**ЛАБОРАТОРНАЯ РАБОТА № 10**

**РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ СООБЩЕНИЙ С ИСПОЛЬЗОВАНИЕМ APACHE KAFKA**

Цель работы: Разработка системы асинхронной обработки сообщений с использованием Apache Kafka, включая создание продюсеров, консьюмеров и настройку их заимодействия.

Программное обеспечение: Apache Kafka 3.0+; ZooKeeper 3.7+; Java 11+ / Python 3.8+; IntelliJ IDEA / VS Code; Docker (опционально)

Необходимая теоретическая подготовка: Основы распределённых систем; Принципы работы брокеров сообщений; API Kafka (Producer/Consumer); JSON-сериализация данных.

**1. Установка и настройка Kafka**

**1.1 Запуск ZooKeeper**

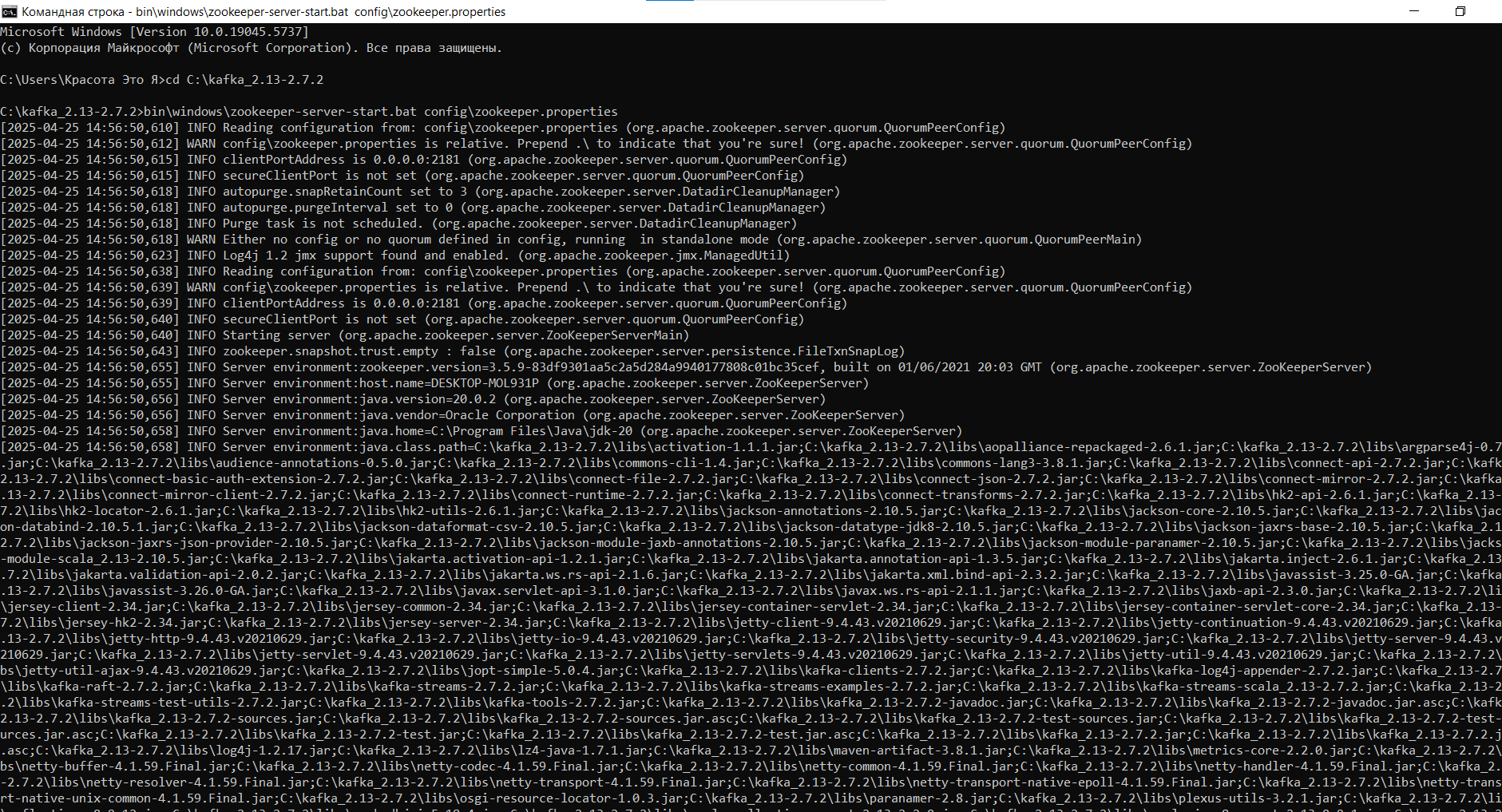


Рисунок 1.1 – **Запуск ZooKeeper**

