# Глоссарий возможно будет вырезан, но мне очень хочется его оставить

# **Определения, обозначения и сокращения**

Термины, использующиеся в работе

**HTTP** (Hypertext Transfer Protocol) – это протокол передачи данных в сети Интернет, который определяет формат и правила обмена информацией между клиентом и сервером. Он обеспечивает передачу различных типов данных, таких как текст, изображения, видео и другие мультимедийные ресурсы, используя модель запрос-ответ.

**Фреймворк** – это набор готовых инструментов, библиотек и шаблонов, который предоставляет разработчикам структуру и ресурсы для создания приложений, упрощая их разработку и обеспечивая согласованность и эффективность в процессе.

**DI** (Dependency Injection) – это паттерн проектирования программного обеспечения, который позволяет управлять зависимостями между компонентами приложения, передавая им необходимые ресурсы извне. Это делает приложения более гибкими, модульными и легко тестируемыми.

**База данных** – это организованная коллекция данных, хранящаяся в компьютерной системе и предназначенная для эффективного хранения, управления и обработки информации. Она предоставляет средства для структурирования данных, выполнения запросов и обеспечения доступа к информации для различных приложений и пользователей.

**СУБД** (система управления базами данных) – это программное обеспечение, предназначенное для создания, управления и обработки баз данных. Оно обеспечивает эффективное хранение, организацию и доступ к данным, а также предоставляет средства для выполнения запросов, обеспечения целостности данных и обеспечения безопасности информации.

**Введение**

В современном мире электронные устройства стали неотъемлемой частью повседневной жизни человека. От смартфонов и планшетов до умных часов и домашних помощников, эти устройства улучшают качество жизни, обеспечивая удобство, доступность информации и новые способы взаимодействия с окружающим миром. В автомобилях, бытовой технике и даже в медицинском оборудовании, встраиваемые устройства помогают в навигации, управлении энергопотреблением и поддержании здоровья человека. Это непрерывно растущая экосистема устройств, которые становятся всё более интеллектуальными и связанными между собой.

С увеличением сложности и функциональности встраиваемых устройств возрастает и их уязвимость к программным сбоям и атакам, что подчеркивает необходимость постоянного обновления программного обеспечения. Эти устройства, будь то элементы интернета вещей, медицинское оборудование или системы управления в автомобилях, требуют постоянных обновлений для устранения уязвимостей, улучшения функциональности и повышения общей надежности. Наличие актуального и безопасного программного обеспечения напрямую влияет на эффективность и безопасность работы этих устройств, а также на удовлетворенность конечных пользователей.

Разработка системы обновления программного обеспечения для встраиваемых устройств является сложной инженерной задачей, которая включает в себя ряд аспектов. Система должна обеспечивать надежное и быстрое распространение обновлений среди множества устройств, минимизируя простои и риски для конечных пользователей. В данной работе будет представлен процесс разработки такой системы, начиная от выбора технологий до реализации ключевых функций и тестирования. Это исследование направлено на создание эффективной, масштабируемой и безопасной системы, способной адаптироваться к быстро меняющимся технологическим требованиям и повышать уровень доверия пользователей к электронным устройствам, которые стали неотъемлемой частью нашей жизни.

**Цель работы**

Целью моей работы будет являться разработка системы для службы обновлений программного обеспечения встраиваемых устройств. Рассмотренной мною сферой будет автомобильные сервисы (или Сервисы Технического Обслуживания). Эта система должна эффективно обрабатывать множество запросов, соответствовать нормам отказоустойчивости (нарыть эти нормы, если такие есть…), иметь удобный интерфейс и быть легко масштабируемой.

**Постановка задач**

Исходя из поставленной цели, можно выделить следующие задачи:

- Проведение анализа предметной области

- Анализ рынка текущих решений, выделение требований к системе

- Анализ и выбор средств разработки

- Разработка системы для службы обновлений встраиваемых устройств для СТО

- Тестирование системы, оценка с точки зрения удобства использования конечным пользователем

Глава 1. Раздел анализа

Разработка системы обновления программного обеспечения для микроконтроллеров в автосервисах является ключевой задачей, направленной на улучшение эффективности обслуживания и сокращение времени простоя автомобилей. Создание такой системы позволит:

* автоматизировать процесс получения прошивок, сократит время, необходимое на диагностику и устранение неисправностей, тем самым снижая операционные расходы сервиса;
* повысить качества обслуживания посредствам быстрой и своевременной установка обновленного ПО, что повышает надежность устройств и удовлетворенность клиентов;
* оптимизировать работы персонала, эффективно управляя процессом загрузки новых версий ПО, отслеживанием конкретного ПО для каждого автомобиля и его обновления, что позволит лучше распределять рабочую нагрузку между сотрудниками сервиса.

