Оглавление

Тема проекта 2
Описание технологического стека 3
Выработка требований 5
Разработка архитектуры и детальное проектирование 8
Детализация нагрузки 8
Детализация объемов трафика и дискового хранилища 8
Объемы дискового хранилища 9
Пиковые нагрузки 9
Как система справляется с нагрузкой? 10
Углубленная C4-диаграмма 11
Контракты API с примерами 11
Схема БД с индексами 12
План масштабирования (x10) 12
Мониторинг и логи 13

Тема проекта:

Разработка системы интеллектуального управления лампой с функцией отображения изображений и поддержкой взаимодействия через веб-интерфейс по протоколу MQTT.

Устройства умного дома часто ограничены функционалом, трудны в настройке и требуют знаний для интеграции. Не все лампы обладают возможностью обмена данными через веб. Сложности в взаимодействии с цифровыми интерфейсами ограничивают их удобство и функционал.

Комплексное решение проекта включает:

Веб-интерфейс:

- Панель управления с кнопками для отображения различных изображений и состояний лампы.
- Передача команд на устройство через MQTT-протокол.
- Простота использования через любое современное устройство (ПК, планшет, смартфон).

Умное устройство (ESP8266 + TFT-дисплей):

- Подключение к Wi-Fi и подписка на MQTT-канал.
- Получение и визуализация эмоций/изображений по команде.
- Управление через сенсорную кнопку.

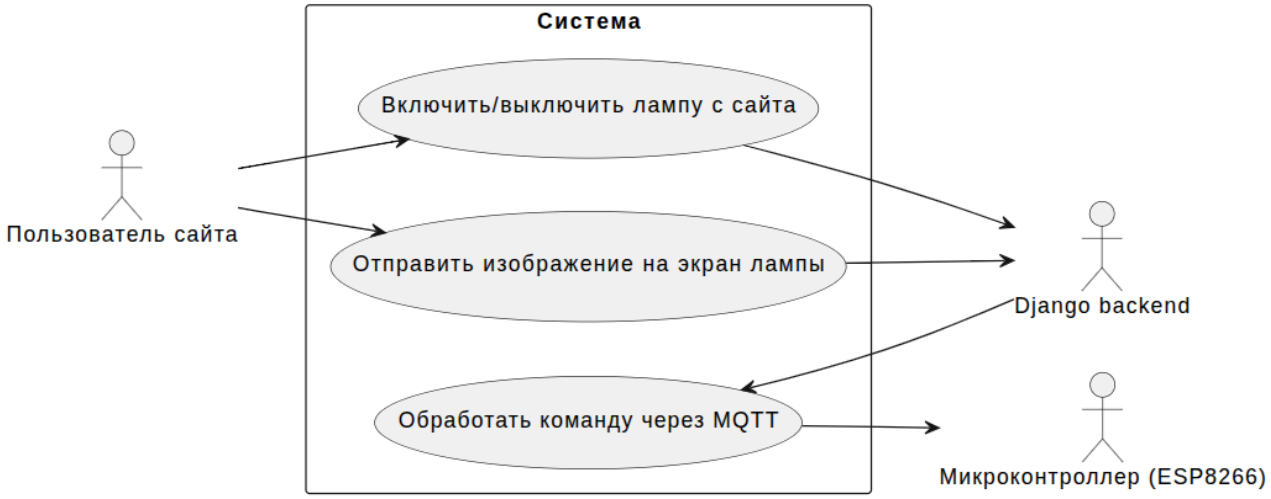
Административная часть:

- Возможность расширения интерфейса для управления изображениями.
- Масштабируемая архитектура, подходящая для подключения нескольких устройств.
- Интеграция мониторинга и логирования для отслеживания активности.

Описание технологического стека:

1. Серверная часть:
    - Язык программирования: C++ (для микроконтроллера)
    - Платформа разработки: Arduino
    - Используемые библиотеки: Adafruit GFX, Adafruit ST7735, PubSubClient, FastLED
2. Веб-интерфейс:
    - Язык программирования: Python
    - Фреймворк: Django
    - Интерфейс: HTML, CSS
3. Внешние зависимости:
    - MQTT брокер (Mosquitto)
    - Docker (контейнеризация и сборка)

Выработка требований:



1. Use Case диаграмма

Описание Use Case диаграммы

Система включает три актора:

- Пользователь сайта — взаимодействует с веб-интерфейсом, отправляет команды.
- Django backend — обрабатывает команды, публикует сообщения в MQTT-брокер.
- Микроконтроллер (ESP8266) — получает сообщения по MQTT и выполняет соответствующие действия (включение/выключение лампы, отображение изображения).

