**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, 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.

**1.1.1 OTA (Over-the-Air) Update Systems**

Bosch IoT Suite

На официальном сайте Bosch IoT Suite можно найти актуальную информацию, предоставляющую общую картину о продукте [1].

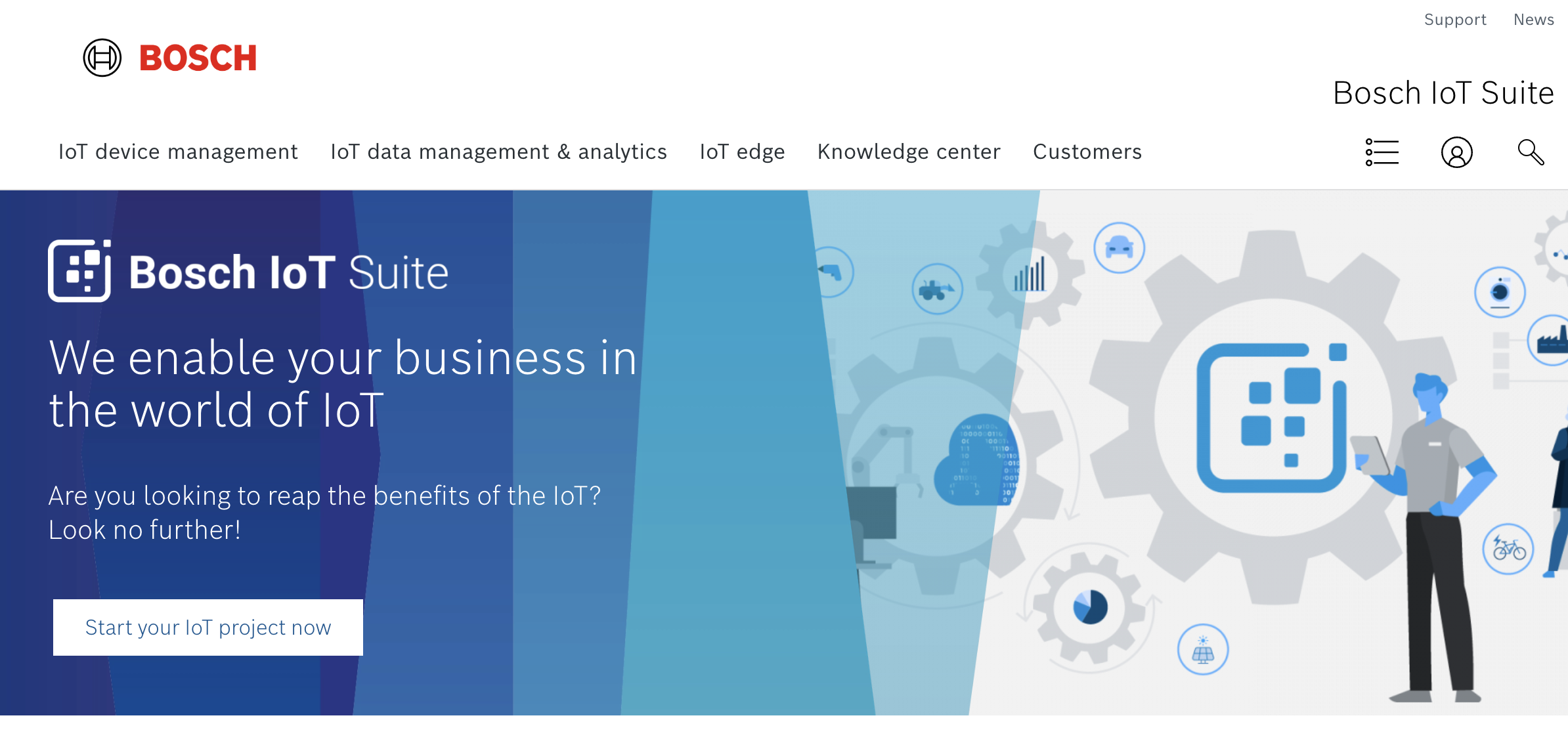


Рис 1.1. Официальный сайт Bosch IoT Suite.

Bosch IoT Suite — это семейство продуктов и услуг в области управления устройствами IoT, управления данными IoT и IoT edge от Bosch.

