакция исполняется успешно.
5. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `endAgreement()`. В ходе вызова метода контракта выполняется команда `revert("The contract is being in not allowed state")`. Транзакция в блокчейн не отправляется. Block timestamp: 400.
6. Аккаунт с адресом `<address-of-somebody>` вызывает у контракта метод `pay()` и передаёт `130` wei. Транзакция исполняется успешно. Block timestamp: 500.
7. Аккаунт с адресом `<tenant-address>` вызывает у контракта метод `withdrawTenantProfit()`. Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что на баланс аккаунта `<tenant-address>` будет перечислено `30` wei со счета контракта. Транзакция исполняется успешно. Block timestamp: 600.
8. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `withdrawLandlordProfit()`. Транзакция исполняется успешно, но не включается в блок. В результате исполнения транзакции ожидается, что на баланс аккаунта `<landlord-address>` будет перечислено `100` wei со счета контракта. Block timestamp: 700.
9. Аккаунт с адресом `<landlord-address>` вызывает у контракта метод `endAgreement()`. Транзакция исполняется успешно. Контракт удаляется из сети блокчейн, а на аккаунты `<tenant-address>` и `<landlord-address>` перечисляется `30` и `200` wei соответственно. Block timestamp: 5000.

Серверная часть приложения: третья итерация

Цель итерации: реализация четвёртого и пятого пунктов из списка функционала.

Серверная часть должна поддерживать запросы согласно следующей схеме `GraphQL`:

```
type Query {
    authentication: Authentication

    rooms: [Room!]!
    room(id: ID!): Room!

    ticket(id: ID!): Ticket!
}

type Mutation {
    requestAuthentication(address: String!): String!
    authenticate(
        address: String!
        signedMessage: InputSignature!
    ): Authentication!

    createRoom(room: InputRoom!): Room!
    editRoom(id: ID!, room: InputRoom!): Room!
    setRoomContractAddress(id: ID!, contractAddress: String): Room!
    setRoomPublicName(id: ID!, publicName: String): Room!
    removeRoom(id: ID!): Room!

    createTicket(ticket: InputTicket!): Ticket!
}

# Authentication

type Authentication {
    address: String!
    isLandlord: Boolean!
}

# Rooms

type Room {
    id: ID!
    internalName: String!
    area: Float!
    location: String!

    contractAddress: String
    publicName: String
}
```

```
input InputRoom {
    internalName: String!
    area: Float!
    location: String!
}

# Tickets

type Ticket {
    id: ID!
    room: Room!
    value: Wei!
    deadline: Datetime!
    nonce: Nonce!
    cashierSignature: Signature!
}

input InputTicket {
    room: ID!
    nonce: InputNonce!
    value: InputWei!
    deadline: InputDatetime!
    cashierSignature: InputSignature!
}

# Primitives

## Signature

type Signature {
    v: String!
    r: String!
    s: String!
}

input InputSignature {
    v: String!
    r: String!
    s: String!
}

## Wei

type Wei {
    wei: String!
}

input InputWei {
    wei: String!
}
```

```
## Datetime
##
## In ISO 8601 format, e.g. `2022-03-15T06:00:00.000Z`

type Datetime {
    datetime: String!
}

input InputDatetime {
    datetime: String!
}

## Nonce

type Nonce {
    value: String!
}

input InputNonce {
    value: String!
}
```

Серверная часть приложения: приемочное тестирование третьей итерации

US-110 Регистрация квитанции на оплату (25 баллов)

Зависит от успешного прохождения US-104

Описание: Я, как кассир арендатора, могу выписывать квитанции через API.

