Курс: Otus Software Architect
Домашнее задание 4

по кате Нила Форда “Lights, Please”

Выполнил: Крошкин Игорь

31.01.2024

Содержание

1. Бизнес-контекст

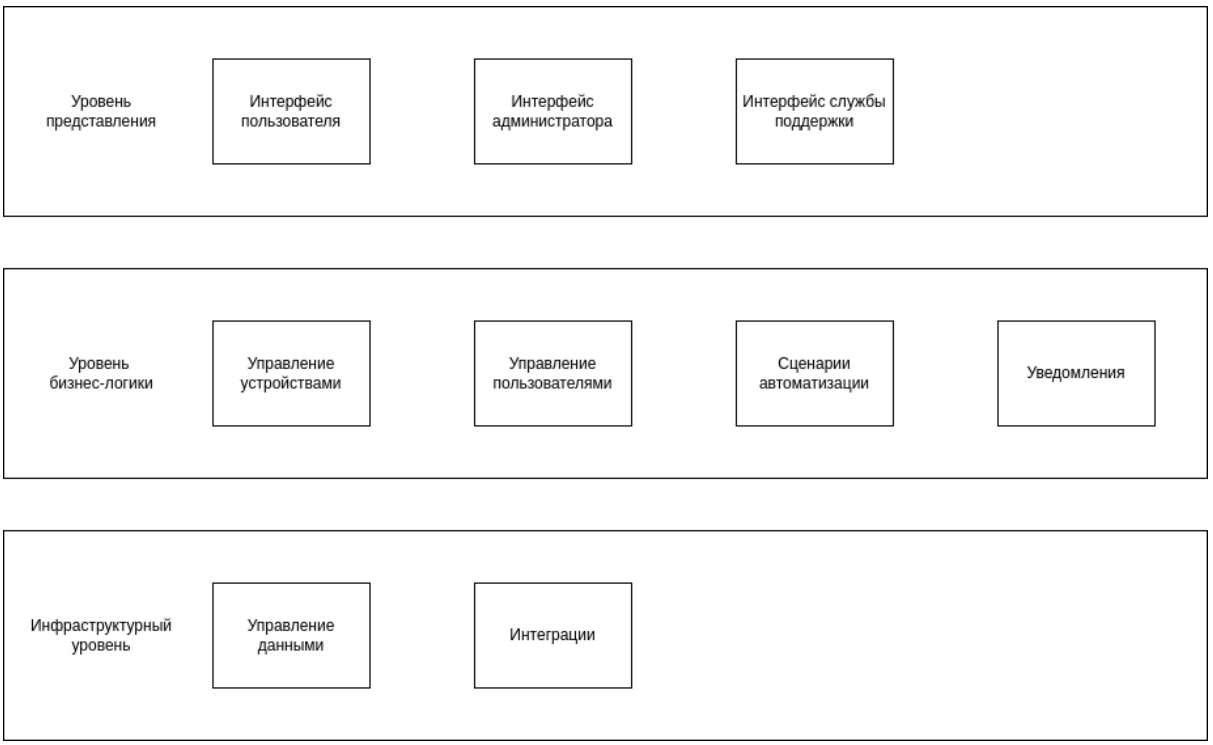
Оригинал: https://nealford.com/katas/kata?id=LightsPlease

Гигант в сфере бытовой электроники хочет создать систему для автоматизации дома: включение и выключение света, запирание и отпирание дверей, удаленное наблюдение с помощью камер и неопределенное поведение в будущем.