На данный момент отслеживание версий ПО происходит вручную. Хранение ПО для каждого микроконтроллера, обновление и актуализация хранимой информации, поиск нужных прошивок для конкретного блока – задачи не самые тривиальные и требуют огромных затрат времени и усилий персонала. Ручное обновление также увеличивает вероятность ошибок, что может привести к неправильной работе устройств и увеличению времени простоя автомобилей, что приводит к длительным простоям в работе автомобилей, а также негативно сказываются на репутации сервиса и удовлетворенности клиентов.

В ситуациях, когда требуется быстрое внедрение исправлений уязвимостей или ошибок в прошивках, отсутствие автоматизированной системы значительно увеличивает время реакции.

Каждое устройство требует индивидуального подхода в обновлении, что делает процесс медленным и увеличивает время, необходимое для обеспечения безопасности всех устройств.

Текущий процесс сильно зависит от уровня квалификации и опыта технических специалистов, которые выполняют обновления. Это создает риски, связанные с текучкой кадров и потребностью в постоянном обучении новых сотрудников.

Отсутствие интегрированных инструменты мониторинга и отчетности затрудняют отслеживание статуса обновлений и анализ эффективности внедренных изменений.

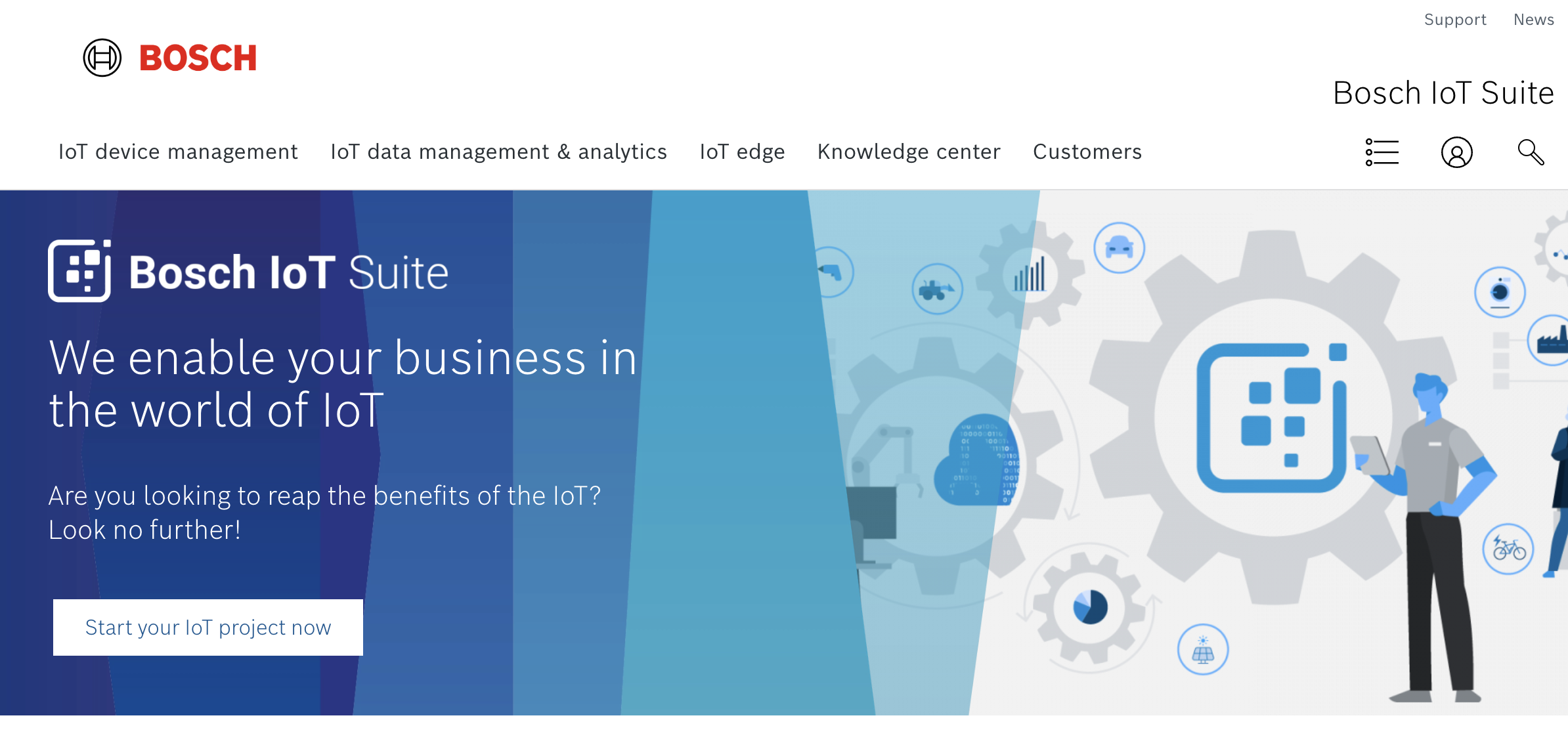
* 1. **Анализ существующих решений**

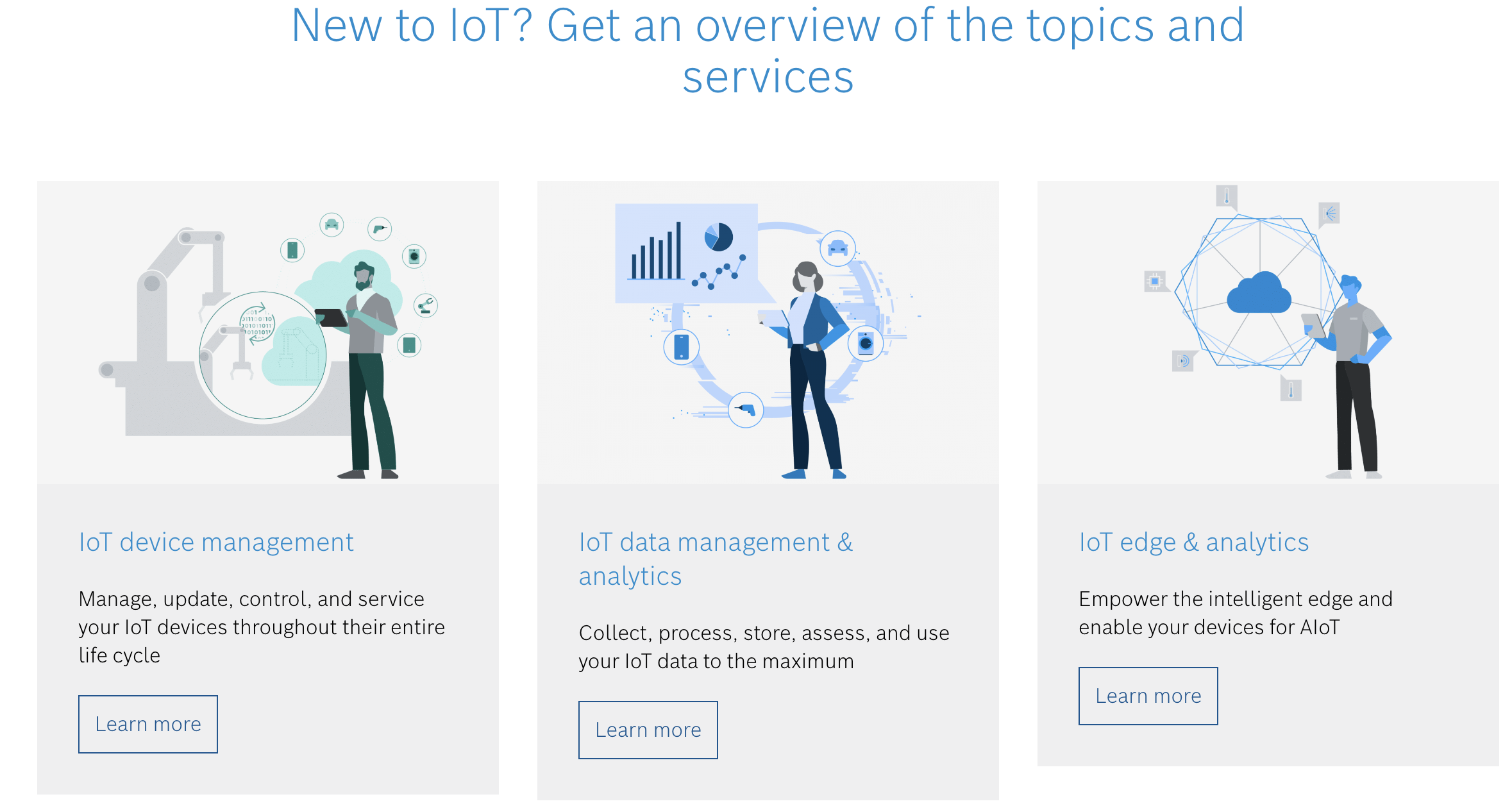
На рынке уже существуют подобные системы, способные решить задачу хранения, просмотра и доставки ПО для встраиваемых устройств.

