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

Курсовой проект

Тема проекта: "Telegram-бот для уведомлений о погоде"

(https://github.com/marlions/weather-telegram-bot)

по дисциплине «Конструирование ПО»

Выполнил
студент гр. 5130904/30104 Маришин Л.В.

Руководитель Иванов А.С.

Содержание

Определение проблемы

Пользователям неудобно регулярно вручную проверять погоду и прогноз в нужном городе, особенно если прогноз требуется каждый день в определённое время. Из-за этого люди пропускают резкие изменения погодных условий (мороз, штормовой ветер и т.п.) и тратят время на повторяющиеся действия. Требуется простой интерфейс с уведомлениями и хранением пользовательских настроек.

Выработка требований

Пользовательские истории:

История 1 (получение текущей погоды):
Когда я хочу быстро узнать текущую погоду в моём городе, я пишу команду/нажимаю кнопку, чтобы получить температуру, влажность, ветер и описание погодных условий.

История 2 (прогноз на несколько дней и выбор дня):
Когда я планирую ближайшие дни, я хочу получить прогноз на 5 дней и/или выбрать конкретный день (1–5), чтобы увидеть ожидаемые условия.

История 3 (ежедневные уведомления и предупреждения):
Когда мне нужен прогноз каждый день, я хочу подписаться на ежедневную рассылку в удобное время и получать предупреждения при опасной погоде, чтобы не пропустить критичные условия.

Функциональные требования

- Регистрация пользователя в системе (через команду /start).
- Установка города по умолчанию (например, /set_city <город>).
- Получение текущей погоды для выбранного города.
- Получение прогноза на 5 дней.
- Получение прогноза на выбранный день (1–5).
- Подписка на ежедневные уведомления.
- Выбор времени ежедневных уведомлений (в формате ЧЧ:ММ; в текущей реализации ориентир на UTC).
- Отписка от уведомлений.
- Выдача подсказки/справки по командам.
- Отправка предупреждений при экстремальной погоде (например, сильный мороз, штормовой ветер и т.п.).

Нефункциональные требования

Производительность и время отклика

- Ответ на команды бота: целевое время формирования ответа **до 1–2 секунд** при доступности внешнего API; допустимо увеличение при проблемах сети.
- Таймаут запросов к внешнему API: ограниченный (например, 10 секунд), после чего пользователю возвращается ошибка “не удалось получить данные”.

Надёжность

- Корректная обработка сетевых ошибок и ошибок внешнего API (не падать целиком, а отвечать понятным сообщением).
- Логирование ключевых действий и ошибок.

Хранение и целостность

- Настройки пользователя (telegram_id, город, подписка, время уведомлений) должны храниться в БД и сохраняться между перезапусками.

Безопасность

- Токен Telegram-бота и ключ внешнего API не должны храниться в коде, только в переменных окружения/конфиге.
- Доступ к БД по паролю, секреты не коммитятся в репозиторий.

Оценка нагрузки и периода хранения данных

Примем оценку:

- **DAU:** 10 000 пользователей/сутки (как рекомендуемый порядок).
- **Всего зарегистрированных:** 100 000 пользователей.
- **Средняя активность:** 3 запроса погоды/сутки на активного пользователя.
- **Подписка на ежедневные уведомления:** 30% пользователей (30 000).
- **Период хранения:** 5+ лет (настройки пользователя храним постоянно, историю погоды не храним, чтобы не раздувать БД).

Архитектура и детальное проектирование

Требуется описать: характер нагрузки, две диаграммы C4 (Context + Container), контракты API и ожидаемые времена отклика, схему БД и обоснование, а также масштабирование при росте нагрузки в 10 раз.

3.1 Характер нагрузки (R/W, трафик, диск)

Соотношение R/W (чтение/запись)

- Запись (W): редкая. Установка города/подписки/времени уведомлений.
  Оценка: 1 запись на пользователя раз в несколько дней.
- Чтение (R): частая. Получение настроек пользователя, выборка подписок на рассылку, чтение данных для отправки уведомлений.

Итого: **нагрузка существенно read-heavy** по отношению к БД. Основной внешний трафик уходит во внешнее погодное API.