- Пользователи: каждая система будет продаваться потребителям (небольшим семьям), но компания рассчитывает продать тысячи таких устройств в течение первых трех лет.
- Требования:
    - система должна быть максимально готова к эксплуатации, но при этом продаваться в модульных блоках (камера, замок, термостат и т. д.) для удобства покупки
    - устройства должны быть доступны через Интернет (для удаленного мониторинга и доступа), и предполагается, что у пользователя будет существующая настройка WiFi (маршрутизатор и подключение) для подключения
    - клиенты могут программировать систему для управления различными модулями в соответствии со своими потребностями.
    - электротехникой для блоков займутся другие группы, а программные протоколы для управления модулями будут гибкими в соответствии с потребностями/проектами вашей архитектуры. (Они займутся реализацией модульной части протокола, как только вы им это укажете.)
- Дополнительный контекст:
    - готов инвестировать большую сумму, чтобы запустить это новое направление бизнеса
    - собирает данные от клиентов, которые согласились собирать более широкую статистику
    - международная компания

2. Функциональная декомпозиция

Рис.2.1. Декомпозиция по функциональным модулям в слоистой архитектуре.



При данном способе декомпозиции система разделяется на модули, каждый из которых отвечает за определённую функциональность (например, управление устройствами, автоматизация, уведомления).

1. Интерфейс пользователя - обеспечивает взаимодействие пользователя с системой через приложения:

- Мобильное приложение (iOS, Android) и веб-интерфейс
- Визуализация данных (состояние устройств, видеопотоки, лог событий)
- Управление устройствами и настройка сценариев
- Голосовое управление через ассистенты

2. Интерфейс администратора - обеспечивает расширенные функции для управления безопасностью и интеграциями, которые доступны всем пользователям системы:

- Расширенное управление пользователями, устройствами
- Настройка шаблонов пользовательского интерфейса
- Проведение диагностики системы, включая состояние устройств, пользователей и сценариев
- Операции, связанные с обновлениями системы.

3. Интерфейс службы поддержки - обеспечивает взаимодействие сотрудника службы поддержки с пользователем:

- Веб-интерфейс
- Просмотр поступивших запросов
- Фильтрация запросов по приоритету
- Управление запросами на основе обращений

- Отслеживание истории запросов от каждого пользователя

4. Управление устройствами - обеспечивает взаимодействие с физическими устройствами умного дома:

- Управление освещением (включение/выключение, регулировка яркости, изменение цвета)
- Контроль замков (запирание/отпирание дверей, мониторинг состояния)
- Управление камерами (включение/выключение, запись, поворотные механизмы, доступ к видеопотоку)
- Поддержка стандартов и протоколов (Zigbee, Z-Wave, Wi-Fi, Bluetooth, MQTT)
- Подключение новых устройств (прошивка, настройка, удаление)

5. Управление пользователями - контролирует доступ пользователей к системе:

- Аутентификация (пароль, биометрия, двухфакторная аутентификация)
- Управление ролями (администратор, пользователь, гость)
- Настройка разрешений на управление отдельными устройствами
- Логирование и мониторинг действий пользователей

6. Сценарии автоматизации - реализует автоматизацию и сценарии для умного дома:

- Создание пользовательских сценариев (например, «включить свет при движении в комнате»)
- Реакция на события (например, срабатывание датчика движения или открытие двери)
- Работа с расписанием (включение/выключение по времени)
- Настройка триггеров на основе условий (например, «закрыть двери, если дома никого нет»)

7. Уведомления - уведомляет пользователей о событиях в системе:

- Push-уведомления, SMS, email (например, о незакрытых дверях)
- Настройка приоритетов уведомлений (обычные, критические)
- Интеграция с системами оповещения (например, тревожные сигналы)

8. Управление данными - сохраняет и обрабатывает данные, поступающие от устройств и пользователей:

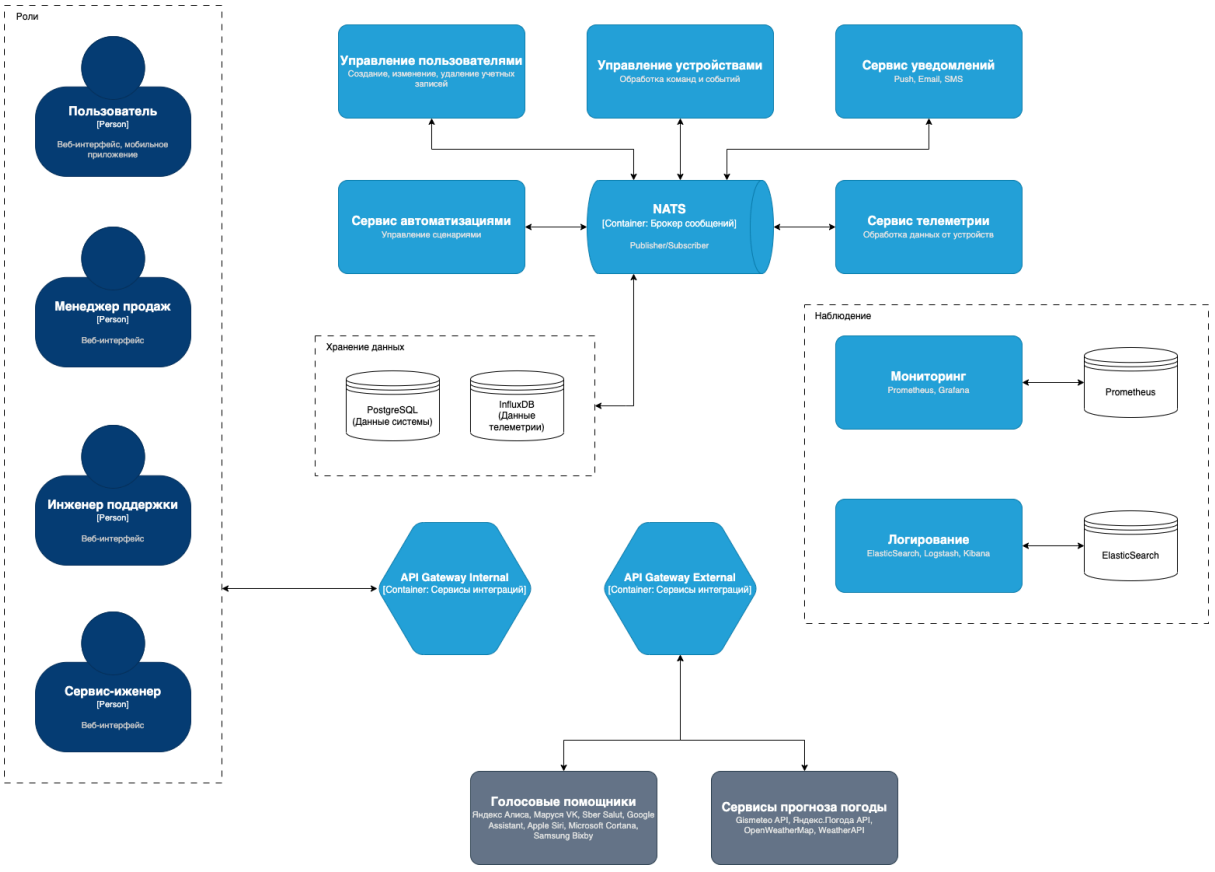
- Хранение истории событий (видеозаписи, действия пользователей, состояния устройств)
- Аналитика (например, энергоэффективность, популярность сценариев)
- Поддержка облачного и локального хранения данных
- Подготовка данных для искусственного интеллекта (например, предсказание поведения).

9. Интеграции - обеспечивает взаимодействие с другими платформами и системами:

- Подключение голосовых ассистентов
- Интеграция с погодными сервисами (для автоматизации на основе прогноза)
- Взаимодействие с системами безопасности (например, вызов службы охраны)
- Поддержка API для интеграции сторонних устройств

3. Диаграмма контейнеров

Рис. 3.1. Диаграмма контейнеров



Контейнеры

1. Управление пользователями - отвечает за регистрацию, аутентификацию и управление правами доступа пользователей. Обеспечивает безопасность и персонализацию работы с системой
2. Управление устройствами - отвечает за регистрацию устройств, настройку, мониторинг и управление умными устройствами. Позволяет администрировать подключенные сенсоры и контроллеры

