Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

Разработка интерактивного Telegram-бота «Country Quiz» на языке Java

по дисциплине «Конструирование программного обеспечения»

Выполнил
студенты гр. 5130904/30102 Иващенко А. В.
Аристов Е. А

Руководитель Иванов А. С.

Санкт-Петербург
2025

Оглавление

Введение 3
Определение проблемы 4
Выработка требований (пользовательские сценарии) 5
Нефункциональные требования 5
Разработка архитектуры и детальное проектирование 6
Оценка нагрузки 6
Диаграммы C4 6
Нефункциональные требования 7
Схема базы данных 8
Схема масштабирования при росте нагрузки в 10 раз 8
Кодирование и тестирование 9
Unit тестирование 10
Интеграционное тестировании 11
Сборка и запуск 12
Заключение 13
Список литературы 14

Введение

В современной образовательной среде геймификация является одним из наиболее эффективных методов удержания внимания. Данный проект представляет собой серверное приложение (Telegram-бот), которое позволяет пользователям изучать географию (флаги, столицы, данные о странах) в формате викторины.

Проект реализован на стеке Java 17 / Spring Boot 3. Для обеспечения персистентности данных используется PostgreSQL. Интеграция с внешними данными осуществляется через RestCountries API. Весь цикл разработки, от сборки до развертывания, автоматизирован с помощью Docker, что соответствует современным стандартам DevOps.

Определение проблемы

Проблема**:** Классические методы изучения географии (атласы, таблицы) характеризуются низким уровнем интерактивности. Существующие мобильные приложения часто требуют отдельной установки, регистрации и занимают место в памяти устройства.

Выработка требований (пользовательские сценарии)

Основные пользовательские сценарии работы с системой:

1. **Сценарий «Быстрый старт»:** Пользователь нажимает /start, бот приветствует его и предлагает выбрать режим игры (например, «Угадай по флагу»).
2. **Сценарий «Игровой цикл»:** Бот присылает изображение флага и 4 кнопки с вариантами ответов. При нажатии пользователь получает мгновенный фидбек (верно/неверно) и следующий вопрос.
3. **Сценарий «Статистика»:** Пользователь запрашивает команду /stats и получает отчет о количестве правильных ответов и общем месте в рейтинге.

Нефункциональные требования

Время отклика системы не должно превышать 1–2 секунд при стандартной нагрузке.
Сервис должен быть доступен 24/7 и корректно обрабатывать одновременные запросы от большого количества пользователей

Разработка архитектуры и детальное проектирование

Оценка нагрузки

Согласно заданию, система должна быть рассчитана на 10 000 пользователей в сутки.

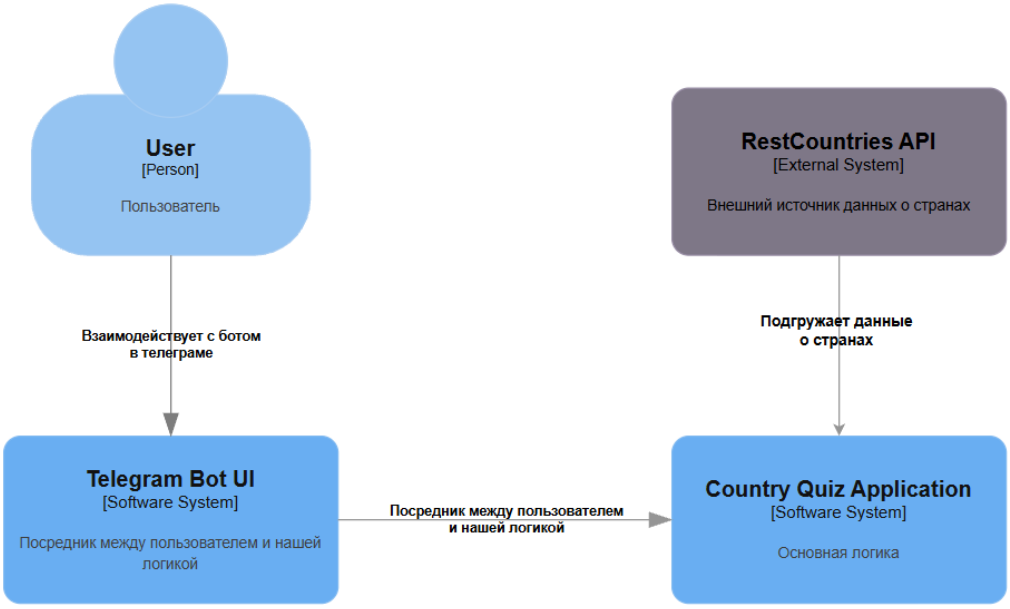
Трафик: Каждый пользователь делает в среднем 20 запросов (вопросов) в день. Средний размер JSON-ответа от Telegram API и RestCountry API составляет ~3 КБ. Расчет: $10000 \times 20 \times 3 \text{ КБ} \approx 600 \text{ КБ/сутки}$

Дисковое пространство: Запись о завершенной игре весит ~100 байт. Расчет за 5 лет: $10000 \text{ пользователей} \times 365 \text{ дней} \times 5 \text{ лет} \times 100 \text{ байт} \approx 1{,}8 \text{ ГБ}$. С учетом индексов, логов и метаданных пользователей — около 5–10 ГБ.