Критерий оценивания (AC-110-01) : Нормальное создание квитанции (8 баллов)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На контракте был успешно вызван метод `addCashier(<cashier-address>)`. Кассир выписывает квитанцию на оплату с валидными данными и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
```

```
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    ) { id }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "ticket": {
            "id": "<ticket-id>"
        }
    }
}
```

2. На сервер отправляется следующий запрос от неаутентифицированного пользователя:
3.
```
query {
    ticket(id: "<ticket-id>") {
        room { id }, id
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "ticket": {
            "room": {
                 "id": "<room-id>"
            },
            "id": "<ticket-id>"
        }
    }
}
```

4. На сервер отправляется следующий запрос от аутентифицированного пользователя (арендатора, арендодателя или кассира):
5.
```
query {
    ticket(id: "<ticket-id>") {
        room { id }, id
```

```
    }
}
```

Запрос выполняется успешно и возвращает следующий JSON-объект:

```
{
    "data": {
        "ticket": {
            "room": {
                "id": "<room-id>"
            },
            "id": "<ticket-id>"
        }
    }
}
```

Критерий оценивания (AC-110-02) : Валидация данных при создании квитанции (7 баллов)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На контракте был успешно вызван метод `addCashier(<cashier-address>)`. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<nonexistent-room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room with such ID not found" }
    ]
}
```

2. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<invalid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Invalid nonce" }
    ]
}
```

3. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "asdasdasd"
            },
```

```
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Value must be an integer" }
    ]
}
```

4. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "1.1"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Value must be an integer" }
    ]
}
```

5. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "-1"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Value must be greater than zero" }
    ]
}
```

6. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "asdasdasd"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
```

```
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Invalid deadline date format" }
    ]
}
```

7. Кассир выписывает квитанцию на оплату и подписывает её. На сервер в момент времени `2022-03-15T06:00:00.000Z` отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "2021-03-15T06:00:00.000Z"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "The operation is outdated" }
    ]
}
```

8. Кассир выписывает квитанцию на оплату и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
```

```
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "asdasdasd",
                "r": "asdasdasd",
                "s": "asdasdasd"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Invalid cashier signature" }
    ]
}
```

9. Кассир выписывает квитанцию на оплату и подписывает её **чужим** приватным ключом, получив строку `<signature>`. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
```

```
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Unknown cashier" }
    ]
}
```

Критерий оценивания (AC-110-03) : Проверка роли при создании квитанции (4 балла)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Сервер успешно обработал запрос, содержащий `setRoomContractAddress(<room-id>, <contract-address>)`. На контракте был успешно вызван метод `addCashier(<cashier-address>)`. Кассир выписывает квитанцию на оплату с валидными данными и подписывает её. На сервер отправляется следующий запрос от имени **неаутентифицированного** пользователя:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
```

```
        { "message": "Authentication required" }
    ]
}
```

2. Кассир выписывает квитанцию на оплату с валидными данными и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного **арендодателя**:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the cashiers" }
    ]
}
```

3. Кассир выписывает квитанцию на оплату с валидными данными и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного **арендатора**:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
```

```
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
    )
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "This method is available only for the cashiers" }
    ]
}
```

Критерий оценивания (AC-110-04) : Проверка факта аренды помещения при создании квитанции (6 баллов)

---

1. Сервер успешно обработал запрос, содержащий `createRoom(<room>)`, создав комнату с ID `<room-id>`. В блокчейне развёрнут контракт с адресом `<contract-address>`. Адрес контракта для комнаты с ID `<room-id>` **не задан**. На контракте был успешно вызван метод `addCashier(<cashier-address>)`. Кассир выписывает квитанцию на оплату с валидными данными и подписывает её. На сервер отправляется следующий запрос от имени аутентифицированного кассира:

```
mutation {
    createTicket(
        ticket: {
            room: "<room-id>",
            nonce: {
                value: "<valid-nonce>"
            },
            value: {
                wei: "<valid-wei>"
            },
            deadline: {
                datetime: "<valid-datetime>"
            },
            cashierSignature: {
                "v": "<valid-v>",
                "r": "<valid-r>",
                "s": "<valid-s>"
            }
        }
```

```
    ) { id }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Room does not have a contract" }
    ]
}
```

US-111 Получение квитанции на оплату (5 баллов)

Зависит от успешного прохождения US-110

Описание: Я, как клиент арендатора, могу получить квитанцию на оплату, используя API.

Критерий оценивания (AC-111-01) : Валидация ID при получении квитанции на оплату (5 баллов))

---

1. На сервер отправляется следующий запрос от **неаутентифицированного** пользователя:

```
query {
    ticket(id: "<nonexistent-id>") {
        room: { id },
        id, value, deadline, cashierSignature
    }
}
```

Запрос выполняется с ошибкой и возвращает следующий JSON-объект:

```
{
    "data": null,
    "errors": [
        { "message": "Ticket with such ID not found" }
    ]
}
```

Пользовательский интерфейс приложения: третья итерация

Цель итерации: реализация функций, связанных с кассирами и квитанциями.

Пользовательский интерфейс приложения: приемочное тестирование третьей итерации

US-208 Управление кассирами (20 баллов)

Описание: Я, как арендатор, могу смотреть, добавлять и удалять кассиров.

Примечание: В примерах этого US подразумевается, что в ответ на запрос

```
query {
    room(id: "<room-0>") {
        contractAddress
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "room": {
            "contractAddress": "<room-0-contract>"
        }
    }
}
```

Кроме того,

- при вызове арендодателем метода `getTenant()` у контракта `<room-0-contract>` возвращается адрес `<tenant-0>`;
- при вызове метода `getCashiersList()` у контракта `<room-0-contract>` изначально возвращается массив `["<cashier-0>", "<cashier-1>"]`.

Критерий оценивания (AC-208-01) : Просмотр списка кассиров (3 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>/cashiers`.
3. На странице присутствует неупорядоченный список `.cashiers` с двумя пунктами. Каждый из пунктов содержит элементы из одного из следующих пунктов:
   - `.cashier__address` с текстом `<cashier-0>`;
   - `.cashier__address` с текстом `<cashier-1>`.

Критерий оценивания (AC-208-02) : Добавление кассира (8 баллов)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>/cashiers`.
3. На странице присутствует форма `.add-cashier` со следующими элементами:
   - `input[type="text"].add-cashier__address:required`;
   - `button[type="submit"].add-cashier__submit`.
4. Внутри этой формы происходят следующие действия:

- В `.add-cashier__address` указывается `<cashier-2>`;
- Нажимается кнопка `.add-cashier__submit`.

5. MetaMask показывает новое всплывающее окно, в котором просит пользователя подтвердить вызов метода `addCashier` у контракта `<room-0-contract>`. Арендатор подтверждает транзакцию, нажав кнопку *Confirm*.
6. Значение `.add-cashier__address` стало пустым.
7. В списке кассиров появился пункт для кассира с адресом `<cashier-0>`.
8. При вызове метода `getCashiersList()` у контракта `<room-0-contract>` возвращается массив, содержащий элементы `"<cashier-0>"`, `"<cashier-1>"` и `"<cashier-2>"` в любом порядке.

Критерий оценивания (AC-208-03) : Удаление кассира (5 баллов)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>/cashiers`.
3. На странице присутствует неупорядоченный список `.cashiers` с тремя пунктами. В пункте, содержащем элемент `.cashier__address` с текстом `<cashier-0>`, нажимается кнопка `.cashier__remove`.
4. MetaMask показывает новое всплывающее окно, в котором просит пользователя подтвердить вызов метода `removeCashier` у контракта `<room-0-contract>`. Арендатор подтверждает транзакцию, нажав кнопку *Confirm*.
5. В списке кассиров больше нет пункта для кассира с адресом `<cashier-0>`.
6. При вызове метода `getCashiersList()` у контракта `<room-0-contract>` возвращается массив, содержащий элементы `"<cashier-1>"` и `"<cashier-2>"` в любом порядке.

Критерий оценивания (AC-208-04) : Навигация на страницу управления кассирами (2 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
3. На странице нажимается ссылка `.room__manage-cashiers`.
4. Адрес страницы изменился на `http://solution-web/room/<room-0>/cashiers`.

Критерий оценивания (AC-208-05) : Проверка права на управление кассирами (2 балла)

---

1. Арендатор с аккаунтом `<tenant-0>` проходит аутентификацию на сайте.
2. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.

3. На странице присутствует ссылка `.room__manage-cashiers`.
4. Арендодатель проходит аутентификацию на сайте.
5. В браузере открывается страница по адресу `http://solution-web/room/<room-0>`.
6. На странице **отсутствует** ссылка `.room__manage-cashiers`.

US-209 Оплата квитанций (20 баллов)

Описание: Я, как покупатель, могу оплачивать квитанции, выписанные кассиром.

Примечание: В примерах этого US предполагается, что в ответ на запрос