**1.2 Запуск Kafka-брокер**

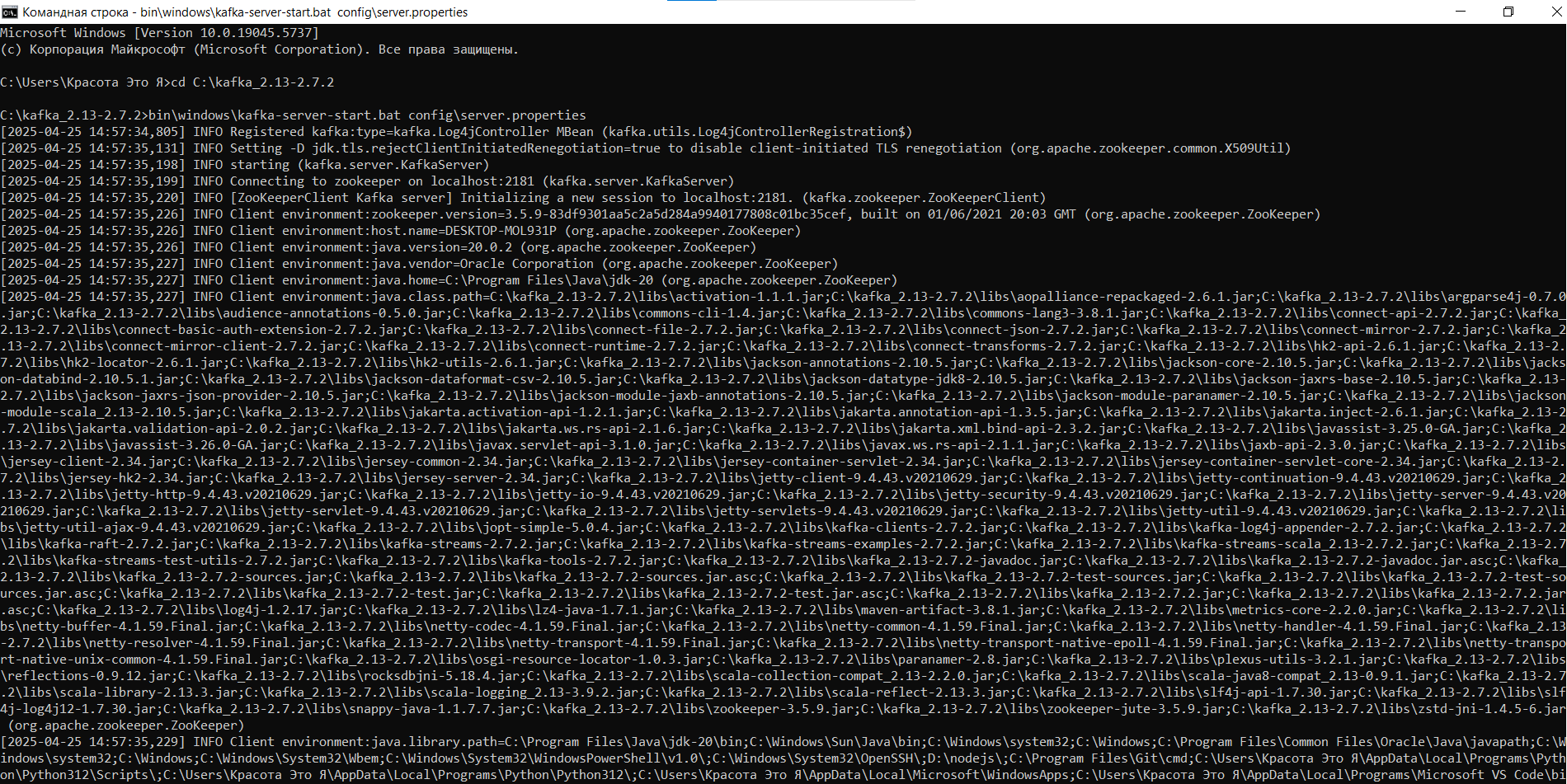


Рисунок 1.2 – **Запуск Kafka-брокер**

**1.3** **Создание топика user\_actions с 3 партициями.**

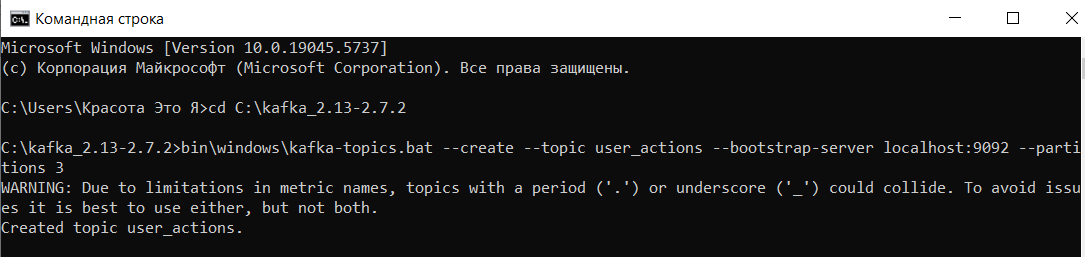


Рисунок 1.3 – **Создание** **топика user\_actions**

**2. Разработка продюсера.**

**2.1. Сборка проекта.**

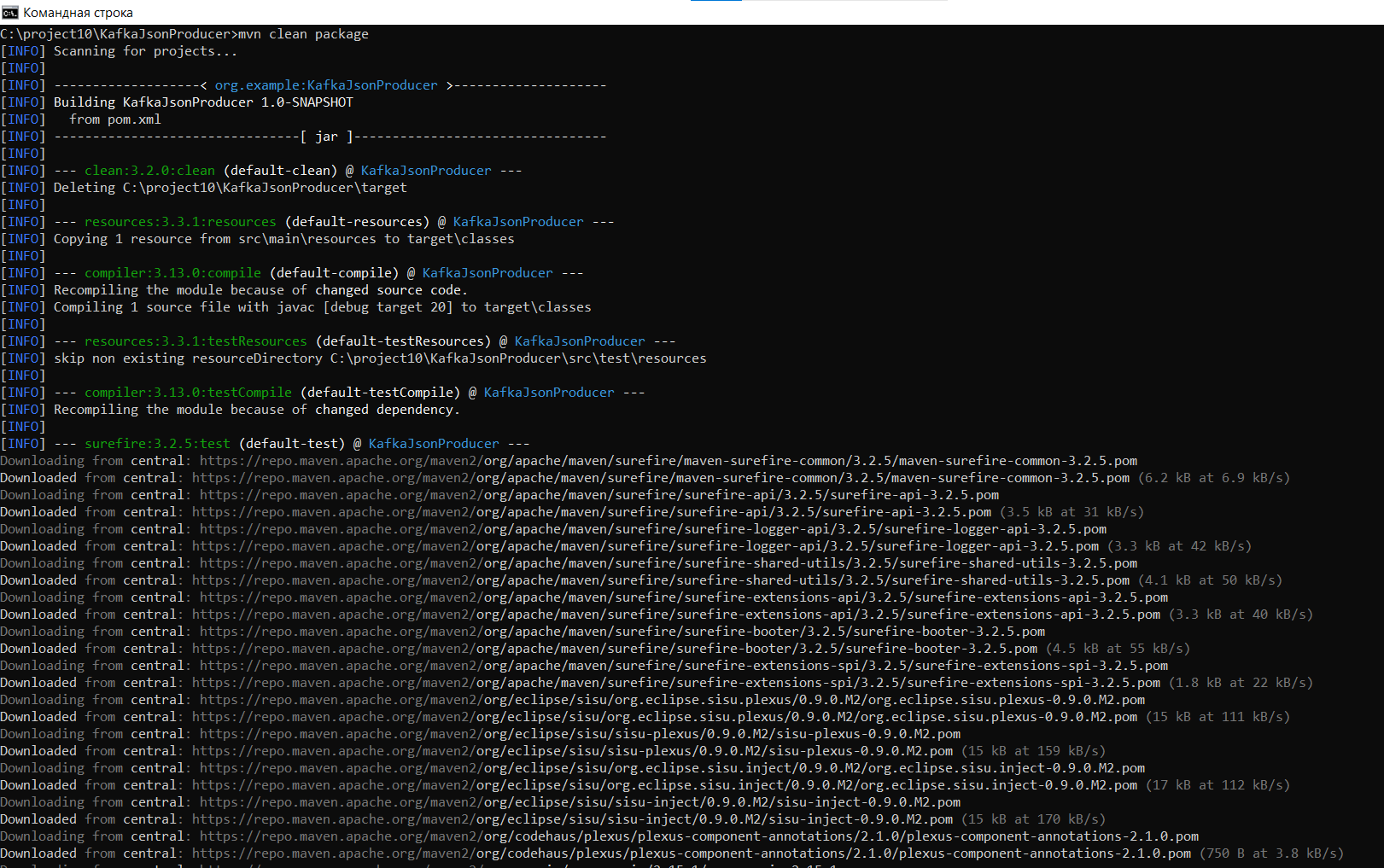


Рисунок 2.1 – **Сборка проекта**

**2.2 Отправка сообщений.**

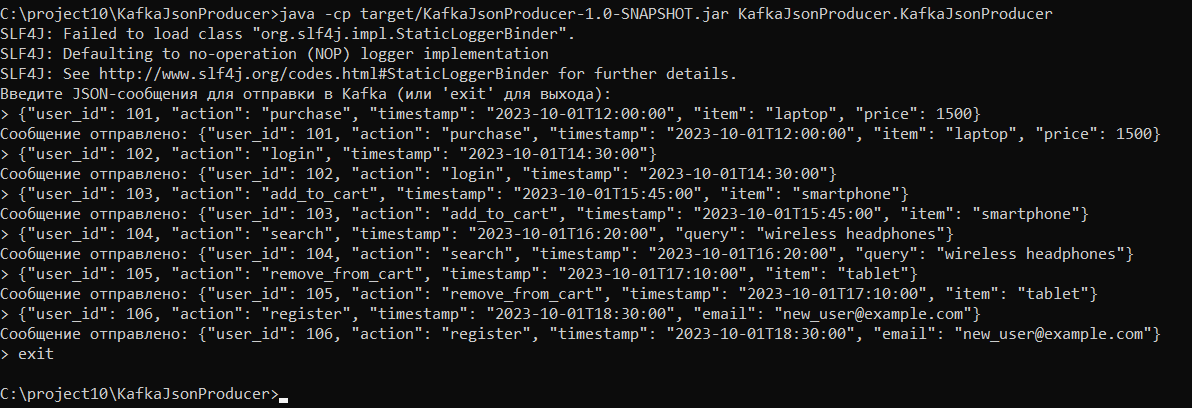


Рисунок 2.2 – **Отправка сообщений**

**2.3 Сообщение в топике user\_actions.**

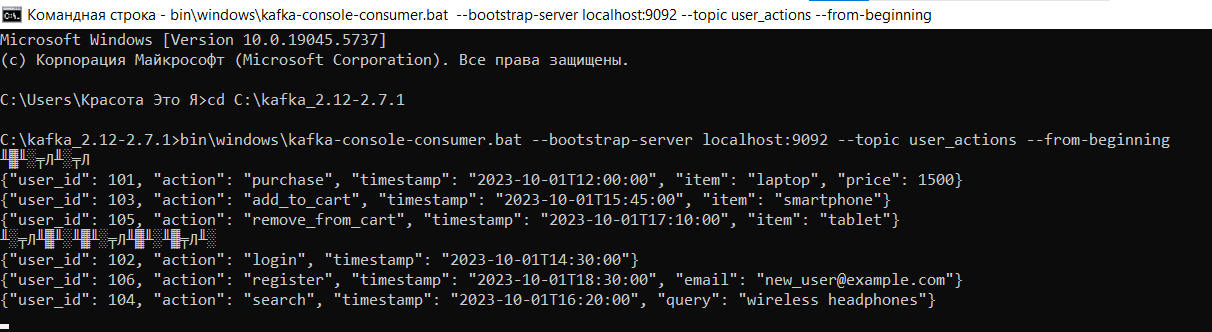


Рисунок 2.2 – **Топик user\_actions**

**3. Разработка консьюмера**

**3.1 Сборка проекта.**

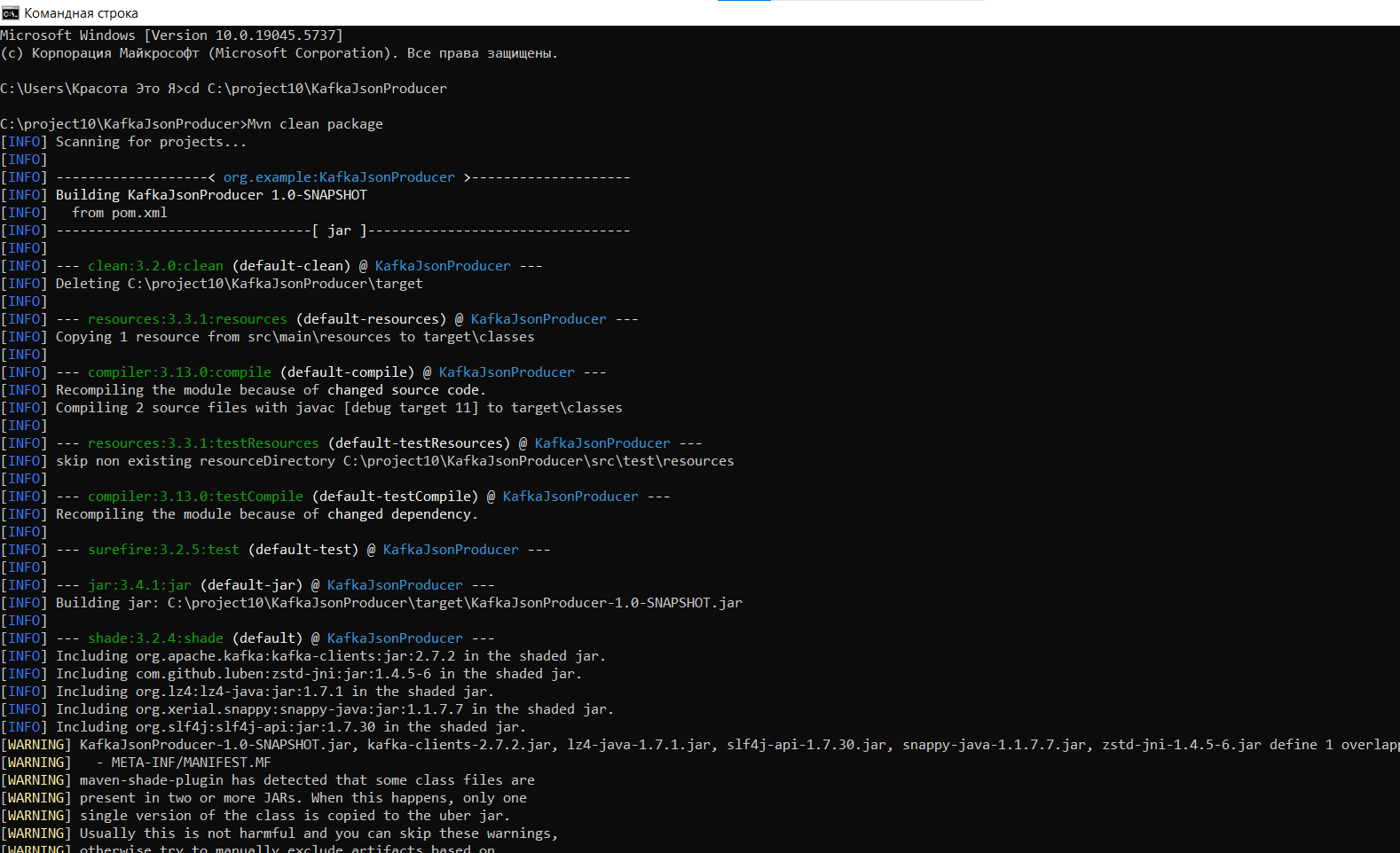


Рисунок 3.1 – **Сборка проекта**