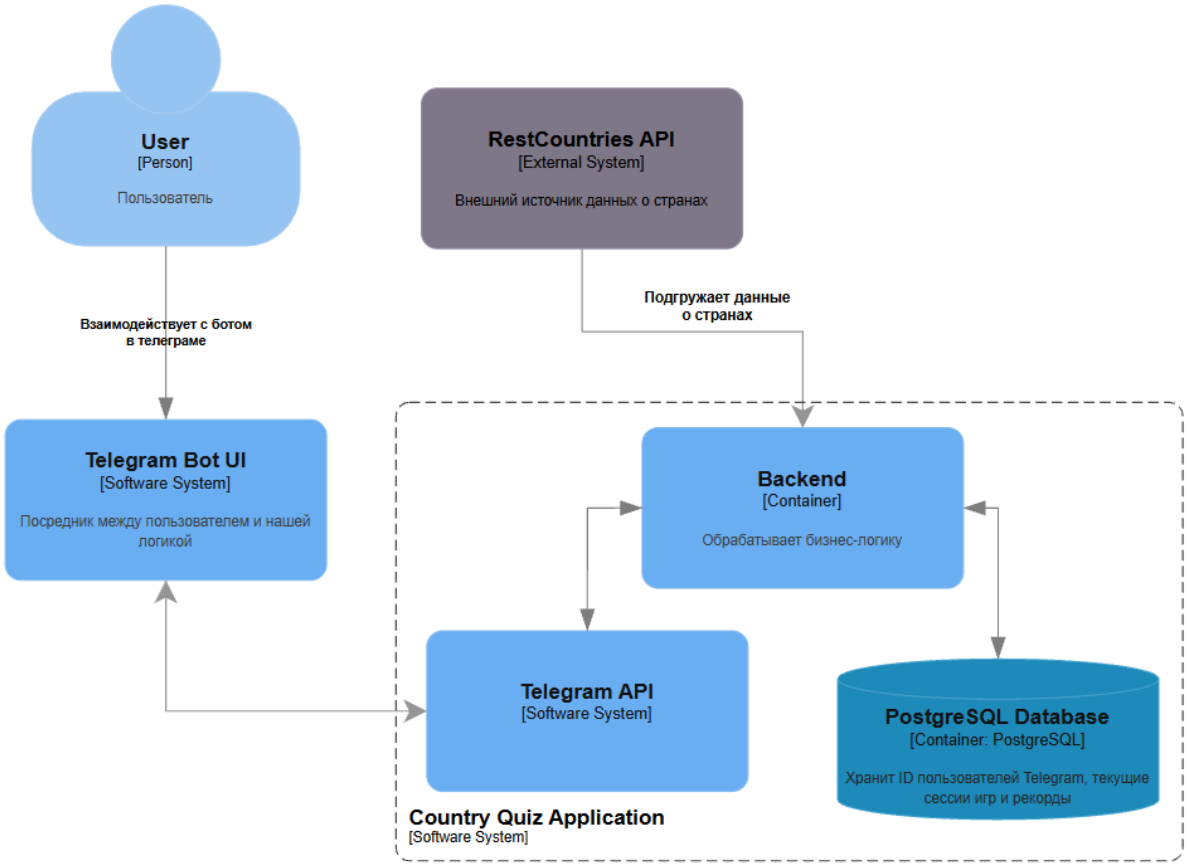
R/W нагрузка: Соотношение составляет 90/10. Чтение преобладает (запрос флагов, вариантов), запись происходит только при фиксации ответа или обновлении профиля.

Диаграммы C4

Уровень 1: System Context



Уровень 2: Container Diagram



Нефункциональные требования

Время отклика системы не должно превышать 1–2 секунд при стандартной нагрузке. Сервис должен быть доступен 24/7 и корректно обрабатывать одновременные запросы от большого количества пользователей.