Основные сценарии использования (Use Cases):

1. Включение/выключение лампы:
    - Пользователь нажимает кнопку на сайте.
    - Django backend формирует и отправляет MQTT-сообщение.
    - ESP8266 получает команду и включает/выключает лампу.
2. Отправка изображения:
    - Пользователь выбирает изображение в веб-интерфейсе.
    - Django backend отправляет команду через MQTT с указанием изображения.
    - ESP8266 отображает соответствующую эмоцию на экране.
3. Обработка MQTT-команд:
    - ESP8266 подписан на определенный MQTT-топик.
    - При получении команды она анализируется, и выполняется действие: изменение изображения или состояния лампы.

Масштабируемость системы

Система спроектирована с учетом возможного роста нагрузки и количества устройств. В случае увеличения числа подключённых умных ламп или пользователей, архитектура может быть масштабирована по следующим направлениям:

- **MQTT-брокер:** заменяется на кластерный брокер (например, EMQX или HiveMQ), что позволяет обрабатывать тысячи одновременных подключений.

- **Backend:** масштабируется с помощью Gunicorn + Nginx и балансировщика нагрузки, при необходимости контейнеризуется в несколько инстансов с общей Redis-кэш системой.

- **Фронтенд:** остается статическим и может быть вынесен на CDN для ускорения отдачи контента.

- **Хранение изображений:** выносится в объектное хранилище (например, S3-совместимое), что позволяет поддерживать тысячи изображений без нагрузки на локальный диск.

- **Мониторинг и логи:** развертываются централизованные системы сбора и анализа (Prometheus, Grafana, Loki) для отслеживания производительности и поиска сбоев.

- **Безопасность и авторизация:** при необходимости подключается система авторизации пользователей (OAuth, JWT) и разграничения прав доступа.

Такой подход позволяет системе безболезненно масштабироваться в 10 и более раз как по количеству пользователей, так и по числу устройств, сохраняя стабильность и отзывчивость интерфейса.

Разработка архитектуры и детальное проектирование:

Объем хранилища: изображения битмапы, объем до 1MB/файл, 100 файлов в хранении.

Детализация нагрузки:

Суточные метрики:

Компонент	Операций/ день	Пиковая нагрузка (18:00–23:00)	R/W соотношение
Отображение эмоций	~ 150	60/час	10% W / 90% R
Отправка изображений	~ 50	25/час	50% W / 50% R
Обновление состояния	~ 100	40/час	30% W / 70% R
Веб-запросы к API	~ 800	250/час	5% W / 95% R
MQTT сообщения	~ 1000	400/час	100% PUBLISH

Итого:

Средняя нагрузка: ~10 RPS (запросов в секунду)

Пиковая нагрузка: до 50 RPS (в выходные)
Годовой объем данных: ~500 MB (с учётом логов, сообщений и резервных копий)

Детализация объемов трафика и дискового хранилища

1. Расчет сетевого трафика

- Входящий трафик (запросы)

Тип запроса	Размер запроса	Запросов/день	Объем/день	Пиковый RPS
Включение/выключение лампы	0.5 KB	150	75 KB	10 RPS
Отправка изображений	2 KB	50	100 KB	5 RPS
Обновление состояния	1 KB	100	100 KB	8 RPS
Веб-запросы от пользователей	5 KB	800	4 MB	50 RPS

Итого входящий трафик: ~4.275 MB/день

- Исходящий трафик (ответы)

Тип ответа	Размер ответа	Запросов/день	Объем/день
Подтверждение команды	1 KB	150	150 KB
Передача изображения устройству	20 KB	50	1 MB
MQTT уведомления	0.2 KB	1000	200 KB

