МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

КУРСОВАЯ РАБОТА

по дисциплине «Архитектура вычислительных и информационных систем»

Тема: Система автоматической модерации отзывов Review Moderation System

Студентка гр. 2308 _____ Рымарь М.И.

Студент гр. 2308 _____ Мелихов М.А.

Студент гр. 2308 _____ Придчин В.Е.

Преподаватель _____ Подклетнов С.Г.

Санкт-Петербург
2025

СОДЕРЖАНИЕ

	Введение	**3**
	Цель и задачи работы	**4**
1.	**Постановка задачи**	**5**
2.	**Архитектура системы**	**6**
3.	**UML-диаграммы**	**7**
4.	**Развёртывание и тестирование**	**11**
5.	**Распределение ролей в команде**	**13**
	Заключение	**14**

Введение

В современном цифровом пространстве пользователи активно оставляют отзывы о товарах, сервисах и компаниях. Такие отзывы являются важным источником обратной связи, но вместе с тем — источником проблем. В большом объёме пользовательского контента часто встречаются спам-сообщения, оскорбления, ненормативная лексика и сообщения, нарушающие политику платформы.

Ручная модерация больших объёмов отзывов требует значительных человеческих ресурсов и времени, поэтому возникает потребность в автоматизированных решениях.

Системы автоматической модерации контента позволяют фильтровать отзывы в реальном времени и значительно повышают эффективность платформ электронной коммерции, социальных сетей и служб поддержки клиентов.

В рамках данного проекта разработана информационная система модерации отзывов, использующая архитектурные принципы микросервисов и технологии машинного обучения (ML). Система анализирует текстовые отзывы пользователей, определяет их категорию и формирует рекомендацию для модератора (допустить, проверить вручную или заблокировать).

Цель и задачи работы

Цель работы — спроектировать и реализовать архитектуру вычислительной системы, обеспечивающей автоматическую модерацию текстовых отзывов, и продемонстрировать её функционирование на тестовых данных.

Для достижения цели необходимо решить следующие задачи:

1. Проанализировать требования к системе автоматической модерации контента.

2. Разработать архитектуру информационной системы и определить ключевые компоненты.

3. Создать ML-модель на Python, обученную на текстовых данных для классификации отзывов.

4. Реализовать REST API на Java (Spring Boot) для приёма и обработки отзывов.

5. Настроить взаимодействие микросервисов через Docker.

6. Разработать UML-диаграммы, описывающие структуру, взаимодействие и жизненный цикл компонентов системы.

7. Провести тестирование системы на наборе русскоязычных примеров отзывов.

1. Постановка задачи

1.1 Входные данные

Текстовые отзывы пользователей на русском языке

Формат входных данных: JSON-объект с одним обязательным полем review и необязательными идентификаторами (reviewId, userId)

1.2 Выходные данные

JSON-ответ, содержащий:

1) исходный текст

2) присвоенную категорию (положительный, отрицательный, нейтральный, спам, оскорбительный и т.п.)

3) рекомендацию для модератора: ALLOW, MANUAL_REVIEW или REJECT

4) уровень уверенности (confidence score)

1.3 Функциональные требования

1. Приём запроса от пользователя

2. Валидация входных данных

3. Вызов внешнего ML-сервиса для анализа текста

4. Получение и обработка результата классификации

5. Формирование и возврат ответа пользователю

1.4 Нефункциональные требования

1. Система должна быть модульной и расширяемой

2. Должна поддерживаться контейнеризация для простого развёртывания

3. Интерфейс взаимодействия — REST API, формат обмена данными — JSON

2. Архитектура системы

Система реализована по микросервисному принципу: основное приложение на Java Spring Boot взаимодействует с моделью машинного обучения, развернутой в Docker-контейнере на Python.

Основные компоненты:

ReviewController — REST-контроллер для обработки запросов;

ReviewService — бизнес-логика и взаимодействие с моделью;

ModelClient — HTTP-клиент для общения с ML-сервисом;

GlobalExceptionHandler — обработка ошибок валидации и выполнения;

Config — файл application.yaml, определяющий параметры системы;

Python Model Service — ML-модель, обученная с использованием CatBoost, принимающая текст и возвращающая результат в JSON-формате.

Docker используется для изоляции среды выполнения и обеспечения совместимости между платформами.

3. UML-диаграммы

3.1 Диаграмма вариантов использования

Диаграмма представлена на рисунке 3.1.