Схема базы данных

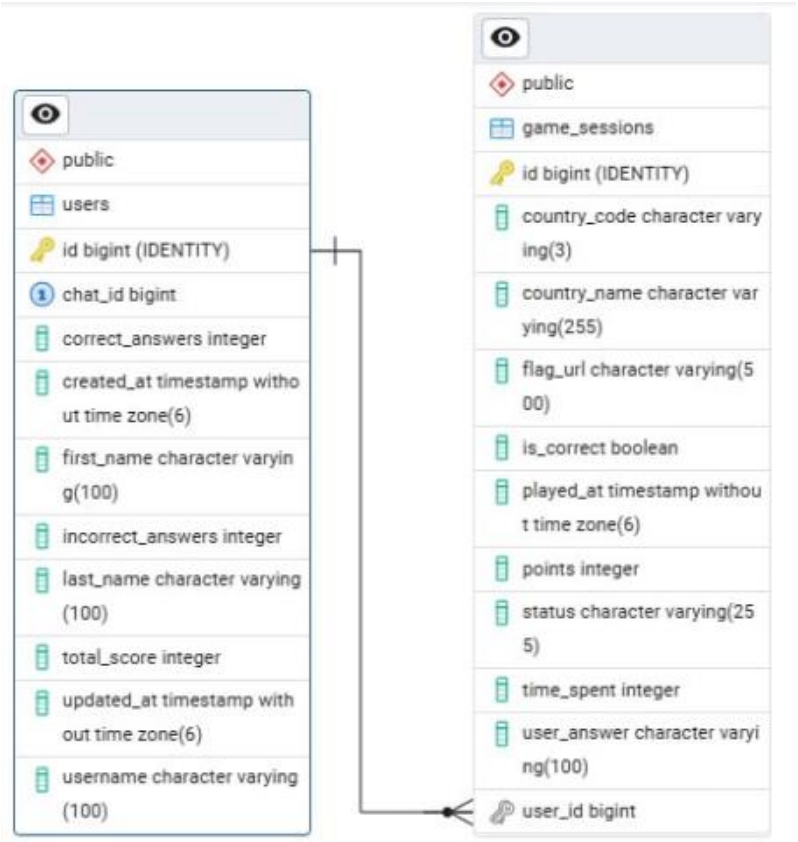


Схема масштабирования при росте нагрузки в 10 раз

При росте нагрузки в 10 раз (до 100к пользователей):

1. Stateless-подход: Приложение не хранит состояние сессии в памяти (все в БД), что позволяет запустить N экземпляров приложения за балансировщиком (Round Robin).
2. Репликация БД: Выделение Master-узла на запись и Slave-узлов на чтение статистики.

Кодирование и тестирование

В процессе разработки каждый участник команды вносил изменения в кодовую базу проекта, что подтверждается историей коммитов в репозитории.

В рамках разработки Telegram-бота-игры «Угадай страну» было реализовано модульное тестирование, направленное на проверку корректности работы основной бизнес-логики приложения. Тестирование позволяет выявлять ошибки на ранних этапах разработки и обеспечивает стабильность работы системы при дальнейшем расширении функциональности. Для тестирования использовались фреймворк JUnit 5 и библиотека Mockito, позволяющая изолировать тестируемые компоненты от внешних зависимостей. Автоматический запуск тестов осуществляется с помощью плагина Maven Surefire при выполнении команды mvn test.

Unit тестирование

В проекте реализованы 8 unit-тестов, разделённых на 3 файла.

Тесты сервиса работы с данными стран проверяют корректность получения игровой информации и работу резервного механизма при отсутствии данных. Это гарантирует стабильную работу игры даже при недоступности внешних источников данных.

Unit-тесты сервиса игровой логики проверяют создание новой игровой сессии, обработку ответов пользователя, а также корректное начисление и списание очков. Таким образом обеспечивается корректная работа основного игрового процесса.

Тестирование сервиса управления пользователями направлено на проверку создания новых пользователей и обновления их игровой статистики, включая общее количество очков и точность ответов.

Интеграционное тестировании

Стабильность взаимодействия компонентов системы проверяется с помощью интеграционных тестов на базе технологии Testcontainers. Это позволяет в процессе тестирования автоматически развертывать реальную базу данных PostgreSQL в Docker-контейнере, обеспечивая полную идентичность тестовой и промышленной сред.

Тестовые сценарии имитируют полный цикл взаимодействия пользователя с ботом: от автоматического создания профиля при первом обращении до проведения серии игровых сессий. В ходе тестов проверяется корректность обработки правильных и неверных ответов, работа алгоритмов начисления баллов и автоматический пересчет агрегированной статистики (точность, общий счет). Кроме того, тестируется функциональность глобального рейтинга, что подтверждает надежность выполнения сложных SQL-запросов и стабильность связи приложения с базой данных под нагрузкой.

Сборка и запуск

Проект полностью контейнеризирован с использованием Docker. Для запуска приложения, unit-тестов и интеграционных тестов достаточно одной команды, при этом на системе требуется только установленный Docker и bash-совместимая оболочка.

Запуск приложения: docker-compose up –build

Сборка и тесты: mvn clean install

Заключение

Разработанная система полностью соответствует поставленным задачам. Использование Spring Boot позволило создать расширяемую архитектуру, а Docker гарантирует идентичность среды разработки и эксплуатации. Система готова к масштабированию и длительному хранению данных (свыше 5 лет).

Список литературы

- Документация Spring Boot (docs.spring.io).
- Telegram Bot API Documentation (core.telegram.org/bots/api).
- Саймон Браун. «C4 model for visualizing software architecture».
- К. Дж. Дейт. «Введение в системы баз данных».