Итого исходящий трафик: ~1.35 MB/день

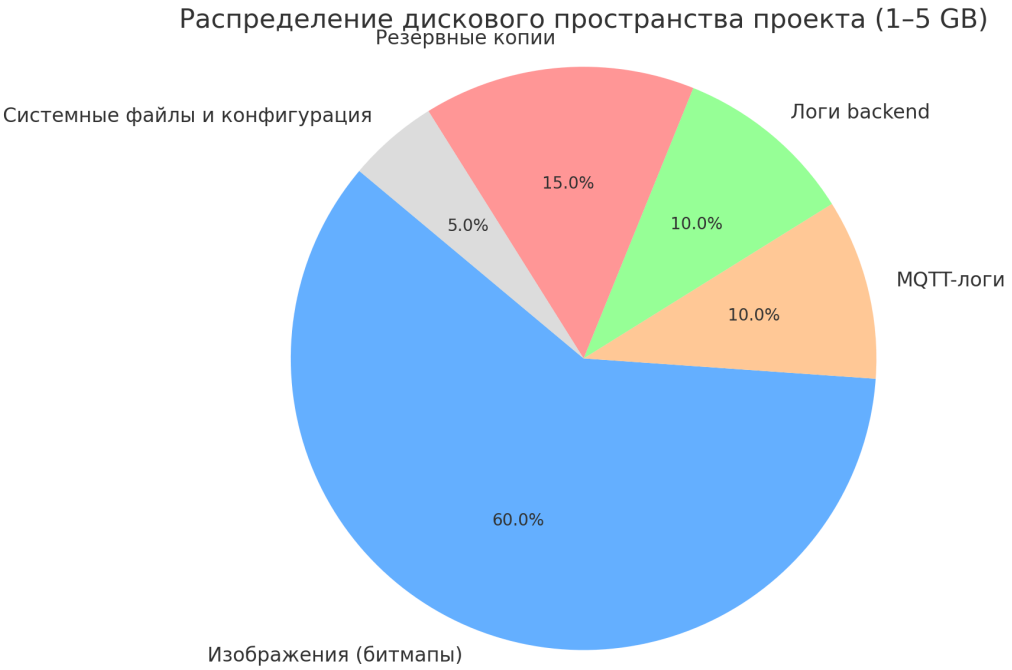
Общий суточный трафик: ~5.6 MB (с учётом накладных расходов и overhead-пакетов).

Пиковые нагрузки

- Основная активность ожидается в вечернее время (18:00–23:00) по будням и в выходные.
- Трафик увеличивается примерно в **3 раза** — до **40 MB/день**.
- Передача изображений и команд через MQTT растёт до **50 сообщений/час**.
- Одновременные подключения к интерфейсу: до **10 пользователей**.

Требования к дисковому хранилищу и сети

- **IOPS**: минимум **100** (оптимально — SSD).
- **Latency**: менее **10 ms**, особенно при загрузке изображений или быстрой смене состояний.
- Хранилище для изображений: от **1 до 5 GB** (в зависимости от количества графических файлов).