**3.1 Обработка консьюмером:**

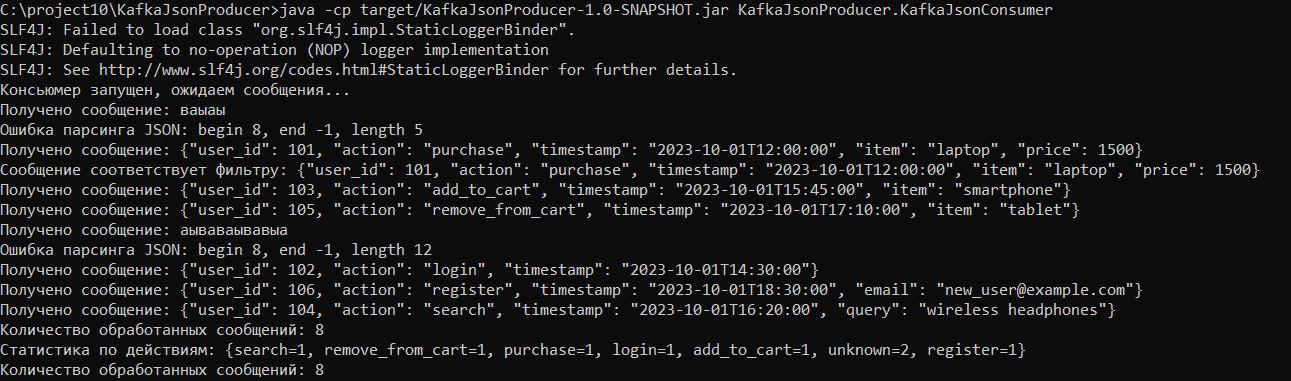


Рисунок 3.2 – **Обработка сообщений консьюмером**

**4. Обработка ошибок и масштабирование**

**4.1 Создание топика для некорректных сообщений**

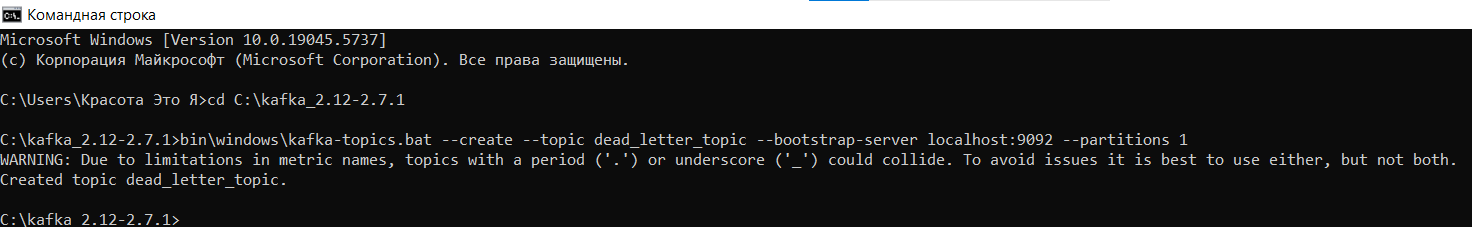


Рисунок 4.1 – **Создание топика dead\_letter\_topic**

**4.2 Запуск продюсера и отправка сообщений.**

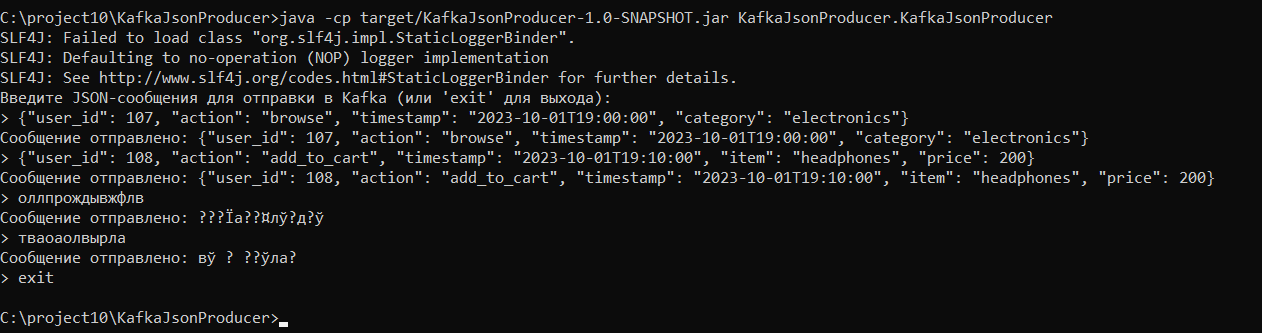


Рисунок 4.2 – **Запуск продюсера и отправка сообщений**

**4.3 Обработка Консюмером 1**

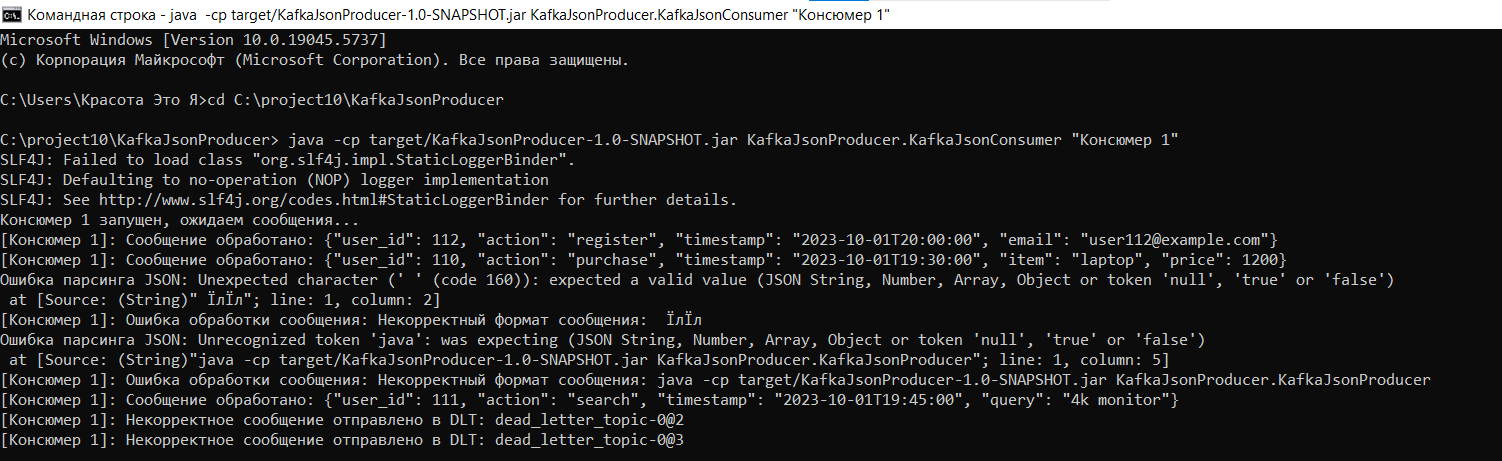


Рисунок 4.3 – **Обработка Консюмером 1**

**4.4 Обработка Консюмером 2**

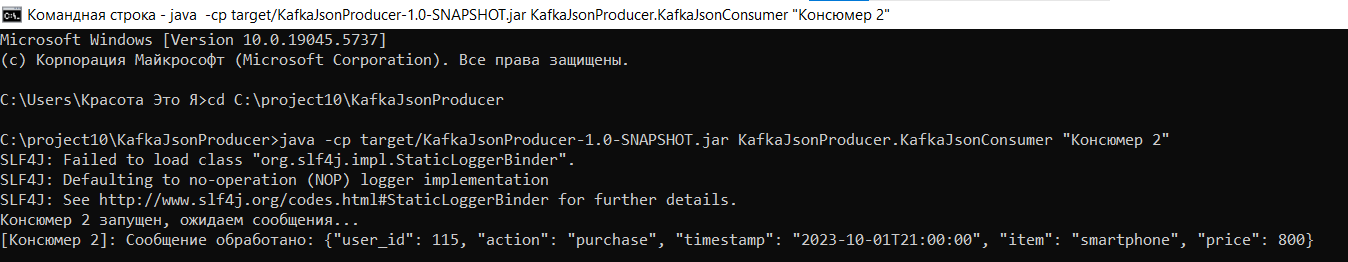


Рисунок 4.4 – **Обработка Консюмером 2**

**4.5 Топик dead\_letter\_topic:**

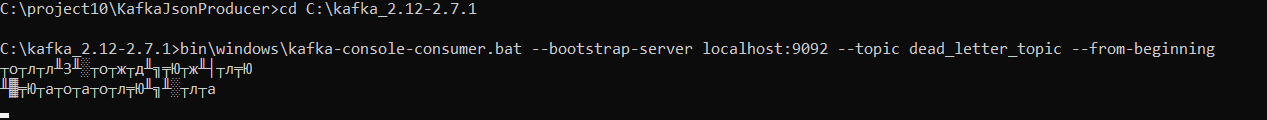


