**Проект ShareIt. Техническое задание. Часть 1**

В этом модуле вы будете создавать сервис для шеринга (от англ. *share —* «делиться») вещей. Шеринг как экономика совместного использования набирает сейчас всё большую полярность. Если в 2014 году глобальный рынок шеринга оценивался всего в $15 млрд, то к 2025 может достигнуть $335 млрд.

Почему шеринг так популярен. Представьте, что на воскресной ярмарке вы купили несколько картин и хотите повесить их дома. Но вот незадача — для этого нужна дрель, а её у вас нет. Можно, конечно, пойти в магазин и купить, но в такой покупке мало смысла — после того, как вы повесите картины, дрель будет просто пылиться в шкафу. Можно пригласить мастера — но за его услуги придётся заплатить. И тут вы вспоминаете, что видели дрель у друга. Сама собой напрашивается идея — одолжить её.

Большая удача, что у вас оказался друг с дрелью и вы сразу вспомнили про него! А не то в поисках инструмента пришлось бы писать всем друзьям и знакомым. Или вернуться к первым двум вариантам — покупке дрели или найму мастера. Насколько было бы удобнее, если бы под рукой был сервис, где пользователи делятся вещами! Созданием такого проекта вы и займётесь.

**Что должен уметь новый сервис**

Ваш проект будет называться ShareIt. Он должен обеспечить пользователям, во-первых, возможность рассказывать, какими вещами они готовы поделиться, а во-вторых, находить нужную вещь и брать её в аренду на какое-то время.

Сервис должен не только позволять бронировать вещь на определённые даты, но и закрывать к ней доступ на время бронирования от других желающих. На случай, если нужной вещи на сервисе нет, у пользователей должна быть возможность оставлять запросы. Вдруг древний граммофон, который странно даже предлагать к аренде, неожиданно понадобится для театральной постановки. По запросу можно будет добавлять новые вещи для шеринга.

**Каркас приложения**

В этом спринте от вас требуется создать каркас приложения, а также разработать часть его веб-слоя. Основная сущность сервиса, вокруг которой будет строиться вся дальнейшая работа, — вещь. В коде она будет фигурировать как Item.

Пользователь, который добавляет в приложение новую вещь, будет считаться ее владельцем. При добавлении вещи должна быть возможность указать её краткое название и добавить небольшое описание. К примеру, название может быть — «Дрель “Салют”», а описание — «Мощность 600 вт, работает ударный режим, так что бетон возьмёт». Также у вещи обязательно должен быть статус — доступна ли она для аренды. Статус должен проставлять владелец.

Для поиска вещей должен быть организован поиск. Чтобы воспользоваться нужной вещью, её требуется забронировать. Бронирование, или Booking — ещё одна важная сущность приложения. Бронируется вещь всегда на определённые даты. Владелец вещи обязательно должен подтвердить бронирование.

После того как вещь возвращена, у пользователя, который её арендовал, должна быть возможность оставить отзыв. В отзыве можно поблагодарить владельца вещи и подтвердить, что задача выполнена — дрель успешно справилась с бетоном, и картины повешены.

Ещё одна сущность, которая вам понадобится, — запрос вещи ItemRequest. Пользователь создаёт запрос, если нужная ему вещь не найдена при поиске. В запросе указывается, что именно он ищет. В ответ на запрос другие пользовали могут добавить нужную вещь.

У вас уже готов шаблон проекта с использованием Spring Boot. Создайте ветку add-controllers и переключитесь на неё — в этой ветке будет вестись вся разработка для первого спринта.

**Реализация модели данных**

В этом модуле вы будете использовать структуру не по типам классов, а по фичам (англ. *Feature layout*) — весь код для работы с определённой сущностью должен быть в одном пакете. Поэтому сразу создайте четыре пакета — item, booking, request и user. В каждом из этих пакетов будут свои контроллеры, сервисы, репозитории и другие классы, которые вам понадобятся в ходе разработки. В пакете item создайте класс Item .