Bosch пишет, что IoT является прямой двусторонней связью с вашими устройствами, такими как подключенные автомобили, шлюзы в зданиях или на заводах и датчики в городской инфраструктуре. IoT позволяет в реальном времени понимать потребности клиентов, улучшать ваши услуги на лету и внедрять инновационные функции.

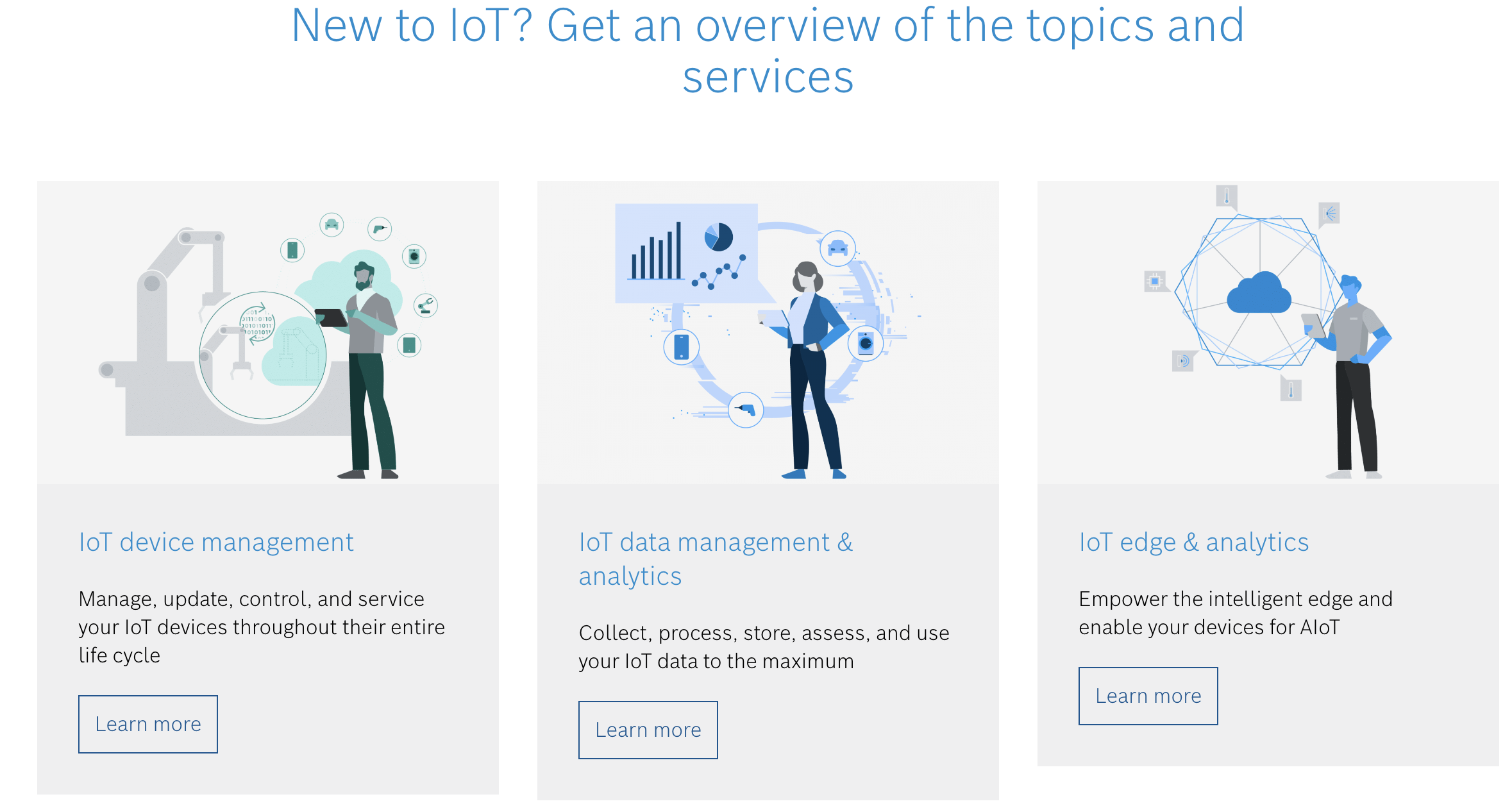


Рис 1.2. Решения Bosch для решений задач по теме IoT.