OTA (Over-the-Air) Update Systems

Bosch IoT Suite

https://bosch-iot-suite.com/





Плюсы:

Широкий спектр возможностей

Мировая компания, заимевшая огромную известность

Хорошая система поддержки пользователей

Минусы:

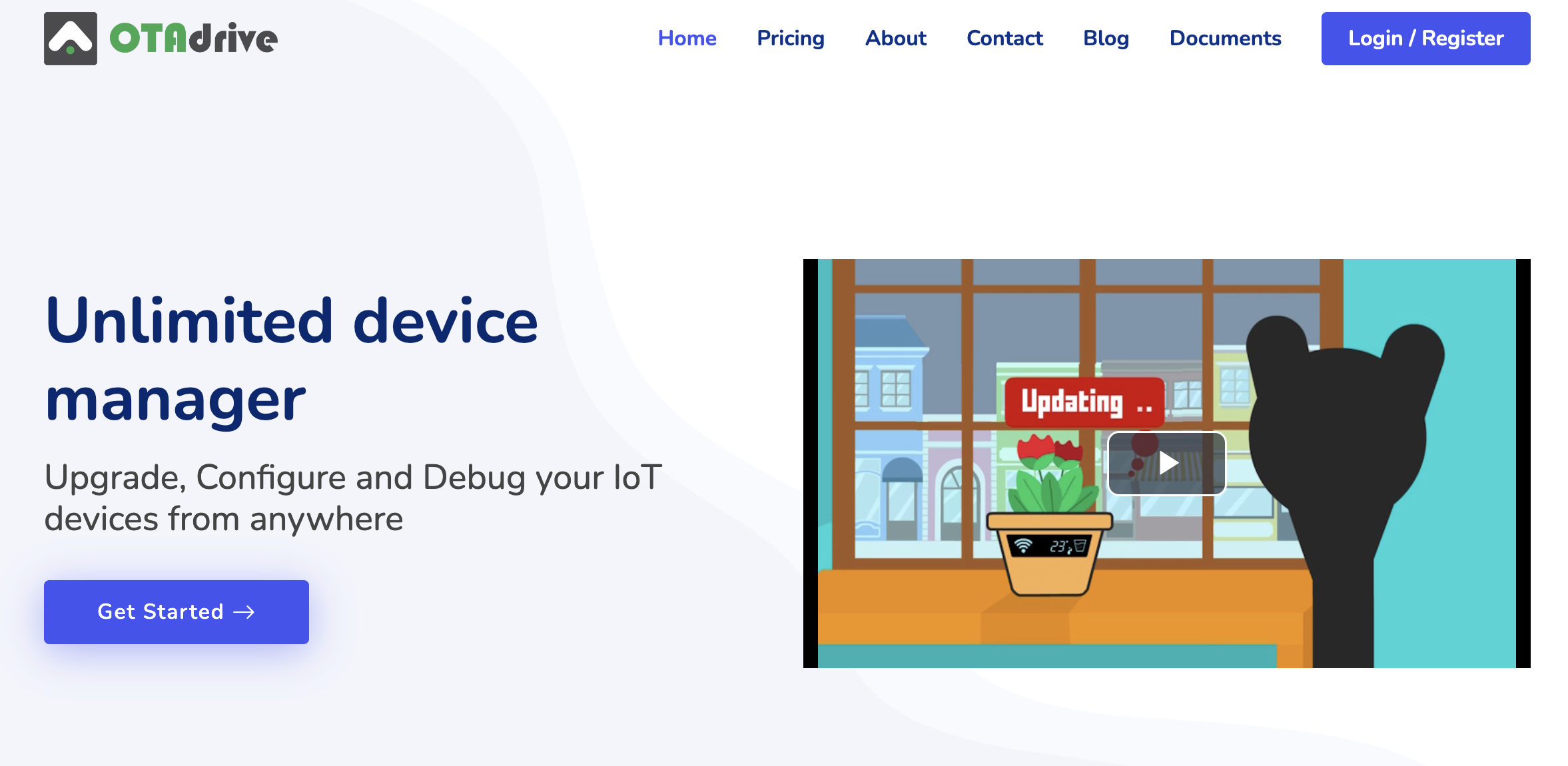
Поддержка закончится к середине 2024 года

