РАЗРАБОТКА АЛГОРИТМА РАБОТЫ УНИВЕРСАЛЬНОЙ СИСТЕМЫ БРОНИРОВАНИЯ

О.Н. Цесько
Научный руководитель: И.А. Ботыгин
Томский политехнический университет
E-mail: oleg.tsesko@gmail.com

Введение

В докладе [1] рассказывалось об архитектуре проекта по системе бронирования, схеме построения и выборе инструментария для разработки. В настоящем докладе рассмотрен процесс реализации разработки базовой версии системы бронирования под работу в облачной платформе Google Firebase (с базой данных Firestore) на базе бессерверных вычислений (облачных функций) [2].

Основной и ключевой особенностью системы является представление универсальной архитектуры и функционального взаимодействия посредством делегированного иерархического обращения к базовым функциональным элементам (нодам). Вся сложность системы базируется на функциональной комбинации нод в иерархии с требуемым количеством уровней.

Технические особенности разрабатываемой системы

Площадка облачных функций от Firebase предоставляет возможность разрабатывать функции на базе различных современных платформ программирования на выбор пользователя. Для конкретной реализации в облачных функциях используется платформа NodeJS. Firebase предоставляет свои пакеты библиотек для NodeJS для более удобного использования платформы. К примеру, такими пакетами является модуль управления базой данных Firestore, модуль аутентификации, модуль доступа к другим облачным составляющим Firebase и т.д. Инициация вызова облачных функций в Firebase происходит несколькими различными способами, в перечень которых входит отправка HTTP-запросов на определённый URL-адрес каждой функции. Именно данный вид триггеров для вызова функций используется в конкретной реализации системы.

Разработка функций для базового слоя функциональных элементов

Шаг первый. Разработка функции для создания сущности для бронирования. На вход API данная функция ожидает, для создаваемой сущности, параметры сроков реализации бронирования, количество ресурсных мест (слотов), частоту генерации событий, правила генерации событий и др. Получив все необходимые параметры, алгоритм функции инициирует запрос в базу данных Firestore с целью сохранения новой полученной информации в виде сущности для бронирования. Результатом, возвращаемым в ответ на вызов функции, является уникальный идентификатор созданной сущности.

Шаг второй. Теперь разработаем функцию, которая проверяет наличие свободных мест для бронирования на указанную дату. Такая функция на вход API ожидает: дату для получения информации по ресурсам и идентификатор сущности для бронирования, который возвращается в ответ на вызов функции, описанной выше. Получив необходимые параметры, извлекаем из базы данных, используя полученный идентификатор, сущность для бронирования, а вместе с ней и все её правила генерации событий. Затем реализуется алгоритм сопоставления всех правил генерации событий к полученной дате потенциального бронирования. Если дата удовлетворяет всем правилам применения бронирования, то делаем запрос в базу данных по точной дате на наличие количества резервирований. В ответ на вызов функции возвращаем сумму всех положительных и отрицательных резервирований.

Шаг третий – функция по занятию мест. На вход API принимаются: точная дата для занятия мест и количество этих мест, чтобы можно было забронировать сразу слотов одним вызовом функции. Система, в момент вызова этой функции, предварительно «убеждается» в наличии свободных мест. Для этого достаточно повторить выполнение алгоритма по получению информации о свободных ресурсах, описанного выше. Если количество свободных мест достаточное для бронирования указанного количества слотов, добавляем то количество новых записей в базу данных, какое было указано при инициации бронирования. Важной особенностью является то, что эти записи помещаются в базу как положительное резервирование мест.

Шаг четвёртый – функция по освобождению мест (отмене брони, отрицательному резервированию). Для её реализации используются следующие параметры, поступающие на вход API: идентификатор сущности, точная дата и количество мест для отмены бронирования (отрицательного резервирования). Алгоритм предварительно проверяет, сделав запрос в базу данных, наличие возможности к отмене бронирования (чтобы общее количество занятых мест не ушло в минус). Если число, полученное в результате разницы имеющихся занятых мест и

количества мест, требуемых для снятия бронирования, оказывается более нуля, то мы можем продолжать выполнение функции. Реализуем записи в базы данных с отрицательным значением резервирования, характеризующие отмену бронирования. В ответ отправим сообщение об успехе.