IoT device management признан решить задачи пользователей по актуализации ПО устройств и отслеживания их состояния [2].

Подключенные устройства IoT делают жизнь их пользователей проще, но для их эффективной работы требуется постоянный контроль. Будь это один датчик тревоги, большая группа умных автомобилей или датчики на производственном оборудовании, каждое устройство нуждается в управлении, обновлениях и техническом обслуживании, независимо от его местоположения. К управлению устройствами также относятся удаленное мониторинг состояния, настройка, управление большим количеством устройств, а также автоматические уведомления и диагностика при возникновении ошибок.

Подводя итоги, можно выделить преимущества Bosch IoT Suite:

* Широкий спектр возможностей
* Мировая компания, заимевшая огромную известность
* Хорошая система поддержки пользователей

Не обойтись и без недостатков решения:

* Поддержка закончится к середине 2024 года. Наверное, это самый критичный момент, который отталкивает от использования данного решения
* Высокая стоимость
* Зависимость от облачных технологий компании
* Большие затраты на внедрение и обслуживание системы – сложные технические процессы требуют регулярного контроля и высококлассных специалистов

**1.1.2 OTA (Over-the-Air) Update Systems**

Рассматривая уже готовые системы для решения поставленной задачи, не стоит упускать Over-the-Air Update Systems (Системы обновлений «по воздуху»).

Обновление по беспроводной сети (OTA) — это беспроводная доставка нового программного обеспечения, прошивки или других данных [3]. Изначально технология OTA стала знаменитой из-за роста количества мобильных устройств. Распространение Интернета вещей побудило производителей использовать OTA-обновления для автономных транспортных средств, колонок для умного дома и других устройств Интернета вещей. Производители обычно обновляют свои системы, используя нелицензионные диапазоны частот, например 868 МГц, и протоколы передачи данных с низкой скоростью, например 802.15.4.

OTA-обновления могут быть доставлены несколькими способами. С точки зрения конечного пользователя, обновление OTA может быть автоматическим или ручным. В сфере автомобильных СТО ручное обновление будет куда предпочтительней, ведь пользователи вряд ли захотят устанавливать дополнительное оборудование для обновления ПО своего автомобиля, привязывая себя к конкретному СТО и затрачивая дополнительные средства на эти мероприятия.

Рассмотрим решения для OTA обновлений на примере OTAdrive.

На официальном сайте OTAdrive можно получить актуальную информацию о продуктах, предлагаемых компанией [4].