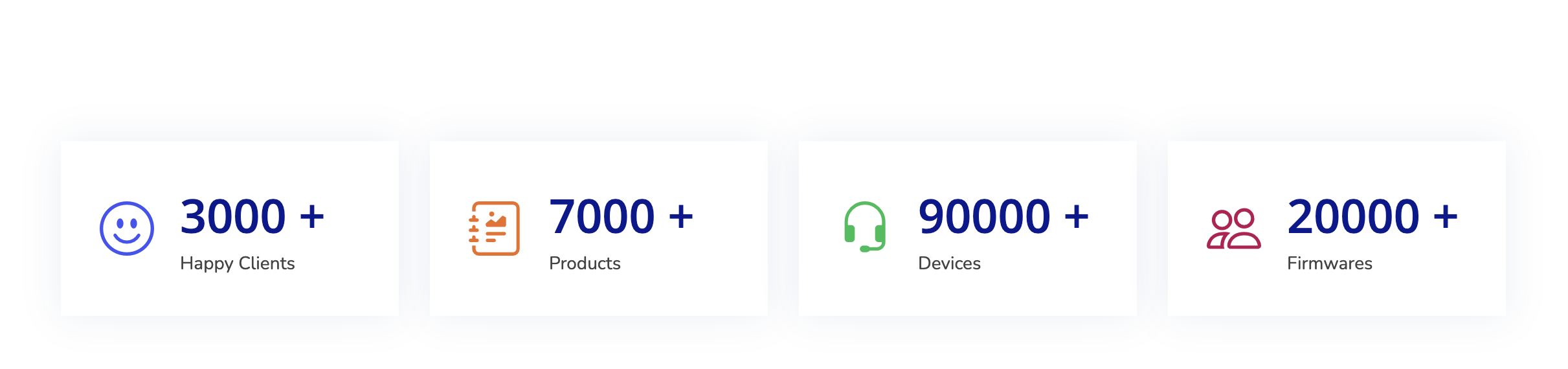
Высокая стоимость

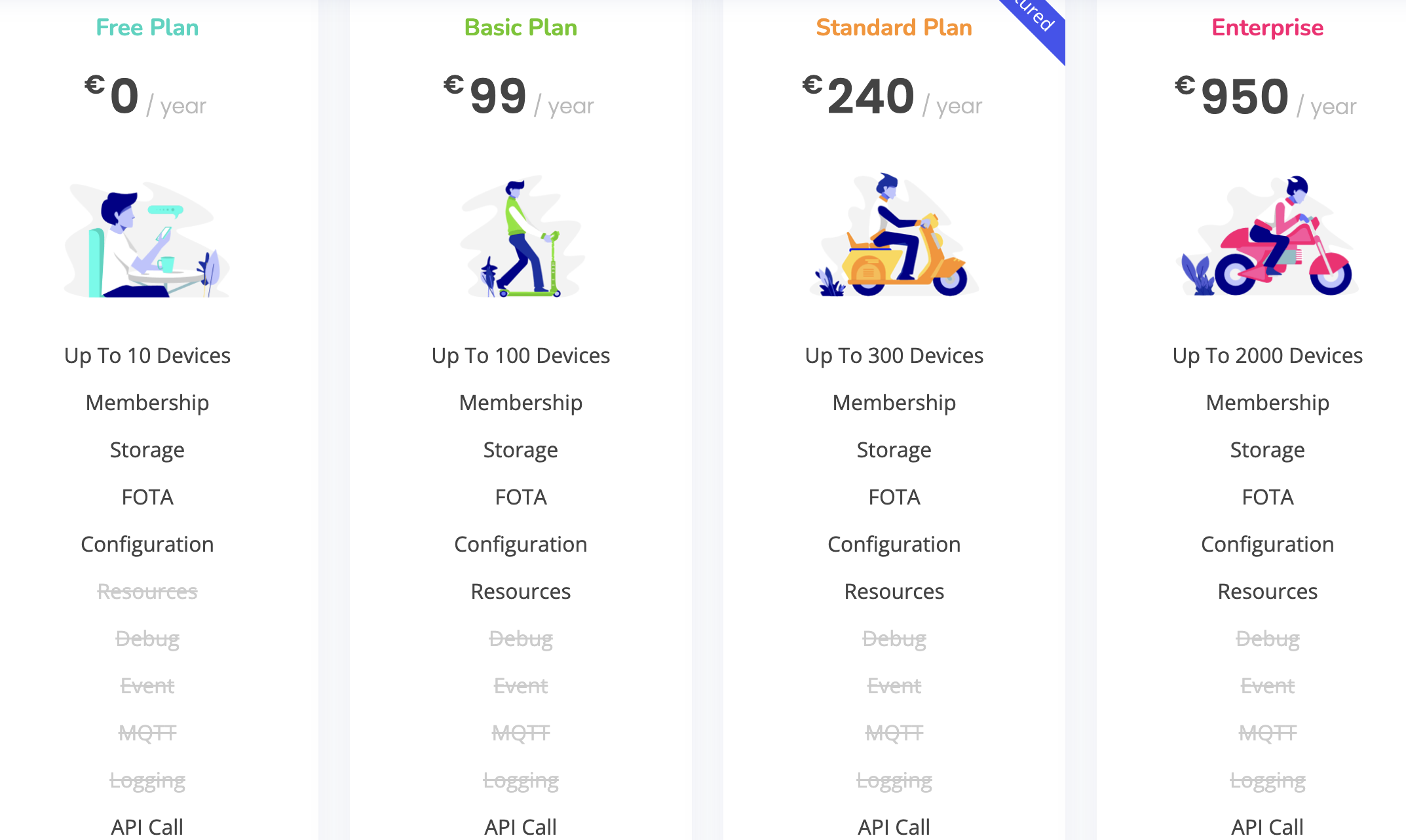
Зависимость от облачных технологий компании

OTA (Over-the-Air) Update Systems

<https://otadrive.com/>







<https://www.techtarget.com/searchmobilecomputing/definition/OTA-update-over-the-air-update>

Статья про OTA

<https://wiki.teltonika-gps.com/view/FOTA_WEB_Getting_Started>

Пример взаимодействия с FOTA

Плюсы OTA систем:

* Управление всеми обновлениями прошивок из одного центра, что обеспечивает единообразие и снижает вероятность ошибок.
* Беспроводные обновления могут быть очень удобны в использовании
* Большинство OTA систем предлагают полную автоматизацию процесса обновления, включая тестирование и откат версий

Минуты OTA систем:

- Чаще всего необходима установка доп оборудования

- Огромные цены на обслуживание сервиса

- Зависимость от облачных технологий провайдера услуг

- Беспроводные обновления требуют стабильного интернет-соединения устройства, что не всегда возможно

- Передача данных через беспроводные каналы увеличивает риск кибератак и утечек данных

2. Vendor-Specific Update Tools

Примеры: Ford SYNC, Tesla Software Updates.