Рисунок 4.5 – **Топик dead\_letter\_topic**

**5. Интеграция с внешней системой**

**5.1 Отправка сообщений.**

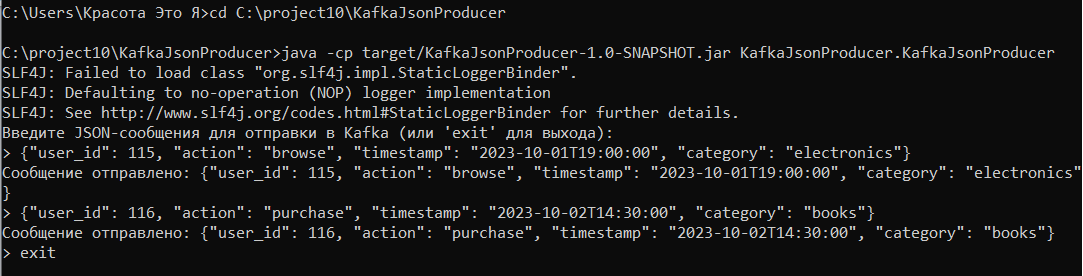


Рисунок 5.1 – **Отправка сообщений**

**5.2 Обработка и сохранение сообщений.**

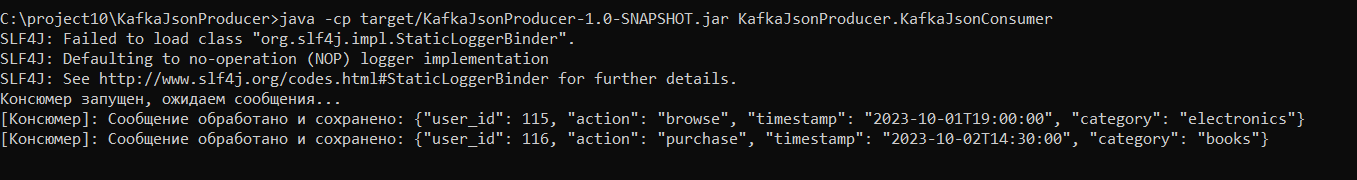


Рисунок 5.2 – **Обработка и сохранение сообщений**

**5.3 Результат в базе данных**

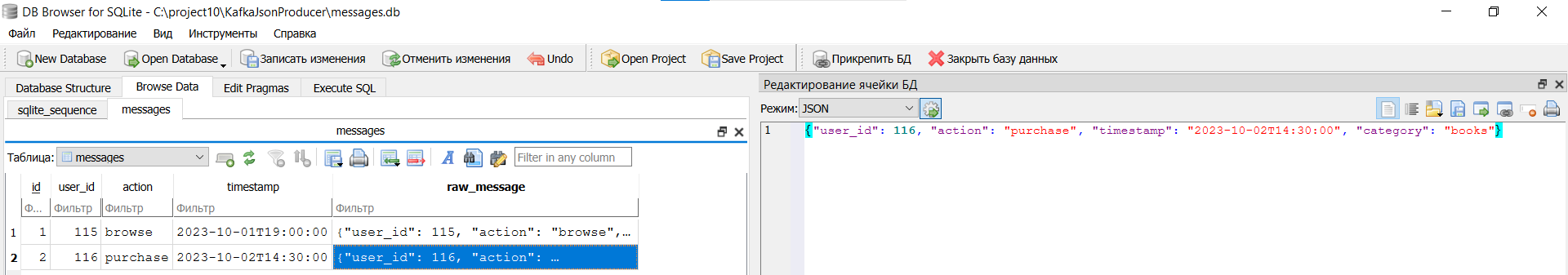


Рисунок 5.3 – **Данные в БД SQLite**

**Вывод:** Kafka предоставляет значительные преимущества в обработке данных. Она обеспечивает высокую производительность, надёжность и масштабируемость, позволяя эффективно управлять большими потоками информации и гарантировать сохранение порядка сообщений внутри партиций. Интеграция с внешними системами и множество доступных коннекторов делают её гибкой и универсальной для различных задач.