**Создание DTO-объектов и мапперов**

Созданные объекты Item и User вы в дальнейшем будете использовать для работы с базой данных (это ждёт вас в следующем спринте). Сейчас, помимо них, вам также понадобятся объекты, которые вы будете возвращать пользователям через REST-интерфейс в ответ на их запросы.

Разделять объекты, которые хранятся в базе данных и которые возвращаются пользователям, — хорошая практика. Например, вы можете не захотеть показывать пользователям владельца вещи (поле owner), а вместо этого возвращать только информацию о том, сколько раз вещь была в аренде. Чтобы это реализовать, нужно создать отдельную версию каждого класса, с которой будут работать пользователи, — DTO (*Data Transfer Object*).

Кроме DTO-классов, понадобятся Mapper-классы — они помогут преобразовывать объекты модели в DTO-объекты и обратно. Для базовых сущностей Item и User создайте Mapper-класс и метод преобразования объекта модели в DTO-объект.

**Разработка контроллеров**

Когда классы для хранения данных будут готовы, DTO и мапперы написаны, можно перейти к реализации логики. В приложении будет три классических слоя — контроллеры, сервисы и репозитории. В этом спринте вы будете работать преимущественно с контроллерами.

Для начала научите ваше приложение работать с пользователями. Ранее вы уже создавали контроллеры для управления пользователями — создания, редактирования и просмотра. Здесь вам нужно сделать то же самое. Создайте класс UserController и методы в нём для основных CRUD-операций. Также реализуйте сохранение данных о пользователях в памяти.

Далее переходите к основной функциональности этого спринта — работе с вещами. Вам нужно реализовать добавление новых вещей, их редактирование, просмотр списка вещей и поиск. Создайте класс ItemController. В нём будет сосредоточен весь REST-интерфейс для работы с вещью.

Вот основные сценарии, которые должно поддерживать приложение.

* Добавление новой вещи. Будет происходить по эндпойнту POST /items. На вход поступает объект ItemDto. userId в заголовке X-Sharer-User-Id — это идентификатор пользователя, который добавляет вещь. Именно этот пользователь — владелец вещи. Идентификатор владельца будет поступать на вход в каждом из запросов, рассмотренных далее.
* Редактирование вещи. Эндпойнт PATCH /items/{itemId}. Изменить можно название, описание и статус доступа к аренде. Редактировать вещь может только её владелец.
* Просмотр информации о конкретной вещи по её идентификатору. Эндпойнт GET /items/{itemId}. Информацию о вещи может просмотреть любой пользователь.
* Просмотр владельцем списка всех его вещей с указанием названия и описания для каждой. Эндпойнт GET /items.
* Поиск вещи потенциальным арендатором. Пользователь передаёт в строке запроса текст, и система ищет вещи, содержащие этот текст в названии или описании. Происходит по эндпойнту /items/search?text={text}, в text передаётся текст для поиска. Проверьте, что поиск возвращает только доступные для аренды вещи.

Для каждого из данных сценариев создайте соответственный метод в контроллере. Также создайте интерфейс ItemService и реализующий его класс ItemServiceImpl, к которому будет обращаться ваш контроллер. В качестве DAO создайте реализации, которые будут хранить данные в памяти приложения. Работу с базой данных вы реализуете в следующем спринте.

**Тестирование**

Для проверки кода мы подготовили [Postman-коллекцию](https://github.com/yandex-praktikum/java-shareit/blob/add-controllers/postman/sprint.json). С её помощью можно протестировать ваше API и убедиться, что все запросы успешно выполняются.

**Дополнительные советы ментора**

Если задание показалось вам недостаточно подробным, вы можете обратиться к этому файлу: [Дополнительные советы ментора](https://code.s3.yandex.net/Java/4mod1sprProject/mentors_advice_1.2.pdf).

В нём вы найдёте дополнительную информацию о том, как выполнить задание спринта. Но помните: реальные ТЗ часто скупы на подробности, поэтому разработчику приходится самостоятельно принимать некоторые архитектурные решения. Чем раньше вы научитесь определять минимальные требования, необходимые для начала разработки проекта, тем проще вам будет работать в команде над реальным проектом.

На этом пока всё! Но и это немало для хорошего старта! Удачного программирования!