Если говорить про систему обновления ПО встраиваемых устройств для автосервисов, стоит учитывать также официальные системы обновления вендоров. Они могут быть удобны в случае сервиса при дилере авто, но имеют свои недостатки при использовании сторонними частными сервисами

Плюсы Vendor-Specific Update Tools:

- Оптимизированы под конкретное оборудование, эти инструменты разработаны специально для определенных моделей или марок, обеспечивая высокую совместимость и стабильность работы.

- Постоянное обновление и поддержка от производителя оборудования.

Минусы Vendor-Specific Update Tools:

- Работают только с определенными устройствами, что не подходит для сервисов, обслуживающих множество разных марок автомобилей.

- Изменения в политике компании могут влиять на доступность и условия использования обновлений.

- Для частных сервисов, не относящихся к дилерским центрам, могут быть полностью недоступны, либо лицензии стоят необоснованно дорого

3. Open Source Update Solutions

Примеры: SWUpdate.

В качестве альтернативы собственной системы можно рассмотреть уже готовые Open Source решения, помогающие решить задачи обновлений ПО встраиваемых устройств. Использование таких инструментов может существенно сократить затраты на создание собственной системы, но взамен на это требуют специалистов, имеющих опыт работы с такими системами, при этом обновление и поддержка этих инструментов привязана к команде, работающей над ними.

Плюсы Open Source решений:

- Гибкость, высокая степень настройки под конкретные нужды и возможность изменения кода.

- По большей части, отсутствие лицензионных платежей, доступность исходного кода.

Минусы Open Source решений:

- Требования к техническим навыкам. Для настройки и поддержки таких систем требуются квалифицированные специалисты.

- Отсутствие гарантированной поддержки или обновлений без активного сообщества или коммерческой поддержки.

- Системы и инструменты, распространяющиеся по лицензии, могут иметь необоснованно большую стоимость, а использование инструмента привязывает к продлению лицензии.

Подводя итоги сравнения представленных на рынке решений, можно сказать, что основные компоненты и функциональные возможности разрабатываемой системы должны включать:

Централизованный сервер обновлений:

Сервер будет являться основным узлом, через который будут распространяться обновления. Он будет обрабатывать запросы на обновление от устройств, проверять их подлинность и отправлять соответствующие пакеты обновлений.

База данных версий прошивок:

База данных будет содержать информацию о всех доступных версиях прошивок для различных устройств. Она позволит отслеживать текущее состояние прошивок на каждом устройстве и историю обновлений.

Модуль безопасности:

Включает механизмы аутентификации устройств, шифрование данных и проверку целостности пакетов обновлений, чтобы предотвратить возможные атаки и гарантировать безопасность процесса обновления.

Интерфейс управления:

Веб-интерфейс для администраторов системы позволит мониторить статус обновлений, управлять расписанием обновлений и настраивать политики безопасности. Также он предоставит доступ к аналитическим отчетам о производительности устройств и успешности процесса обновлений.

Система будет разработана с учетом требований к масштабируемости, чтобы поддерживать работу как с небольшим количеством устройств, так и с большими массивами встраиваемых систем. Это обеспечит возможность её применения в различных отраслях и условиях эксплуатации.

## **Анализ и выбор средств разработки**

На текущий момент существует множество инструментов для разработки приложений. Так как приложение базируется на его веб составляющей, заострим внимание на два аспекта – backend (или «серверная» часть) и frontend (или «клиентская» часть) приложения.

Начнем анализ и выбор с технологий, для backend части.

* + 1. **Backend часть**

Начнем с выбора языка программирования для серверной части приложения.

<https://itproger.com/news/sravnenie-populyarnih-yazikov-programmirovaniya-dlya-veb-razrabotki>

Python является одним из наиболее популярных языков программирования для разработки веб-сервисов

Сравнивая его с другими популярными языками для написания серверной части, можно выделить Ruby, Java, JavaScript (Node.js) и PHP, как самые популярные «аналоги» Python.

**Ruby** отлично подходит для реализации приложения на ранних стадиях стартапа из-за простоты и скорости написания вместе с его конвенцией «Convention over Configuration», однако является не лучшим выбором для высоконагруженных приложений.