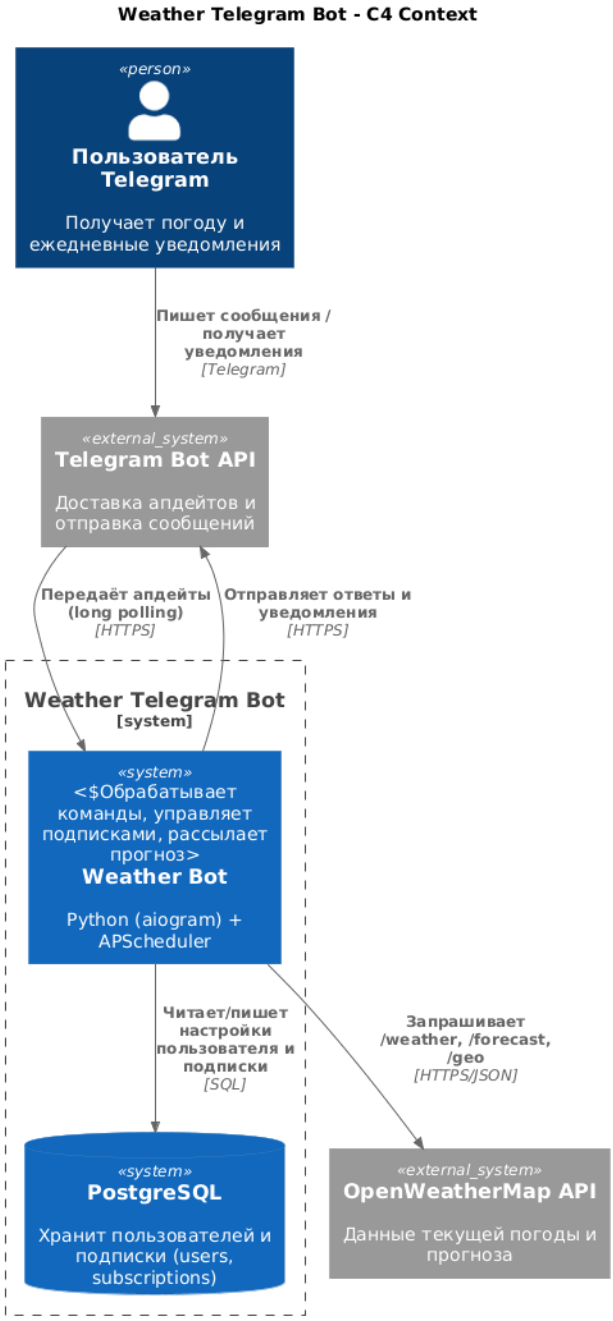
Трафик

- Запросы к внешнему API:
  - Режим “по запросу”: до 10k * 3 = **30k** запросов/сутки (в идеале меньше за счёт кеширования по городу).
  - Режим “ежедневная рассылка”: при группировке по городу можно делать 1 запрос на город и разослать многим пользователям.
- Трафик Telegram: исходящие сообщения (ответы + рассылки).

Диск

- База данных: хранит только пользователей и их настройки, объём небольшой (десятки/сотни МБ даже при росте).
- Логи: могут быть заметнее БД, поэтому нужна ротация логов.

Диаграмма C4: Context (описание)



Система: Weather Telegram Bot.