Однако в процессе работы возникли некоторые сложности. Настройка связки Kafka и ZooKeeper потребовала внимательного подхода, особенно при координации брокеров и потребителей. Также вызвало затруднения определение начальной точки чтения сообщений консюмерами, связанное с параметром auto.offset.reset. Различие между значениями "earliest" и "latest" иногда создаёт путаницу, особенно если необходимо решить, какие данные консюмер должен обрабатывать: старые или только поступающие сообщения. Эти аспекты требуют чёткой настройки и понимания принципов работы системы.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

**1. Почему Kafka использует ZooKeeper?**

Kafka использует **ZooKeeper** для координации и управления кластером. Основные задачи ZooKeeper включают:

* Управление метаданными брокеров Kafka: ZooKeeper отслеживает, какие брокеры активны и готовы принимать данные.
* Выбор лидера: ZooKeeper помогает определить лидера партиции, который отвечает за запись и чтение данных.
* Координация потребителей: ZooKeeper отслеживает состояния консюмеров в группе, чтобы корректно распределять партиции. В более новых версиях Kafka (начиная с версии 2.8), она начинает постепенно заменять ZooKeeper на **KRaft** — собственный механизм координации, встроенный в Kafka.

**2. Как обеспечить порядок сообщений в партиции?**

Чтобы сообщения сохраняли порядок внутри одной партиции:

* **Используйте ключи сообщений**: Kafka распределяет сообщения по партициям на основе ключа. Сообщения с одинаковым ключом всегда попадают в одну и ту же партицию. Например, ключ может быть user\_id.
* **Один продюсер — один партиционный поток**: если один продюсер отправляет сообщения в одну партицию, порядок будет сохранён.
* **Не используйте механизм репликации для записи**: репликация может влиять на порядок, так как последующие копии партиции могут отставать от основной.

**3. Разница между auto.offset.reset=earliest и latest**

**Earliest**: если нет сохранённого оффсета для консюмера, он начинает читать **с самого начала** партиции.Это полезно, если нужно обработать все доступные сообщения.

**Latest**: если нет сохранённого оффсета для консюмера, он начинает читать **только новые сообщения**, игнорируя старые. Используется для потоковой обработки, где важны только актуальные данные.

**4. Как Kafka обеспечивает отказоустойчивость?**

Kafka достигает отказоустойчивости благодаря следующим механизмам:

* **Репликация**: каждая партиция имеет одну основную копию (лидера) и несколько реплик. Если лидер недоступен, одна из реплик автоматически становится новым лидером.
* **Индексация данных**: сообщения хранятся на диске с индексами, что позволяет восстановить данные даже после сбоя.
* **Долговечность оффсетов**: Kafka сохраняет оффсеты консюмеров либо в ZooKeeper, либо в специальной внутренней теме (\_\_consumer\_offsets).
* **Координация через ZooKeeper или KRaft**: ZooKeeper следит за состоянием брокеров и групп консюмеров, позволяя быстро восстановить работу после сбоя.