Telegram Bot Architecture

Production-Ready Стартап на Rust

Rust Tokio Teloxide PostgreSQL SQLx

Railway

Версия 1.0

Архитектурная документация

📚 Содержание

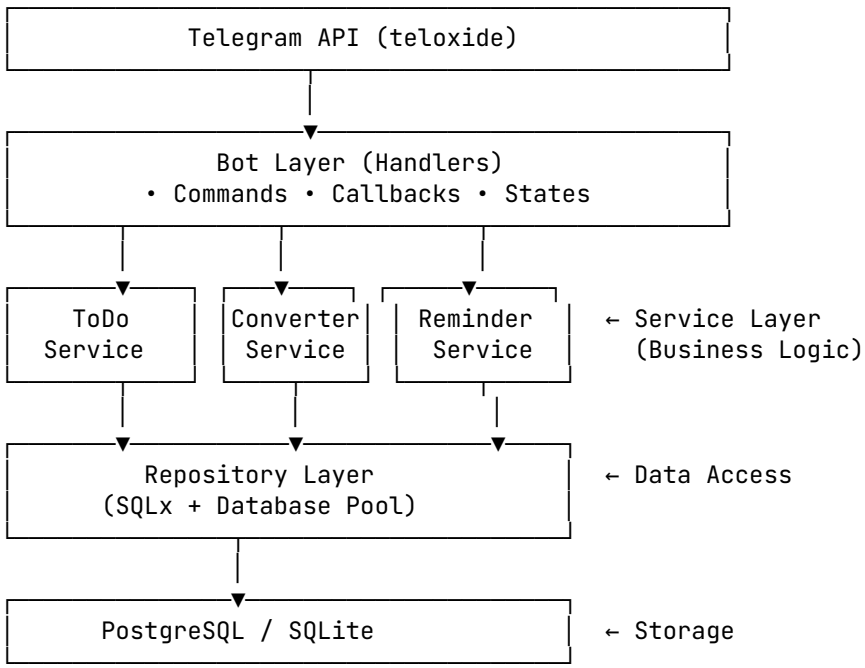
📁Структура проекта

Проект организован по принципу модульной архитектуры с четким разделением ответственности между компонентами.

```
telegram-multitool-bot/ ├── Cargo.toml ├── .env.example ├── .gitignore ├── README.md ├── Dockerfile ├── railway.toml ├── migrations/ │ ├── 001_init_schema.sql │ ├── 002_create_users.sql │ ├── 003_create_todos.sql │ ├── 004_create_reminders.sql │ └── 005_create_file_jobs.sql ├── config/ │ └── default.toml ├── src/ │ ├── main.rs # Entry point │ ├── lib.rs # Public API │ ├── config.rs # Configuration │ ├── error.rs # Error handling │ ├── bot/ │ │ ├── mod.rs │ │ ├── handlers.rs │ │ ├── commands.rs │ │ ├── callbacks.rs │ │ ├── keyboards.rs │ │ └── state.rs │ ├── todo/ │ │ ├── mod.rs │ │ ├── service.rs │ │ ├── models.rs │ │ ├── repository.rs │ │ └── handlers.rs │ ├── converter/ │ │ ├── mod.rs │ │ ├── service.rs │ │ ├── processors/ │ │ │ ├── mod.rs │ │ │ ├── image.rs │ │ │ ├── document.rs │ │ │ └── media.rs │ │ ├── queue.rs │ │ └── storage.rs │ ├── reminder/ │ │ ├── mod.rs │ │ ├── service.rs │ │ ├── scheduler.rs │ │ ├── models.rs │ │ ├── repository.rs │ │ └── notifier.rs │ ├── db/ │ │ ├── mod.rs │ │ ├── pool.rs │ │ ├── migrations.rs │ │ └── models.rs │ └── shared/ │ ├── mod.rs │ ├── types.rs │ ├── utils.rs │ └── telemetry.rs └── tests/ └── e2e.rs
```

🧩Архитектура и взаимодействия

Слоистая архитектура (Clean Architecture)



Поток данных

Принцип работы:

1. **Bot Layer** — точка входа, маршрутизация команд
2. **Service Layer** — бизнес-логика приложения
3. **Repository Layer** — абстракция над базой данных
4. **Scheduler Component** — фоновые задачи и напоминания

Компоненты системы

Компонент	Ответственность	Технологии
Bot Handler	Обработка команд и сообщений	Teloxide, FSM
ToDo Service	CRUD операции над задачами	SQLx, Repository pattern
Converter Service	Конвертация файлов	Image, PDF crates
Reminder Service	Планирование уведомлений	Tokio tasks, Chrono
Database Pool	Управление подключениями к БД	SQLx, PostgreSQL

🗃️База данных

Схема базы данных

```sql
 CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    telegram_id BIGINT UNIQUE NOT NULL,
    username VARCHAR(255),
    first_name VARCHAR(255),
    language_code VARCHAR(10) DEFAULT 'en',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_active_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE todos (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    title TEXT NOT NULL,
    description TEXT,
    status VARCHAR(20) DEFAULT 'pending',
    priority INTEGER DEFAULT 3,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP
);

CREATE TABLE reminders (
    id SERIAL PRIMARY KEY,
    todo_id INTEGER REFERENCES todos(id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    remind_at TIMESTAMP NOT NULL,
    message TEXT,
    is_sent BOOLEAN DEFAULT FALSE,
    sent_at TIMESTAMP,
    is_recurring BOOLEAN DEFAULT FALSE,
    recurrence_pattern VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE file_conversions (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    source_file_id VARCHAR(255),
    source_format VARCHAR(50),
    target_format VARCHAR(50),
    status VARCHAR(20),
    result_file_path TEXT,
    error_message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP
);

-- Индексы для производительности
CREATE INDEX idx_todos_user_status ON todos(user_id, status);
CREATE INDEX idx_reminders_remind_at ON reminders(remind_at)
    WHERE is_sent = FALSE;
CREATE INDEX idx_conversions_user_status ON file_conversions(user_id, status);
```

Зависимости

Cargo.toml

```
 [package]
name = "telegram-multitool-bot"
version = "0.1.0"
edition = "2021"

[dependencies]
# Async runtime
tokio = { version = "1.35", features = ["full"] }

# Telegram bot framework
teloxide = { version = "0.12", features = ["macros", "auto-send"] }

# Database
sqlx = { version = "0.7", features = [
    "runtime-tokio-rustls",
    "postgres",
    "sqlite",
    "chrono",
    "uuid"
] }

# Serialization
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
toml = "0.8"

# Date and time
chrono = { version = "0.4", features = ["serde"] }

# Error handling
thiserror = "1.0"
anyhow = "1.0"

# Logging
tracing = "0.1"
tracing-subscriber = { version = "0.3", features = ["env-filter"] }

# File processing (minimal set)
image = "0.24"  # Для работы с изображениями
pdf = "0.8"     # PDF обработка

# Utils
uuid = { version = "1.6", features = ["v4", "serde"] }
dotenv = "0.15"
once_cell = "1.19"

# Configuration
config = "0.13"

[dev-dependencies]
tokio-test = "0.4"
mockall = "0.12"
```

> **Принцип минимализма:** Используются только необходимые зависимости для обеспечения минимального размера бинарника и максимальной производительности.

🔧Система напоминаний

Архитектура планировщика

```
 // Концептуальная схема планировщика

Scheduler {
    // При запуске загружаем все активные напоминания
    load_pending_reminders() → Vec<Reminder>

    // Для каждого создаем Tokio task с delay
    for reminder in reminders {
        tokio::spawn(async move {
            tokio::time::sleep_until(reminder.remind_at).await;
            send_notification(reminder);
            mark_as_sent(reminder.id);

            // Если recurring, создаем следующее
            if reminder.is_recurring {
                schedule_next_occurrence(reminder);
            }
        });
    }

    // Background task проверяет новые напоминания каждые 30 сек
    tokio::spawn(async {
        loop {
            check_new_reminders();
            tokio::time::sleep(Duration::from_secs(30)).await;
        }
    });
}
```

Альтернативные подходы

Подход	Преимущества	Недостатки	Когда использовать
Tokio tasks	Простой, эффективный	Ограничен памятью	До 10K напоминаний