**Описание решения**

Схема взаимодействия сервисов - <https://github.com/gennadytiunov/arch-labs/blob/master/transactions-sagas/Architecture/Service-Interaction.jpg>.

Сага построена с использованием стейт машины (на основе фреймворка <http://masstransit-project.com/usage/sagas/automatonymous.html>). Граф переходов состояний стейт машины - <https://github.com/gennadytiunov/arch-labs/blob/master/transactions-sagas/Architecture/State-Machine.jpg>. Исходный код описания стейт машины для данной работы, а также соответствующие внутренние команды / события - <https://github.com/gennadytiunov/arch-labs/tree/master/transactions-sagas/Source/Saga>.

Согласно best practises, стейт машина содержит только описание состояний и события, по которым выполняются переходы между состояниями (то есть код стейт машины не перемешивается с бизнес-логикой). Стейт машина (а также соответствующая сущность Booking) также реализует подсостояния, позволяющие более информативно описать, что происходит с бронированием в конкретный момент времени.

Стейт машина взаимодействует с внешним миром через BookingProcessor посредством внутренних событий и команд (содержащие только BookingId и дату их отправки). BookingProcessor транслирует внутренние события и команды во внешние, обогащая их необходимыми данными и отправляя их внешним потребителям через Kafka. Сообщения от внешних сервисов принимаются из Kafka и обрабатываются соответствующими Listeners, которые транслируют их во внутренние и отправляют в стейт машину. StateMachine, BookingProcessor, Listeners взаимодействуют через общую шину (Saga Bus), совместно образуя сущность BookingOrchestrator. Устройство и взаимодействия BookingOrchestrator - <https://github.com/gennadytiunov/arch-labs/blob/master/transactions-sagas/Architecture/Saga-Orchestrator.jpg>

В рамках данной лабораторной работы, в целях упрощения, состояние инстансов стейт машины / саги хранится в памяти (т.е. лабораторное решение не является горизонтально масштабируемым).

**Описание процесса бронирование**

* При регистрации пользователя в системе через User REST Endpoint для него автоматически создаётся платёжный Account в Payment Service.
* Перед покупкой билетов / бронированием мест на сеанс необходимо положить деньги в платёжный Account, используя метод DepositAmount в User REST Endpoint.
* Через Timetable REST Endpoint пользователь получает информацию об имеющихся в афише цирка представлениях (Show), расписании их сеансов (ShowSesssion) и стоимости.
* Через Booking REST Endpoint пользователь запрашивает информацию о доступных местах (HallSeats).
* Пользователь выполняет бронирование выбранных мест из списка доступных.
* Если места всё ещё доступны (то есть не были параллельно забронированы другим пользователем), Booking REST Endpoint создаёт Booking в статусе Created и указанием рассчитанной стоимости с учётом количества мест.
* Booking Endpoint отправляет событие BookingCreated.
* BookingMessageListener обнаруживает событие BookingCreated и создаёт StateMachine / Saga (экземпляры которой идентифицируются по BookingId).
* StateMachine отправляет внутреннюю команду MakeBooking в BookingProcessor, который:
  + извлекает необходимые данные о Booking из BookingRepository;
  + генерирует PaymentId и запоминает его в Booking;
  + отправляет внешнюю команду MakeBooking в Payment Service через Kafka.
* Кроме того, BookingProcessor отправляет событие PaymentInitiated в StateMachine и самому себе.
* Получив событие PaymentInitiated, StateMachine переходит в соответствующее состояние, а BookingProcessor обновляет состояние Booking через BookingRepository.
* В зависимости от успешности / неуспешности платежа в Payment Service, PaymentMessageListener получает либо PaymentSucceeded, либо PaymentFailed.
* Внешние сообщения MakeBooking, PaymentSucceeded и PaymentFailed коррелируются только по PaymentId, знанием о сущностях из BookingService они не обладают.
* PaymentSucceeded / PaymentFailed транслируются в соответствующее внутренне событие (обогащённое соответствующим BookingId) и отправляются в Saga Bus.
* BookingProcessor обновляет состояние Booking.
* StateMachine переход в состояние Failed и переходит к фазе отправки пользователю соответствующей нотификации, используя вышеописанный механизм взаимодействия...