```
query {
    firstTicket: ticket(id: "<ticket-0>") {
        room {
            publicName
            contractAddress
        }
        value { wei }
        deadline { datetime }
        nonce { nonce }
        cashierSignature { v, r, s }
    }
    secondTicket: ticket(id: "<ticket-1>") {
        room {
            internalName
            publicName
        }
        value { wei }
        deadline { datetime }
    }
}
```

сервер возвращает следующий JSON-объект:

```
{
    "data": {
        "firstTicket": {
            "room": {
                "publicName": "Coffee & Coffee",
                "contractAddress": "<room-0-contract>"
            },
            "value": { "wei": "1000" },
            "deadline": { "datetime": "2022-03-17T14:56:21.000Z" },
            "nonce": "<ticket-0-nonce>",
            "cashierSignature": {
                "v": "<signature-0-v>",
                "r": "<signature-0-r>",
                "s": "<signature-0-s>"
```

```
            }
        },
        "secondTicket": {
            "room": {
                "internalName": "2nd floor small store",
                "publicName": null
            },
            "value": { "wei": "1500" },
            "deadline": { "datetime": "2022-03-17T13:10:59.000Z" }
        }
    }
}
```

Кроме того, метод `pay()` у контракта `<room-0-contract>` никогда не вызывался с параметрами `(1647528981, <ticket-0-nonce>, 1000, <signature-0>)`.

Критерий оценивания (AC-209-01) : Отображение информации о квитанции (4 балла)

---

1. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-0>`.
2. На странице присутствуют следующие элементы:
   - `.ticket__store` с текстом `Coffee & Coffee`;
   - `.ticket__value` с текстом `1000 wei`;
   - `.ticket__deadline` с текстом `Thu, 17 Mar 2022 14:56:21 GMT`.
3. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-1>`.
4. На странице присутствуют следующие элементы:
   - `.ticket__store` с текстом `2nd floor small store`;
   - `.ticket__value` с текстом `1500 wei`;
   - `.ticket__deadline` с текстом `Thu, 17 Mar 2022 13:10:59 GMT`.

Критерий оценивания (AC-209-02) : Подключение к MetaMask (6 баллов)

---

1. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-0>`.
2. На странице нажимается кнопка `.account__connect`.
3. MetaMask показывает всплывающее окно, в котором просит пользователя выбрать аккаунт для подключения. Пользователь выбирает аккаунт с адресом `<address-0>` и предоставляет необходимые разрешения.
4. Кнопка `.account__connect` исчезает со страницы, и появляется новый элемент `.accont__address` с текстом `<address-0>`.
5. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-1>`.

6. На странице присутствует элемент `.accont__address` с текстом `<address-0>`.

Критерий оценивания (AC-209-03) : Проверка сроков оплаты (2 балла)

Примечание: В примерах этого AC предполагается, что текущее время — `2022-03-17T14:05:10.127Z`.

1. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-0>`.
2. Пользователь подключает свой аккаунт MetaMask.
3. На странице присутствует кнопка `.ticket__pay`.
4. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-1>`.
5. На странице **отсутствует** кнопка `.ticket__pay`, но присутствует элемент `.ticket__past-deadline` с текстом `Current time is past the deadline`.

Критерий оценивания (AC-209-04) : Проведение оплаты (8 баллов)

Примечание: В примерах этого AC предполагается, что текущее время — `2022-03-17T14:07:16.863Z`.

1. В браузере открывается страница по адресу `http://solution-web/ticket/<ticket-0>`.
2. Пользователь подключает свой аккаунт MetaMask.
3. На странице нажимается кнопка `.ticket__pay`.
4. MetaMask показывает новое всплывающее окно, в котором просит пользователя подтвердить вызов метода `pay` у контракта `<room-0-contract>`. Пользователь подтверждает транзакцию, нажав кнопку *Confirm*.
5. На странице появляется элемент `.ticket__success` с текстом `Payment successful`.