Описание работы диаграммы:

1. Пользователь (модератор) отправляет текст отзыва для анализа.

2. Система обрабатывает запрос, обращается к ML-модели, получает от неё категорию и возвращает пользователю итоговый результат.

3. В случае ошибок валидации или недоступности модели активируется сценарий обработки ошибок.

Рисунок 3.1 – Диаграмма вариантов использования

3.2 Диаграмма классов

Диаграмма представлена на рисунке 3.2.

Описание работы диаграммы:

1. ReviewController принимает входные запросы и вызывает ReviewService, который через ModelClient обращается к ML-модели.

2. Модель возвращает результат классификации, который упаковывается в ReviewResponseDTO.

3. GlobalExceptionHandler отвечает за корректную обработку исключений.

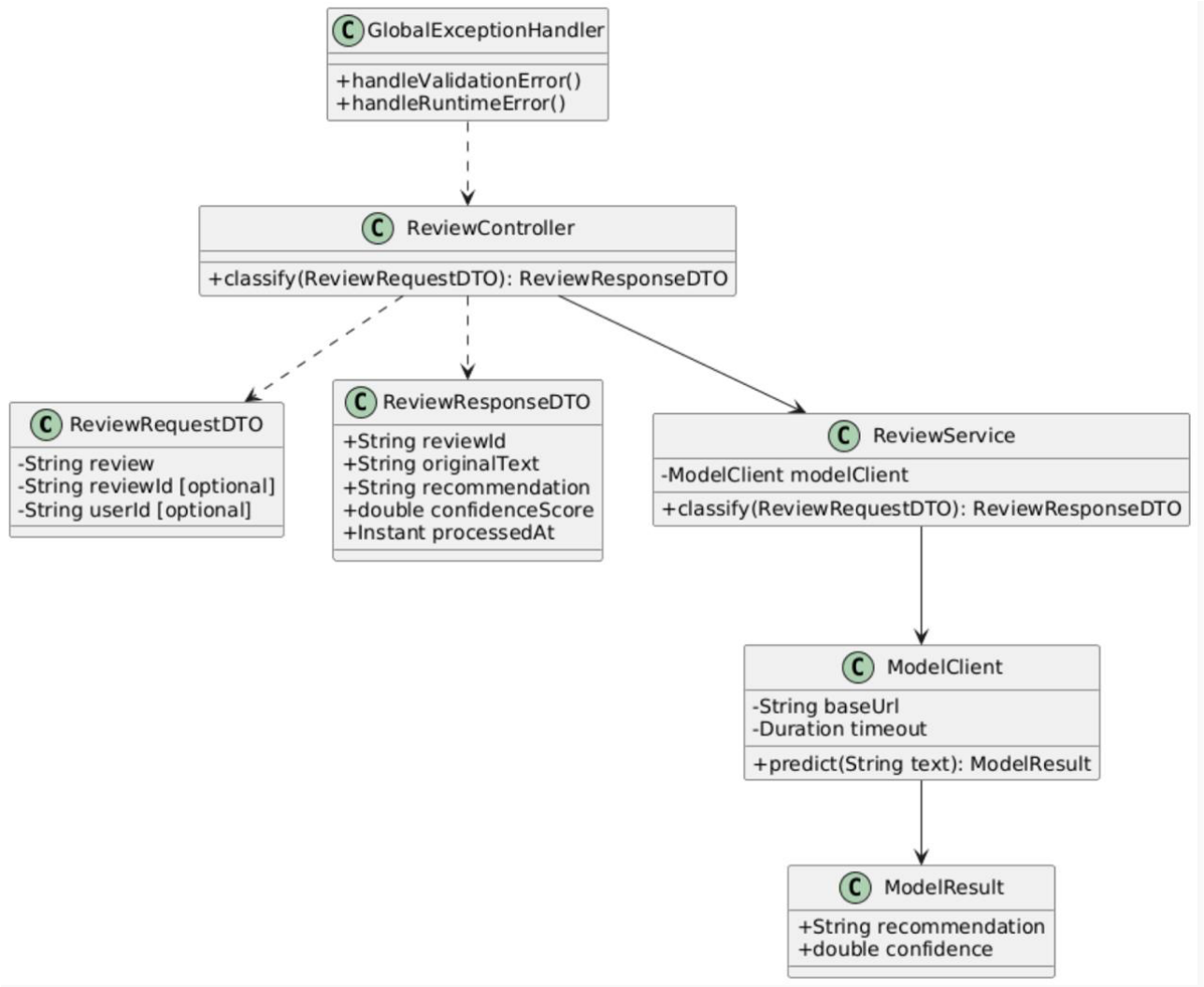

Рисунок 3.2 – Диаграмма классов системы Review Moderation System

3.3 Диаграмма состояний

Диаграмма представлена на рисунке 3.3.