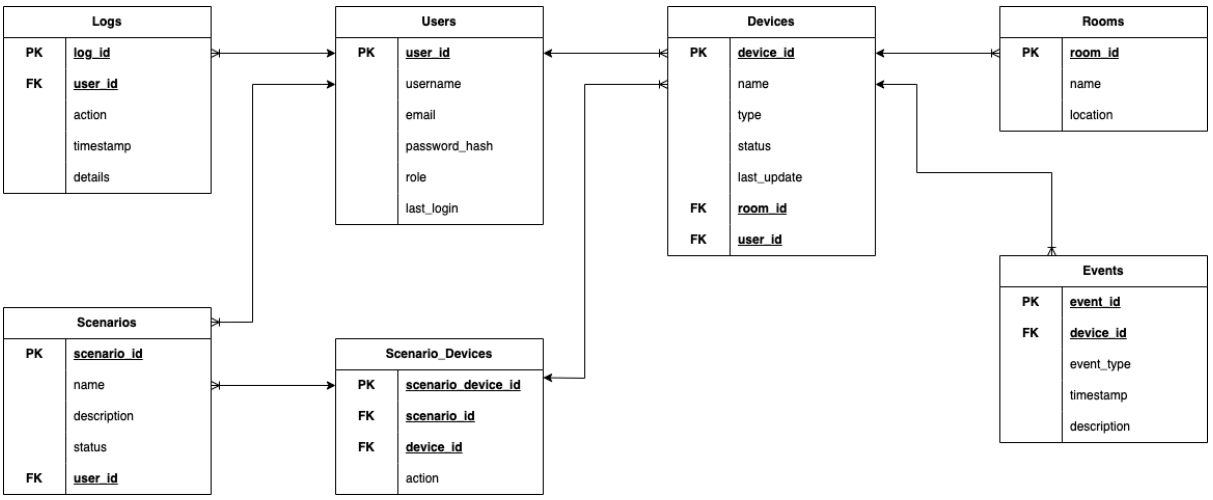
3. Сервис уведомлений - отправляет пользователям оповещения о событиях в системе (изменение состояния устройств, аварии, расписание и т. д.) через push-уведомления, SMS, email или мессенджеры
4. Сервис автоматизации - реализует сценарии автоматизации. Позволяет пользователям задавать правила (например, включение света при движении) и управлять устройствами на основе условий и расписаний
5. Сервис телеметрии - модуль для сбора, хранения и анализа данных с умных устройств. Позволяет вести мониторинг состояния оборудования, строить графики и выявлять аномалии в работе системы
6. NATS - брокер сообщений для асинхронного взаимодействия между модулями системы. Гарантирует надежную доставку сообщений, маршрутизацию и поддержку подписок
7. Наблюдение - для сбора и анализа журналов используется ELK, для мониторинга компонентов Prometheus и дашборды Grafana
8. Хранение данных - для хранения данных системы используется реляционная БД PostgreSQL, для хранения данных телеметрии используется БД временных рядов InfluxDB
9. API Gateway Internal - внутренний API-шлюз, который управляет взаимодействием между сервисами внутри системы. Он выполняет маршрутизацию запросов, балансировку нагрузки, аутентификацию и контроль доступа
10. API Gateway External - внешний API-шлюз, предоставляющий безопасный доступ для интеграции с внешними платформами. Обеспечивает защиту API, мониторинг и управление трафиком

4. Декомпозиция слоя данных

4.1. Данные системы

Для хранения данных системы предлагается использовать реляционную СУБД, которая поддерживает гибкость, масштабируемость и быструю обработку для хранения данных об устройствах, пользователях, комнатах и сценариях.

