УДК 004.415.2

ШАБЛОН ПРОЕКТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ SHARED MODELS

А. Д. Захаренко

Белорусский государственный университет,
пр. Независимости, 4, 220030, г. Минск, Беларусь, arsen.zaharenko@gmail.com
Научный руководитель: Е. В. Кремень, кандидат физико-математических наук, доцент

Описаны определение, характеристика и особенности использования шаблона проектирования веб-приложений Shared Models. Предлагается новый актуальный подход к решению проблемы консистентности данных в межсервисном взаимодействии. В результате выделены и охарактеризованы принцип работы шаблона, его преимущества и недостатки, а также возможные проблемы при использовании и решения для них.

Ключевые слова: шаблон проектирования; межсервисное взаимодействие; асинхронная коммуникация; консистентность данных; распределенные модели; CQRS; Redis.

Актуальность

Шаблоны проектирования используются для решения типовых проблем, возникающих при разработке веб-приложений, а также для повышения их гибкости, масштабируемости и обслуживаемости, поэтому изучение шаблонов проектирования является актуальным.

В случае сложных веб-систем, состоящих из нескольких сервисов, взаимодействующих между собой, консистентность данных является одной из главных проблем. Поскольку не существует четкого и стандартизированного решения для этой проблемы, предлагается рассмотреть применение шаблона *Shared Models* в качестве интересного подхода к ее решению.

Определение шаблона Shared Models

Shared Models (распределенные модели) – реализация информационного поля, основанная на моделях ORM, несущих в себе функционал передачи своих объектов на другие сервисы при их создании или изменении.

Распределенные модели представлены внутри специального shared-подмодуля, который интегрируется в каждый сервис, использующий такие модели. Использование подмодуля позволяет описать основные свойства моделей, а также базовый принцип передачи объектов в одном месте.

Шаблон *Shared Models* является разновидностью реализации CQRS архитектуры [1], т.к. операции записи и чтения распределенных моделей разделены. Особенность такой реализации в том, что операция записи провоцирует передачу объекта на другие сервисы.

Принцип работы

1. Для передачи данных используется резидентная база данных со своим шаблоном Pub/Sub [2], выступающая в роли брокера сообщений, например, Redis [3];

2. Все сервисы имеют доступ к одной и той же базе данных Redis;

3. Каждый сервис, взаимодействующий с распределенными моделями, имеет свою объектно-реляционную базу данных;

4. Все изменения, записанные в базу данных, связанные с распределенными моделями, должны быть представлены в формате JSON и отправлены в Redis-очередь;

5. Все сервисы должны непрерывно прослушивать привязанные к ним Redis-каналы и записывать все изменения объектов распределенных моделей в свою базу данных.

Преимущества использования

• *Консистентность данных*

Постоянное прослушивание Redis-каналов сервисами обеспечивает актуальность данных на каждом из сервисов и отсутствие конфликтов в будущем.

• *Простота введения*

Для использования в новых сервисах достаточно подключить специальный shared-сабмодуль и использовать распределенные модели, т.к. они уже несут в себе весь необходимый функционал.

• *Повышение производительности*

Асинхронный обмен данными обеспечивает более гибкую и отказоустойчивую архитектуру, где сервисы могут работать независимо друг от друга и обрабатывать сообщения по своему темпу [4].

Недостатки использования

• *Сложность*

Использование брокера сообщений требует дополнительной инфраструктуры, настройки и мониторинга.

• *Уязвимость*

Использование Redis-сервиса для передачи данных является самым уязвимым местом системы, что может потребовать дополнительного тестирования shared-функционала.

Возможные проблемы и их решение

• *Потеря соединения с Redis*

Выход из строя Redis-сервиса означает, что потерян способ передачи данных между сервисами, т.е. остановка работы всей системы. Для предотвращения такого сценария нужно периодически проверять работу Redis-сервиса и иметь в запасе возможность передачи данных через REST API [5].

• *Потеря соединения с одним или несколькими сервисами*

Выход из строя сервиса означает, что он может иметь устаревшую версию данных относительно прочих сервисов. Для предотвращения такого сценария нужно временно сохранять недошедшие сообщения на сервисе-отправителе и отправлять повторно после восстановления принимающего сервиса.

• *Избыточная обработка полей объекта*

При изменении объекта на одном из сервисов лучше отправлять не все данные об объекте, а только измененные поля, т.е. дельту между исходным состоянием и его новым состоянием после редактирования.

Библиографические ссылки

1. CQRS [Electronic resource]. URL: https://martinfowler.com/bliki/CQRS.html (date of access: 16.03.2024).

2. Publish-subscribe pattern [Electronic resource]. URL: https://en.wikipedia.org/wiki/Publish-subscribe_pattern (date of access: 16.03.2024).

3. Introduction to Redis [Electronic resource]. URL: https://redis.io/docs/about (date of access: 16.03.2024).

4. System Design A Comprehensive Guide on Synchronous & Asynchronous Microservice Communication [Electronic resource] URL: https://medium.com/@systemdesignbychk/system-design-a-comprehensive-guide-on-synchronous-asynchronous-microservice-communication-8bda324943b8 (date of access: 16.03.2024).

5. REST vs Message Brokers: Choosing the Right Communication [Electronic resource] URL: https://memphis.dev/blog/comparing-rest-and-message-brokers-choosing-the-right-communication (date of access: 16.03.2024).