Описание работы диаграммы:

1. Система получает запрос на классификацию и переходит в состояние проверки валидации.

2. Если данные некорректны — выполняется переход в состояние ошибки валидации.

3. При успешной проверке система отправляет текст модели, ожидает результат и формирует ответ пользователю.

4. Если модель не отвечает — переход в состояние ошибки модели.

5. После успешной обработки система возвращает результат и завершает цикл.

Рисунок 3.3 – Диаграмма состояний обработки запроса

3.4 Диаграмма последовательности (дополнительная)

Несмотря на то, что диаграмма последовательности не является обязательной частью курсового проекта, было решено добавить её для более наглядного описания взаимодействия компонентов системы в процессе классификации отзыва.

Диаграмма, представленная на рисунке 3.4, демонстрирует порядок вызовов между основными модулями системы: пользователем, контроллером, сервисом, клиентом и моделью, развернутой в отдельном Docker-контейнере.

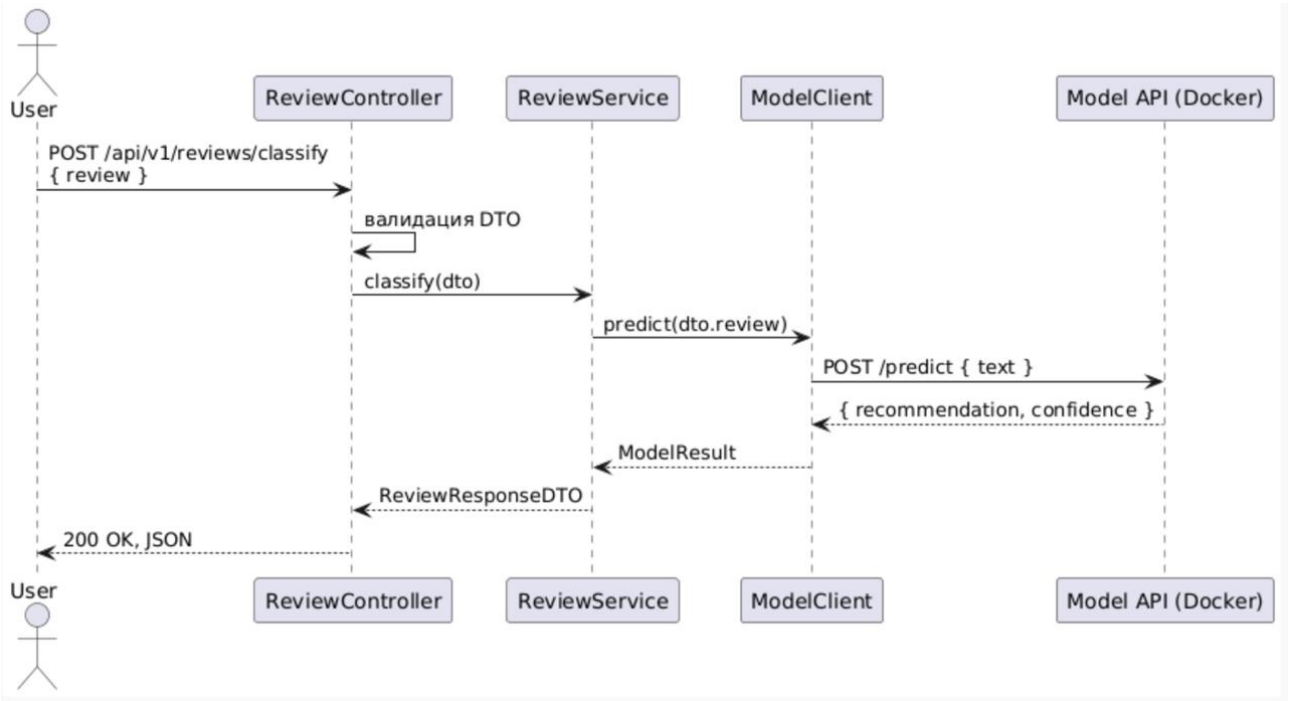

Рисунок 3.4 – Диаграмма последовательности обработки запроса классификации

Описание работы диаграммы:

1. Пользователь (User) отправляет POST-запрос на адрес /api/v1/reviews/classify, передавая JSON-объект с текстом отзыва.