Разработка функций для высокоуровневого слоя функциональных элементов

Как и в случае с базовым слоем нод, фундаментальный набор функций будет продублирован, но на более высоком уровне, делегирующем сложность структуры для бронирования между несколькими (многими) слоями. Здесь же, по ходу разработки фундаментальных функций для высокоуровневого слоя нод, описывается рекурсивная связь между слоями. Поскольку в данном случае одни и те же функции будут обрабатывать разные ветви ситуаций, связанных либо с уровнем L1, либо с уровнем LN, то необходимо в каждой из функций делать ветвление по факту того, с какими нижележащими нодами сейчас будут проводиться манипуляции.

Шаг первый. Для начала разрабатываем функцию по созданию высокоуровневой сущности для бронирования. На вход API данная функция принимает либо список идентификаторов базовых сущностей, либо список идентификаторов высокоуровневых сущностей. В первом случае, создаётся простейшее объединение (высокоуровневая сущность для бронирования), являющееся первым уровнем в рекурсии. Во втором случае, создаётся сложная сущность для бронирования, которая уже ссылается не на базовые элементы системы, а на другой слой абстракции, нижележащий, который, в свою очередь, также может ссылаться либо на более нижние слои, либо на базовый. Алгоритм, выбрав ту или иную ветвь развития событий, производит запись новой составной высокоуровневой сущности, включающей в себя ряд элементов, в базу данных. В ответ возвращается идентификатор вновь созданной сущности.

Шаг второй. Следующая функция – проверка количества ресурсов для бронирования в сложной сущности. На вход API принимаем дату и идентификатор сущности. Алгоритм выбирает из базы сущность по идентификатору, «проходит» по каждому составному элементу рекурсивно до низшего уровня, сопоставляет правила генерации событий каждого элемента в дереве рекурсии с полученной на вход датой, и определяет возможность получения мест. Если все правила выполнили успешную проекцию на указанную дату, то возвращаем минимальное количество мест среди всех составных элементов.

Шаг третий – бронирование в сложной сущности. На входе: идентификатор этой сущности, дата и количество требуемых свободных мест. Рекурсивно выбираем все элементы множества у данной высокоуровневой сущности, производим последовательное бронирование в каждом элементе рекурсивно. Если один из элементов отказал в бронировании по какой-либо причине, немедленно прекращаем череду бронирований в множестве, и отменяем резервирование в уже забронированных элементах, посредством вызова соответствующей функции отмены бронирования. Результат вызова функции – сообщение об успешном резервировании или ошибка.

Шаг четвёртый. Разрабатываем функцию по отмене бронирования высокоуровневой сущности. На входе API: идентификатор сущности, точная дата и количество слотов, требуемых для отрицательного резервирования (отмены бронирования). Затем, как и в предыдущих случаях, проходимся рекурсией по всем элементам сложной структуры и последовательно отменяем резервирование в каждом элементе. Если один из элементов вернёт ошибку по какой-либо причине, немедленно прекращаем отмену бронирования, а все элементы, которые уже успели выполнить указание, должны быть возвращены в предыдущее состояние посредством вызова, применительно к каждому, противоположной функции – обычного бронирования (положительного резервирования).

Заключение

Продемонстрирован процесс разработки базовых облачных функций сложной, универсальной системы бронирования. Рассмотрены фундаментальные функции и их взаимное использование по мере выполнения алгоритма. Благодаря раздельным облачным функциям, упрощается возможность их комбинации с целью достижения требуемого уровня сложной абстракции в приложении.

Список использованных источников

1. Цесько О.Н. Проектирование универсальной системы бронирования на базе бессерверных вычислений // Молодежь и современные информационные технологии. Сборник трудов XVII Международной научно-практической конференции студентов, аспирантов и молодых ученых. – Томск, 2020 (в печати).
2. Официальная документация AWS // Amazon. URL: https://www.aws.amazon.com (дата обращения: 14.01.2020).
3. Официальная документация Firebase // Google Firebase. URL: https://firebase.google.com (дата обращения: 14.01.2020).