Рис. 4.1.1. Схема базы данных для хранения данных системы



Таблицы

1. Users (Пользователи)
    - user_id (PK): Уникальный идентификатор пользователя
    - username: Имя пользователя для входа в систему
    - email: Электронная почта
    - password_hash: Хэш пароля пользователя
    - role: Роль пользователя (например, администратор, пользователь)
    - last_login: Время последнего входа пользователя
2. Devices (Устройства)
    - device_id (PK): Уникальный идентификатор устройства
    - name: Название устройства (например, "Умная лампа", "Термостат")
    - type: Тип устройства (например, лампа, термостат, датчик движения)
    - status: Статус устройства (включено/выключено)
    - last_update: Время последнего обновления состояния устройства
    - room_id (FK): Ссылка на комнату, к которой относится устройство
    - user_id (FK): Ссылка на владельца устройства (пользователь)
3. Rooms (Комнаты)
    - room_id (PK): Уникальный идентификатор комнаты
    - name: Название комнаты (например, "Гостиная", "Кухня")
    - location: Местоположение комнаты (например, этаж, номер квартиры)
4. Scenarios (Сценарии)
    - scenario_id (PK): Уникальный идентификатор сценария
    - name: Название сценария (например, "Уход из дома", "Вечерний режим")
    - description: Описание сценария (что он делает).
    - status: Статус сценария (активен/неактивен)
    - user_id (FK): Ссылка на пользователя, создавшего сценарий

5. Scenario_Devices (Устройства в сценах)
    - scenario_device_id (PK): Уникальный идентификатор устройства в сценарии
    - scenario_id (FK): Ссылка на сценарий
    - device_id (FK): Ссылка на устройство
    - action: Действие, выполняемое с устройством (например, включение/выключение)
6. Events (События)
    - event_id (PK): Уникальный идентификатор события
    - device_id (FK): Ссылка на устройство, которое вызвало событие
    - event_type: Тип события (например, "включение", "выключение", "добавление нового устройства")
    - timestamp: Время возникновения события
    - description: Описание события (например, "Лампа включена", "Термостат установил 22°C")
7. Logs (Журналы действий)
    - log_id (PK): Уникальный идентификатор записи в журнале
    - user_id (FK): Ссылка на пользователя, который выполнил действие
    - action: Тип действия (например, "включил устройство", "изменил настройки")
    - timestamp: Время действия
    - details: Подробности действия

Связи между таблицами

1. Users ↔ Devices
    - Один ко многим: Каждый пользователь может иметь несколько устройств
2. Users ↔ Scenarios
    - Один ко многим: Каждый пользователь может создать несколько сценариев
3. Devices ↔ Rooms
    - Один ко многим: Каждая комната может содержать несколько устройств
4. Scenarios ↔ Devices
    - Многие ко многим: сценарий может включать несколько устройств, и одно устройство может быть частью нескольких сценариев, связь через таблицу Scenario_Devices
5. Devices ↔ Events
    - Один ко многим: Одно устройство может генерировать множество событий
6. Users ↔ Logs
    - Один ко многим: Один пользователь может выполнить множество действий, которые будут записаны в журнал

4.2. PostgreSQL для хранения данных системы

В системе управления умным домом обычно требуется быстрое чтение и запись данных, поддержка различных типов устройств и сценариев, поэтому выбор сделан в пользу реляционной БД с открытым исходным кодом PostgreSQL, которая отличается хорошей производительностью и подходит для хранения структурированных данных.

4.3. Данные телеметрии

Для хранения данных телеметрии предлагается использовать БД временных рядов, которые в отличие от реляционных СУБД используют не фиксированную схему таблиц, а следующую структуру:

1. Measurements (Измерения) - аналог таблиц в реляционных СУБД
    - sensor_data: данные с датчиков (температура, влажность и др.).
    - energy_usage: потребление энергии.
    - device_status: статус устройств (вкл/выкл, ошибки и т. д.).
    - security_events: события безопасности (открытие дверей, срабатывание сигнализации).
    - climate_control: управление климатом (температура, режим работы кондиционеров/отопления).
2. Tags (Тэги) - используются для быстрой фильтрации данных
    - device_id: уникальный идентификатор устройства.
    - room: помещение, к которому относится устройство (например, "living_room", "kitchen").
    - sensor_type: тип датчика (например, "temperature", "humidity", "motion").
    - unit: единица измерения (например, "°C", "%", "W").
    - status: состояние устройства (например, "ON", "OFF", "ERROR")
3. Fields (Поля) - содержат фактические значения измерений, пример sensor_data:

timestamp	device_id	room	sensor_type	unit	value
2025-01-29T12:00:00Z	sensor_1	kitchen	temperature	°C	22.5
	sensor_2	living_room	humidity	%	45.2