**PHP** – один из самых популярных языков для написания серверной части. Свою популярность он набрал из-за повсеместного использования. Однако внимание к безопасности при разработке на PHP требуется куда больше, чем с остальными языками, что может усложнить разработку. К тому же, асинхронная и многопоточная работа в PHP крайне усложнена и требует дополнительных модулей. Множество дублирующих функций для поддержки “legacy” кода мешают писать чистый и хорошо поддерживаемый код на PHP.

**Java** также является очень популярным языком. Он предлагает высокую производительность в большинстве задач, хорошую надежность и JVM – Java Virtual Machine, из-за которой развертка приложения может стать куда проще. Однако высокая производительность Java перекрывается ее не самым простым синтаксисом и строгой типизацией, что сильно замедляет разработку и поддержку кода. К тому же, высокая производительность при веб-разработке не тот параметр, на который стоит опираться, так как основное время приложение занято I/O (input/output) операциями, время на которые отлично сокращаются асинхронностью.

JavaScript (Node.js) – очень мощный для разработки веб-приложений инструмент, хорошо обрабатывающий множество асинхронных запросов, но обладающий некоторыми минуса в сравнении с Python.

Минусы JavaScript:

* Не самое эффективное использование CPU. Приложения на JavaScript (Node.js) требуют куда больше ресурсов для своей работы
* Однопоточность. Это не составляет проблем при обработке запросов из-за наличия асинхронности, но может стать проблемой при расширении проекта. JavaScript принуждает к микросервисной архитектуре, так как абсолютно не годится для большого количества вычислений или работы, например, с нейросетями.

Итого, из преимуществ Python можно выделить:

- Чистый и лаконичный синтаксис, который упрощает поддержку кода и сокращает время на разработку.

- Мощные библиотеки и фреймворки, которые облегчают реализацию сложного функционала без необходимости писать много кода.

- Высокая производительность благодаря оптимизациям и возможности интеграции с другими языками программирования.

- Возможности асинхронной разработки. FastAPI и другие асинхронные библиотеки позволяют эффективно управлять вводом-выводом, что критично для систем обновления ПО из-за большого количества I/O операций.

- Python обеспечивает легкую интеграцию с другими системами и технологиями.

Таким образом, выбор Python для разработки веб-сервиса системы обновления ПО для встраиваемых устройств является обоснованным с точки зрения как технических характеристик, так и операционной эффективности.

ТУТ ТОЖЕ САМОЕ ПРО ФРЕЙМОРКИ (аля почему FastAPI)

ТУТ ПРО БАЗЫ (почему Neo4j)

* + 1. **Frontend часть**

Для написания клиентской части веб-приложения, выбирать из технологий особо не приходится. В данной части точно будут задействованы HTML, CSS и JavaScript. Существует два подхода к отрисовке веб-интерфейса:

Client-side Rendering

Такой подход используется в таких фреймворках, как React. При запросе, пользователь получает «скелет» страницы с JavaScript кодом, который с помощью запросов на сервер динамически подгружает всю необходимую информацию. Из плюсов этого подхода можно выделить относительно простую масштабируемость при условии, что изначальный код достаточно чист. Однако сама разработка требует очень хороших знаний JavaScript (или TypeScript) и очень много времени. Однако в этом подходе есть зависимость от ресурсов клиентской части. Если компьютер клиента будет крайне маломощным, загрузка страницы займет очень много времени. К тому же, отрисовка интерфейса будет «отъедать» значительную долю слабой машины, что не очень приятно в условиях, когда изначально неизвестно, какое железо будет трудиться на клиентской части.

Server-side Rendering

При этом подходе вся информация изначально собирается на сервере в единую «кучу» и отправляется клиенту вместе со скелетом страницы. Плюсом является независимость от мощности клиента, ведь он просто должен принять запрос со всеми данными. Однако это накладывает ограничение на мощность серверной части – она должна быстро обрабатывать запрос пользователя и сразу отдавать ему все нужные данные, чтобы данный подход имел смысл.

В моем случае Server-side Rendering будет предпочтительней, так как FastAPI поддерживает библиотеку для шаблонизированного написания клиентской части приложения – Jinja2, что очень ускорит разработку и поможет с легкостью поддерживать приложение даже специалистам, не обладающих высокими навыками Frontend разработки.

ТУТ ОПИСАНИЕ ЖИЖИ2 И ПОЧЕМУ SERVER-SIDE RULES