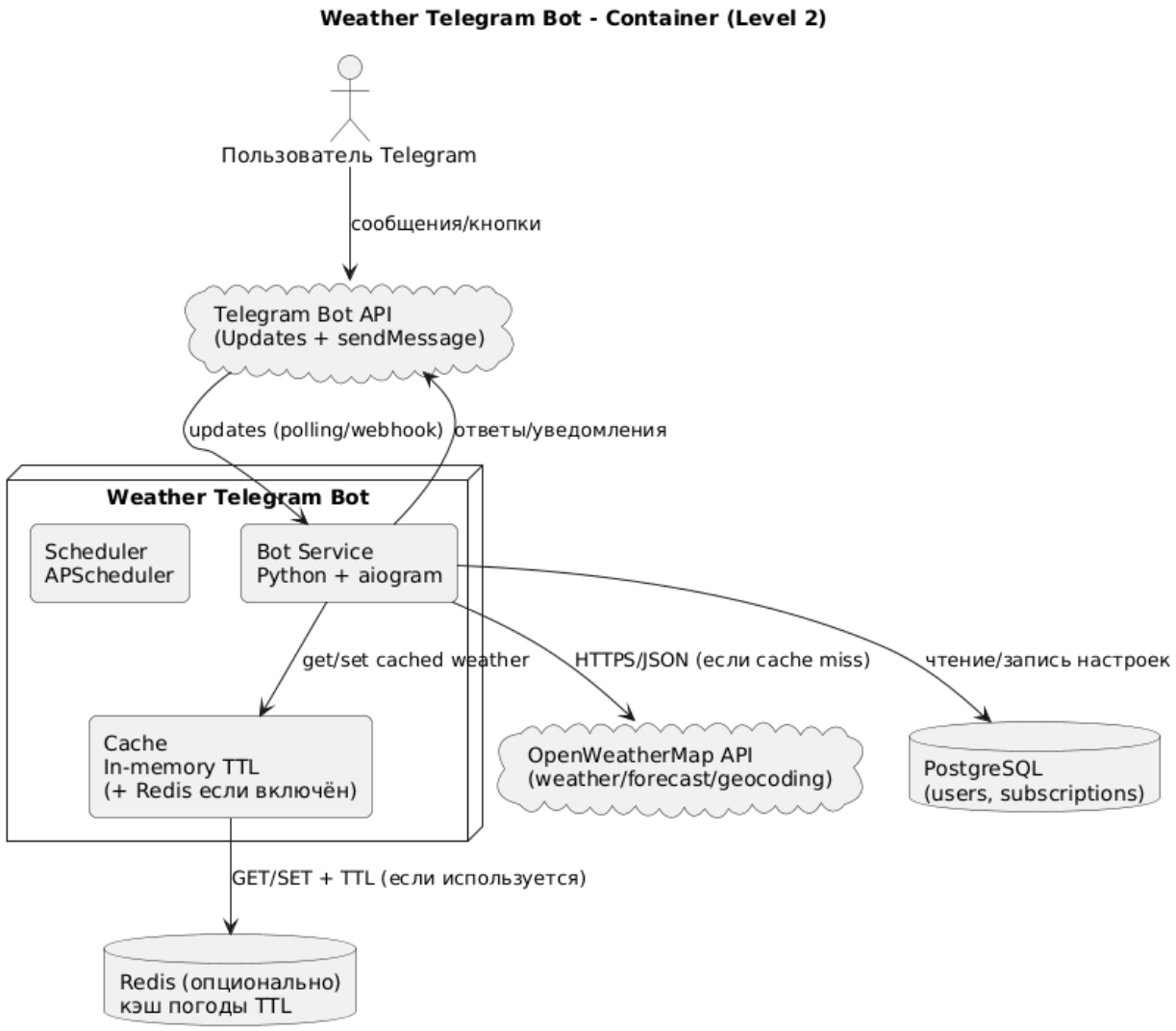
Акторы и внешние системы

- Пользователь Telegram (человек) взаимодействует с ботом через чат.
- Telegram API доставляет сообщения боту и принимает ответы.
- Внешний поставщик данных: OpenWeatherMap API (текущая погода, прогноз).
- База данных PostgreSQL хранит пользователей и подписки.

Связи:

- Пользователь ↔ Telegram ↔ Бот: команды/кнопки/ответы/уведомления.
- Бот ↔ OpenWeatherMap: запросы данных о погоде.
- Бот ↔ PostgreSQL: хранение и чтение настроек пользователей.

Диаграмма C4: Container (описание)



Контейнеры:

1. **Bot Application (Python, aiogram)**
   - Обрабатывает входящие команды/сообщения.
   - Запрашивает погоду во внешнем API.
   - Отправляет сообщения пользователю.
   - Запускает планировщик задач (ежедневная рассылка).
2. **PostgreSQL**
   - Таблицы пользователей и подписок.
   - Индексы по telegram_id и/или user_id.
3. **Cache/Redis**
   - Кеширование погоды по городу на короткий TTL (например, 5 минут) для снижения нагрузки на внешнее API и ускорения ответов.