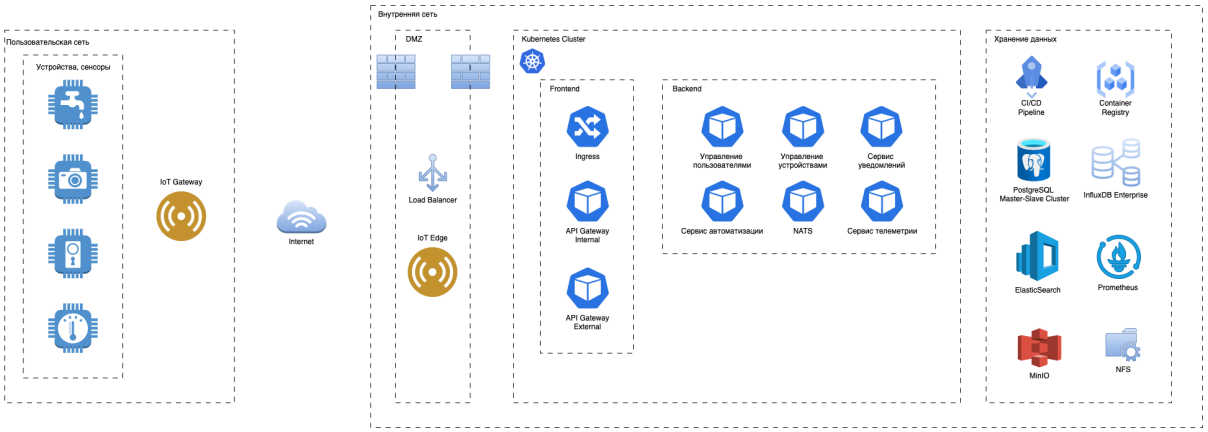
2025-01 -29T12: 00:05Z					

4.4. InfluxDB для хранения данных телеметрии

Выбор сделан в пользу InfluxDB, так как она оптимизирована для обработки больших объемов данных в реальном времени и распределенных системах для высокоскоростной записи и анализа данных, что делает ее оптимальным выбором для современных IoT-систем.

5. Диаграмма развертывания

Рис. 5.1. Диаграмма развертывания



Так как система использует микросервисную архитектуру, для развертывания модулей используется кластер Kubernetes на внутренней площадке. Со временем система может быть мигрирована в облако либо использоваться гибридный подход.

Компоненты

1. IoT Gateway - пользовательский шлюз, обеспечивающий подключение и взаимодействие умных устройств с системой. Он выполняет сбор, предварительную обработку и передачу данных от сенсоров и контроллеров
2. IoT Edge - модуль периферийных вычислений, принимающий данные от пользовательских шлюзов. Уменьшает задержки и нагрузку на сеть, обеспечивая автономность работы системы

3. DMZ - демилитаризованная зона для изоляции внутреннего сегмента от сети Интернет
4. Load Balancer - балансировщик нагрузки между API Gateway External и API Gateway Internal
5. Кластер Kubernetes
    - Ingres - обеспечивает управление входящим трафиком к сервисам внутри кластера Kubernetes
    - Namespace - для логического разделения ресурсов между Frontend и Backend соответствующие поды работают в разных namespace
6. CI/CD - используется для автоматического обновления сервисов внутри кластера Kubernetes
7. Container Registry - используется для хранения артефактов и контейнеров, необходимых при развертывания приложений
8. PostgreSQL - используется синхронная репликация данных в режиме Active-Active
9. InfluxDB Enterprise - используется редакция Enterprise в режиме High Availability и шардированием данных между узлами
10. ElasticSearch Cluster - используется кластер из нескольких узлов в режиме Master-Slave и шардированием индексов между узлами
11. Prometheus - использование двух реплик в режиме Active-Passive
12. MinIO, NFS - используются в качестве объектного и файлового хранилища персистентных данных контейнеров