2. Контроллер (ReviewController) принимает запрос, выполняет проверку DTO и вызывает метод classify(dto) у сервиса.

3. Сервис (ReviewService) обращается к клиенту модели (ModelClient), вызывая метод predict(dto.review).

4. Клиент модели (ModelClient) отправляет REST-запрос POST /predict в ML-сервис, развернутый на порту 8000, передавая текст отзыва.

5. ML-сервис (Model API) обрабатывает текст, классифицирует его с помощью модели CatBoost и возвращает JSON-ответ с полями recommendation и confidence.

6. Сервис получает объект ModelResult, формирует итоговый DTO (ReviewResponseDTO) и передаёт его контроллеру.

7. Контроллер возвращает пользователю результат с HTTP-кодом 200 OK и телом JSON-ответа.

4. Развёртывание и тестирование

4.1 Среда выполнения

macOS Sonoma, процессор Apple M4

Docker Desktop

Java 21 JRE

Python 3.10 + CatBoost

4.2 Запуск системы

Система запускается через Docker Compose, что позволяет автоматически собрать и развернуть оба сервиса (основное приложение и ML-модель) одной командой.

```
docker compose app build
docker compose up
```

После выполнения команды app build оба контейнера создаются и запускаются автоматически:

порт 8080 — REST API (Spring Boot оболочка)

порт 8000 — сервис с ML-моделью на Python (CatBoost)

Таким образом, тестировать работу системы можно сразу после сборки, без дополнительной настройки.

4.3 Пример тестирования API

Запрос:

```
curl -X POST http://localhost:8080/api/v1/reviews/classify \
  -H "Content-Type: application/json" \
  -d '{"review": "Товар не понравился, ожидал большего качества."}'
```

Ответ:

```
{
  "reviewId": "3f10cb6f-08de-4aef-8bb0-8c4cf08aec8f",
  "originalText": "Товар не понравился, ожидал большего качества.",
  "recommendation": "MANUAL_REVIEW",
  "confidenceScore": 0.80,
  "processedAt": "2025-10-18T20:37:16Z"
}
```

4.4 Набор тестов

№	Пример отзыва	Категория	Рекомендация	Описание
1	Отличный сервис! Всё	Положительный	Допущен	Отзыв без нарушений

	быстро, удобно и вежливо.			
2	Товар не понравился, ожидал большего качества.	Отрицательный	Проверить вручную	Негативная, но допустимая оценка
3	Продавец хам и обманщик!	Оскорбительный	Заблокировать	Нарушение правил общения
4	Зарабатывай по 1000₽ в день!	Спам	Заблокировать	Рекламное сообщение
5	Посылка пришла вовремя.	Нейтральный	Допущен	Нормальный отзыв
6	Сам продукт хороший, но доставка подвела.	Смешанный	Проверить вручную	Содержит положительные и отрицательные аспекты
7	(пустая строка)	Ошибка	—	Ошибка валидации (пустой ввод)

4.5 Результаты тестирования

Во время тестирования система показала стабильную работу и корректную классификацию.

Все запросы обрабатывались в среднем менее чем за 0.5 секунды.

Ошибки валидации и сетевые исключения корректно перехватываются глобальным обработчиком ошибок.

Docker-окружение обеспечило быстрое развёртывание и переносимость системы.

5. Распределение ролей в команде

Участник	Роль и ответственность
Мария Рымарь	Разработка UML-диаграмм, оформление отчёта, код-ревью, интеграция компонентов
Матвей Мелихов	Создание и обучение ML-модели на Python (CatBoost), подготовка данных
Владислав Придчин	Реализация backend-части на Java Spring Boot, настройка Docker и инфраструктуры запуска

Командная работа велась с использованием GitHub. Каждый участник вёл отдельные ветки, а слияние выполнялось через pull request.

На рисунке 5.1 приведён скриншот истории коммитов из программы Fork, отражающий участие всех членов команды.