Контракты API и ожидания по времени отклика

Внешний API погоды (OpenWeatherMap)

1) Текущая погода

- Endpoint: /data/2.5/weather
- Параметры:
  - q – город (например, London)
  - appid – ключ API
  - units=metric – градусы Цельсия
  - lang=ru – русский язык описаний
- Ответ (важные поля, которые используются в боте):
  - main.temp, main.feels_like, main.humidity
  - wind.speed
  - weather[0].description

2) Прогноз

- Endpoint: /data/2.5/forecast
- Параметры те же (q, appid, units, lang)
- Ответ: массив прогнозов по времени, из которого формируется прогноз на 5 дней (агрегация/выборка).

Ошибки/исключения (как обрабатывается):

- 401/403 — неверный ключ API → бот сообщает пользователю об ошибке сервиса.
- 404 — город не найден → бот просит уточнить название.
- Timeout/сеть — бот сообщает “не удалось получить данные, попробуйте позже”.

Ожидаемое время отклика:

- При нормальной сети: 200–800 мс на запрос к OpenWeatherMap + обработка.
- В ответе бота целимся в **1–2 секунды** суммарно; при проблемах сети показываем ошибку.

Telegram Bot API

Используется через библиотеку (aiogram): приём апдейтов и отправка сообщений.

Ожидаемое время отклика:

- Отправка сообщения: обычно < 1 секунды.

Схема базы данных и обоснование

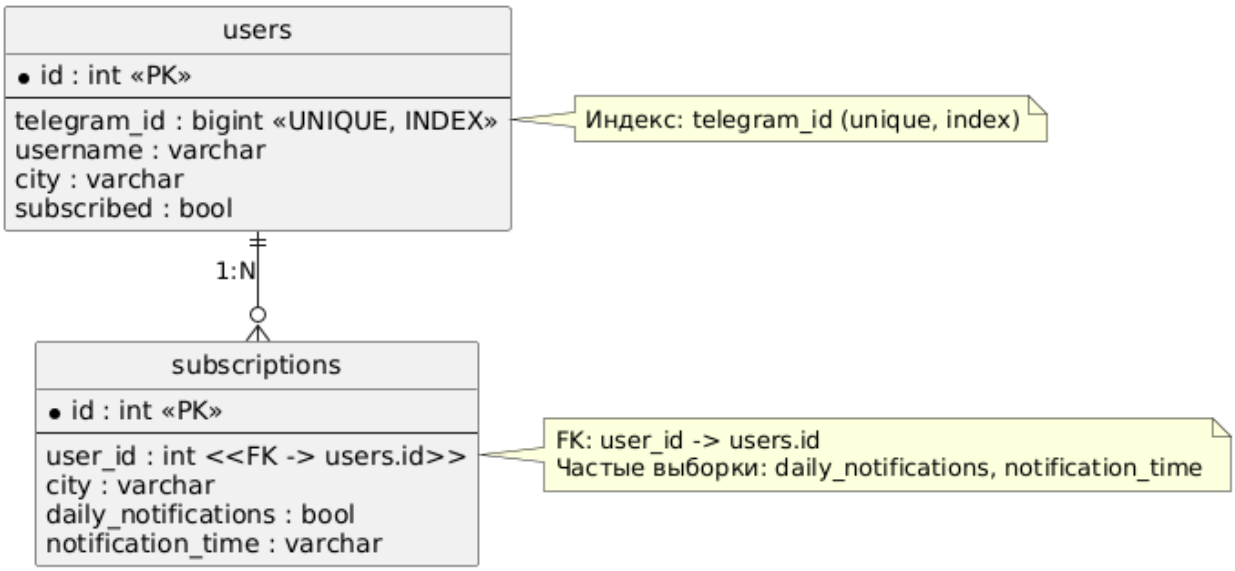


Таблица users

- id (PK)
- telegram_id (уникальный идентификатор пользователя Telegram, UNIQUE INDEX)
- city (город по умолчанию, nullable)
- created_at, updated_at

Таблица subscriptions

- id (PK)
- user_id (FK → users.id, INDEX)
- daily_notifications (bool)
- notification_time (строка “HH:MM” или time)
- (опционально) city (если хотим подписку на отдельный город)

Почему выдержит нефункциональные требования

- Данные маленькие, запросы простые (по индексу telegram_id и join по user_id).
- Основные операции: получить пользователя, сохранить настройки, выбрать подписчиков на конкретное время.
- Индексы покрывают самые частые условия выборки.