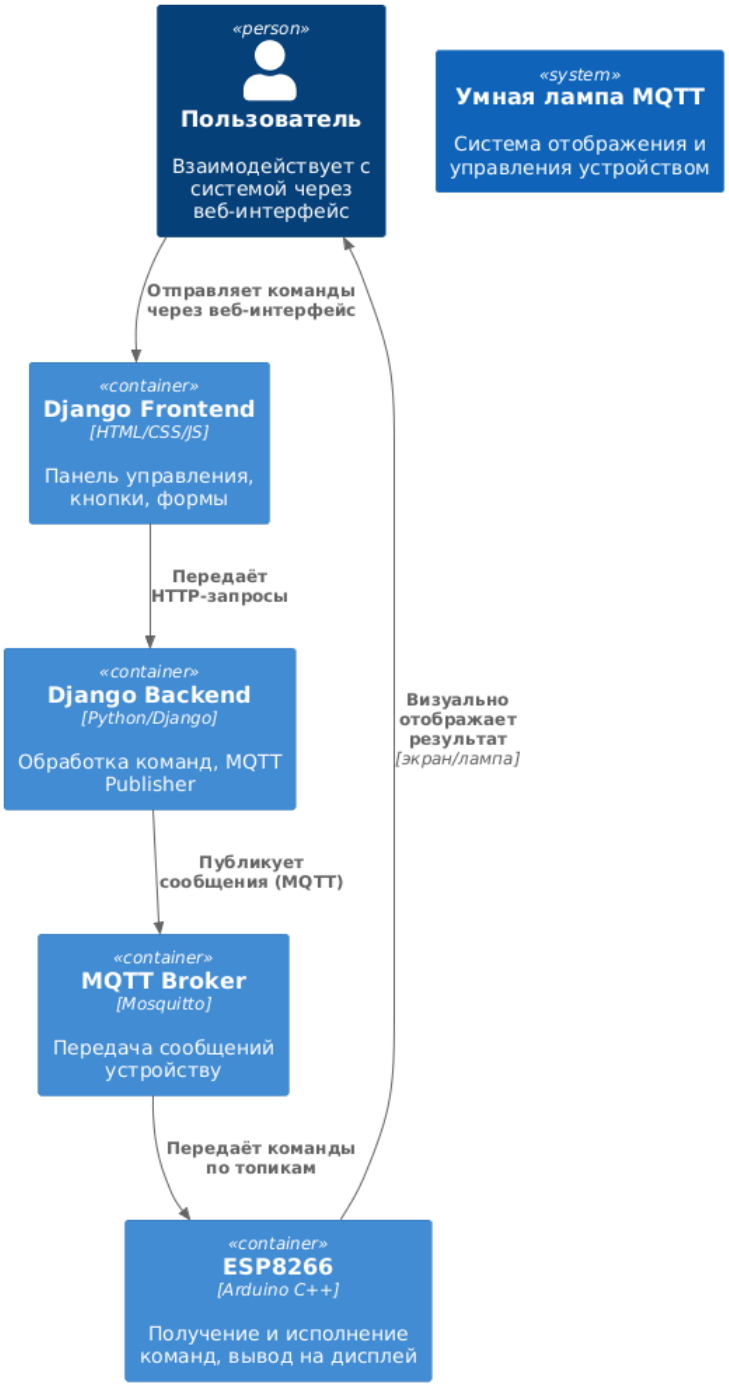
Механизмы обеспечения стабильности

- **MQTT-брокер Mosquitto** оптимизирован под высокочастотные публикации.
- **Django backend** масштабируется с помощью Gunicorn + Docker Swarm.
- Используется **Redis** для кэширования часто запрашиваемых данных (напр., список эмоций).
- Логирование через Docker volumes с ротацией и сжатием.
- Ежедневные резервные копии файлов (изображений) в облачное хранилище (например, S3-совместимое MinIO).

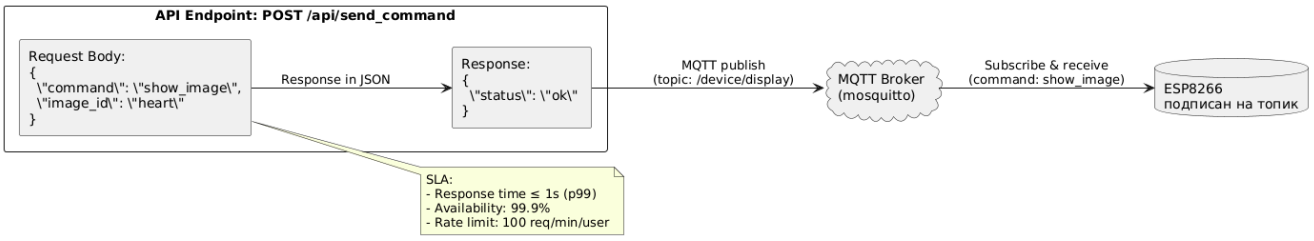
Итоговые цифры

- **Сетевой трафик**: ~13 MB/день (в пике до 40 MB).
- **Дисковое хранилище**: 1–5 GB.

Углубленная C4-диаграмма:



SLA:



`POST /api/send_command`.

Этот эндпоинт позволяет пользователю отправить команду на отображение изображения. Backend обрабатывает запрос и публикует сообщение в MQTT-брокер, где его уже принимает устройство ESP8266.

Ключевые характеристики SLA:

- **Время ответа:** ≤ 1 секунды в 99% случаев (p99).
- **Доступность:** 99.9%, что допускает максимум 8.8 часов простоя в год.
- **Лимит нагрузки:** 100 запросов в минуту на пользователя, чтобы избежать перегрузки системы.

План масштабирования (x10):

Сценарий:

- Пиковая нагрузка: 500 RPS (50 000 операций/день).
- Одновременные подключения: до 100 пользователей.
- Количество устройств: до 100 единиц.

Решение:

1. API (Django Backend):
    - Развертывание 10+ инстансов за балансировщиком нагрузки (Nginx, round-robin).
    - Автомасштабирование при CPU > 70% или latency > 1 сек.
    - Использование Docker Swarm для управления контейнерами.
2. MQTT-брокер:
    - Замена Mosquitto на кластерное решение (EMQX или HiveMQ).
    - Настройка балансировки нагрузки между нодами брокера.
3. Хранение изображений:
    - Перенос в S3-совместимое хранилище (MinIO или AWS S3).
    - Кэширование часто используемых изображений через Redis.
4. Веб-интерфейс:
    - Вынос статического контента на CDN (Cloudflare, AWS CloudFront).

Мониторинг и логи:

Дашборды (Grafana):

- Основные метрики:
    - RPS, latency (p99), ошибки API.
    - Нагрузка на MQTT-брокер (количество сообщений, подписчиков).
    - Использование CPU, памяти и диска на серверах.
- Устройства:

    - Онлайн/оффлайн статус.
    - Задержка обработки команд.

Алерты:

- Нарушение SLA:
    - Latency > 1 сек для API.
    - Доступность MQTT-брокера < 99.9%.

Логирование:

- Централизованный сбор логов (Loki + Grafana).
- Мониторинг аномалий (например, частые переподключения устройств).

Возможность расширения интерфейса для управления изображениями.

- Масштабируемая архитектура, подходящая для подключения нескольких устройств.
- Интеграция мониторинга и логирования для отслеживания активности