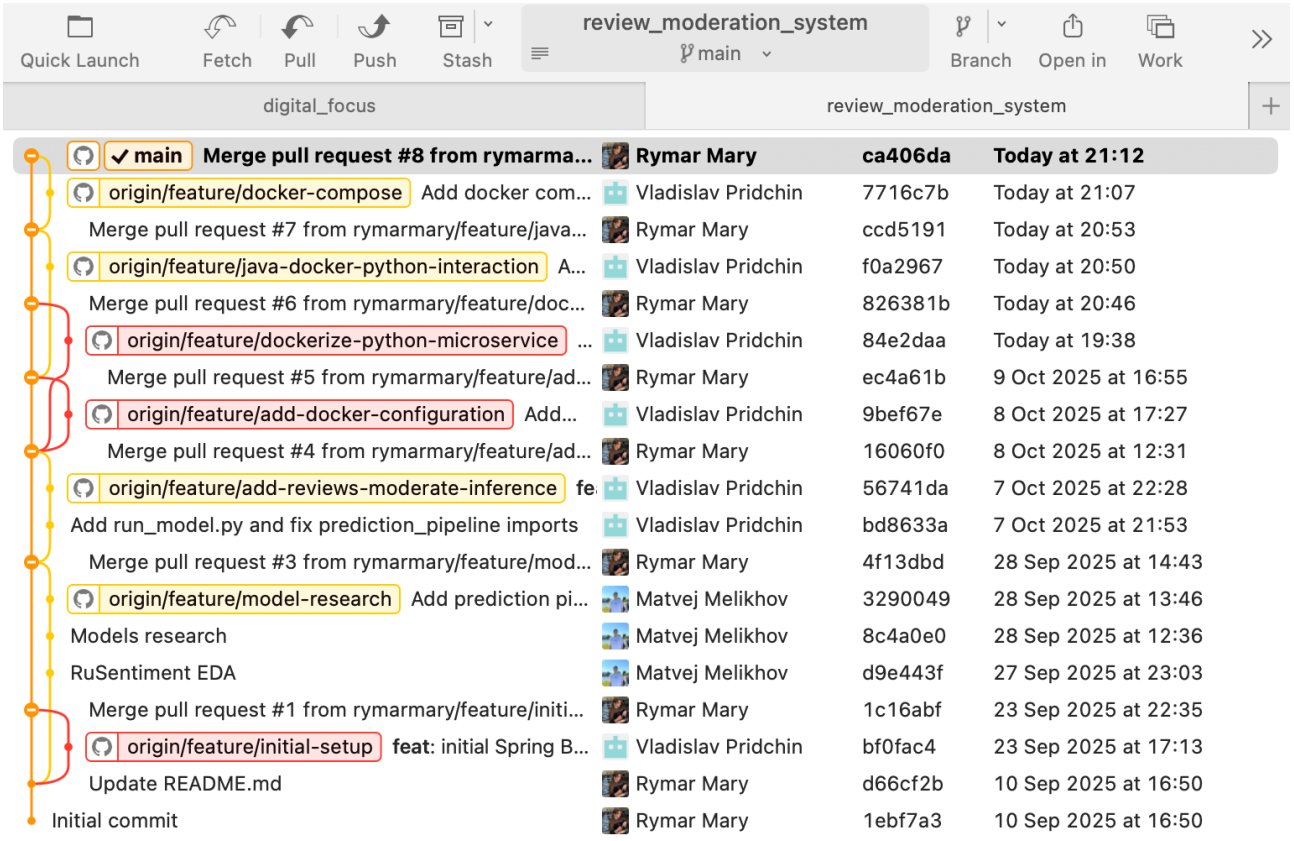

Рисунок 5.1 – История коммитов в GitHub

Репозиторий проекта:

https://github.com/rymarmary/review_moderation_system

Заключение

В ходе выполнения курсового проекта была спроектирована и реализована система Review Moderation System, предназначенная для автоматической модерации текстовых отзывов на русском языке.

Проект реализует принципы модульности, повторного использования компонентов и интеграции технологий машинного обучения в промышленную архитектуру.

Использование микросервисного подхода позволило разделить ответственность между ML-моделью и backend-приложением, обеспечив гибкость и масштабируемость.

Система успешно прошла тестирование и корректно классифицировала различные типы отзывов — положительные, отрицательные, оскорбительные и спам-сообщения.

Реализация с использованием Docker обеспечила воспроизводимость и упрощённый процесс развёртывания.

Результаты работы демонстрируют эффективность выбранных решений и применимость подобного подхода для построения реальных систем контент-модерации.