Масштабирование при росте нагрузки в 10 раз

При росте до **100k DAU** и кратного увеличения рассылок:

1. **Горизонтальное масштабирование приложения**
    - Запуск нескольких экземпляров бота (stateless) за балансировщиком.
    - Перейти на webhook-режим (если нужно), чтобы несколько инстансов могли принимать трафик корректно.
2. **Выделение рассылки в отдельный воркер**
    - Чтобы не мешать обработке команд, ежедневные уведомления выполнять отдельным процессом/контейнером.
3. **Кеширование**
    - Redis для кеша погоды по городу (TTL 2–10 минут). Это резко снижает число запросов к OpenWeatherMap.
4. **Очередь сообщений (опционально)**
    - Для массовых рассылок: класть задания “send message” в очередь (RabbitMQ/Kafka/SQS), воркеры отправляют сообщения дозировано.
5. **PostgreSQL**
    - На начальном этапе хватит vertical scale (CPU/RAM).
    - При росте: реплика на чтение (read replica), настройка пула соединений.

Кодирование и отладка

Проект реализован на Python и размещён в публичном репозитории GitHub. UI реализован через Telegram-бота, данные пользователей хранятся в PostgreSQL, внешняя зависимость: погодный API, запуск и тестирование автоматизируются через Docker и команды make (требование курса).

Структура репозитория (верхний уровень):

- app/
- handlers/
- services/
- tests/
- scripts/
- Dockerfile, docker-compose.yml, Makefile, README.md (описание запуска).

В README перечислены основные команды бота (/start, /set_city, /subscribe_daily и т.д.) и быстрые команды make для сборки/запуска/тестов.

Отладка:

- логирование (ошибки запросов к внешнему API, ошибки отправки сообщений, ошибки БД);
- обработка исключений, чтобы бот не завершался при частичных сбоях.

Unit-тестирование

В проекте предусмотрены unit-тесты (pytest) для проверки изолированных частей логики:

- форматирование/обработка погодных данных;
- логика определения “экстремальной погоды” и генерации предупреждений;
- функции кеширования/клиента (по возможности через мок запросов).

Интеграционное тестирование

Интеграционный сценарий должен покрывать хотя бы одну пользовательскую историю end-to-end.

Course

Пример сценария (История 3):

1. пользователь регистрируется (/start)
2. задаёт город (/set_city)
3. подписывается на ежедневные уведомления
4. выставляет время уведомлений
5. запускается задача рассылки и отправляет сообщение пользователю (в тесте: либо тестовая среда Telegram, либо имитация отправки через мок bot.send_message)

Результаты измерений

Замер задержки OpenWeather API (London):

- n=10: p50 ≈ 43.8 ms, p95 ≈ 187.8 ms
- n=30: p50 ≈ 44.6 ms, p95 ≈ 192.9 ms (max до ~530 ms)

База данных (тестовая нагрузка):

- users: 10 000
- subscriptions: 3 036

Производительность запроса выборки подписчиков (EXPLAIN ANALYZE):

- Execution Time порядка ~1 ms на указанных объёмах данных.

Сборка и запуск (Docker)

По требованиям курса, сборка/запуск/тесты должны выполняться “в одну команду” и опираться на Docker .

В проекте это решается через Makefile и docker-compose:

- make start (быстрый старт проекта)
- make build, make up, make down, make logs, make test, make integration, make lint, make format и др.

В отчёте можно оформить так:

Запуск приложения:

- make start

Unit-тесты:

- make test

Интеграционные тесты:

- make integration

Сборка контейнеров:

- make build

Заключение

Разработан индивидуальный проект “weather-telegram-bot”: Telegram-бот для получения текущей погоды и прогноза, а также для ежедневных уведомлений с настраиваемым временем. Проект использует внешнее погодное API, хранит пользовательские настройки в PostgreSQL и поставляется в Docker-окружении с автоматизированным запуском и тестированием через make. Архитектура допускает масштабирование за счёт горизонтального запуска приложения, кеширования и выделения рассылки в отдельный воркер.

Источники

1. Документация Telegram Bot API (общая справка по боту и сообщениям).
2. Документация OpenWeatherMap API (эндпоинты текущей погоды и прогноза).
3. Документация aiogram (фреймворк Telegram-бота на Python).
4. Документация SQLAlchemy + asyncpg (работа с PostgreSQL).
5. Документация Docker / docker-compose.
6. Документация pytest (тестирование).