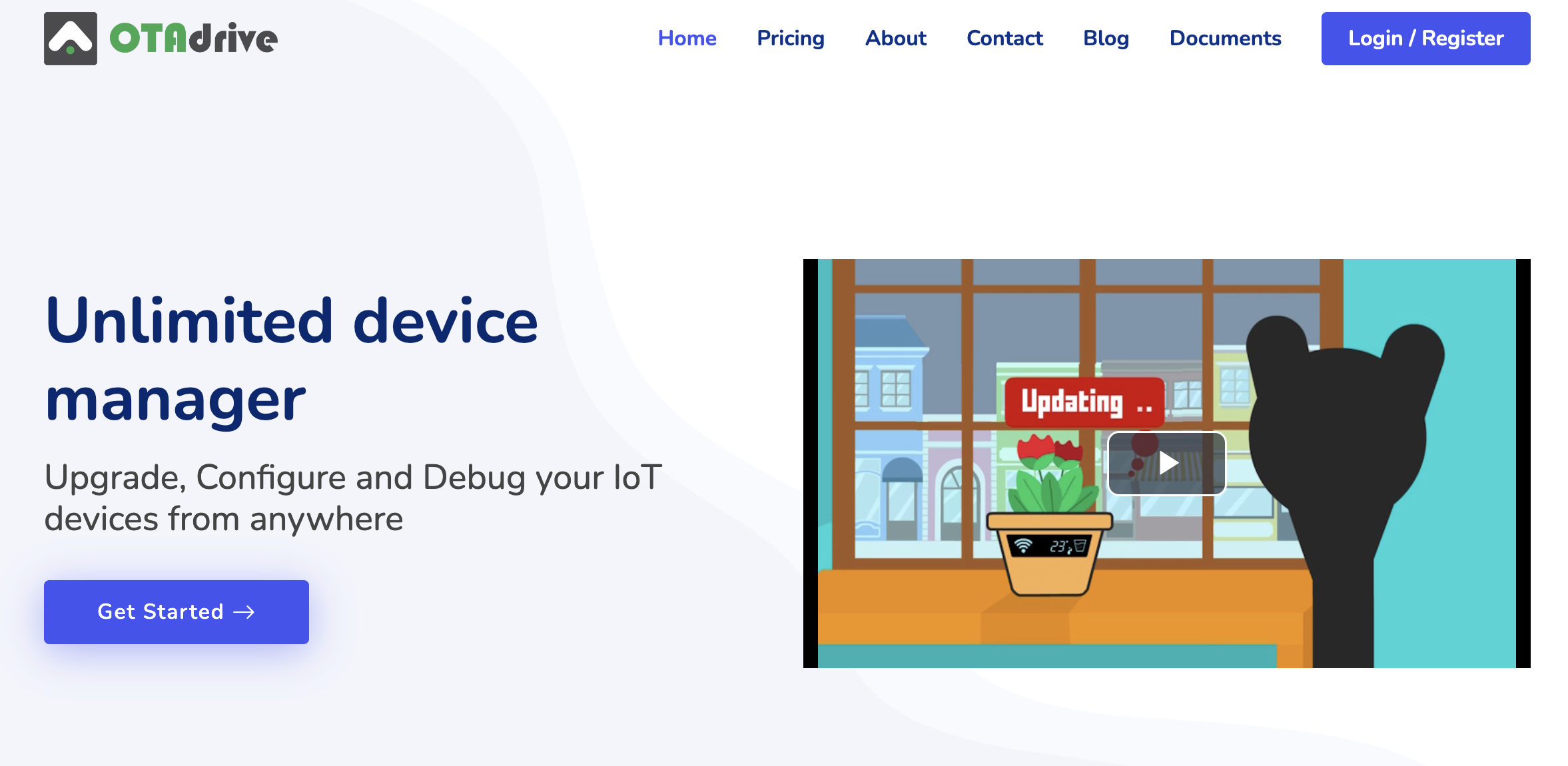


Рис 1.3. Официальный сайт OTAdrive.

На этой же странице мы можем увидеть показатели компании, в том числе – количество отзывов, количество продуктов, подключенных устройств и прошивок [4].

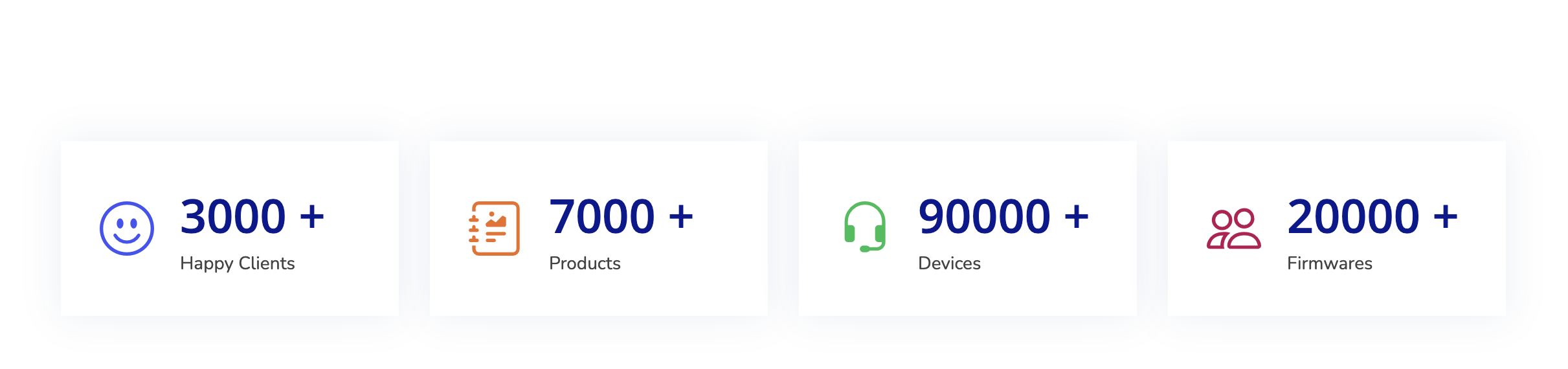


Рис 1.4. Показатели компании OTAdrive на начало 2024 года.

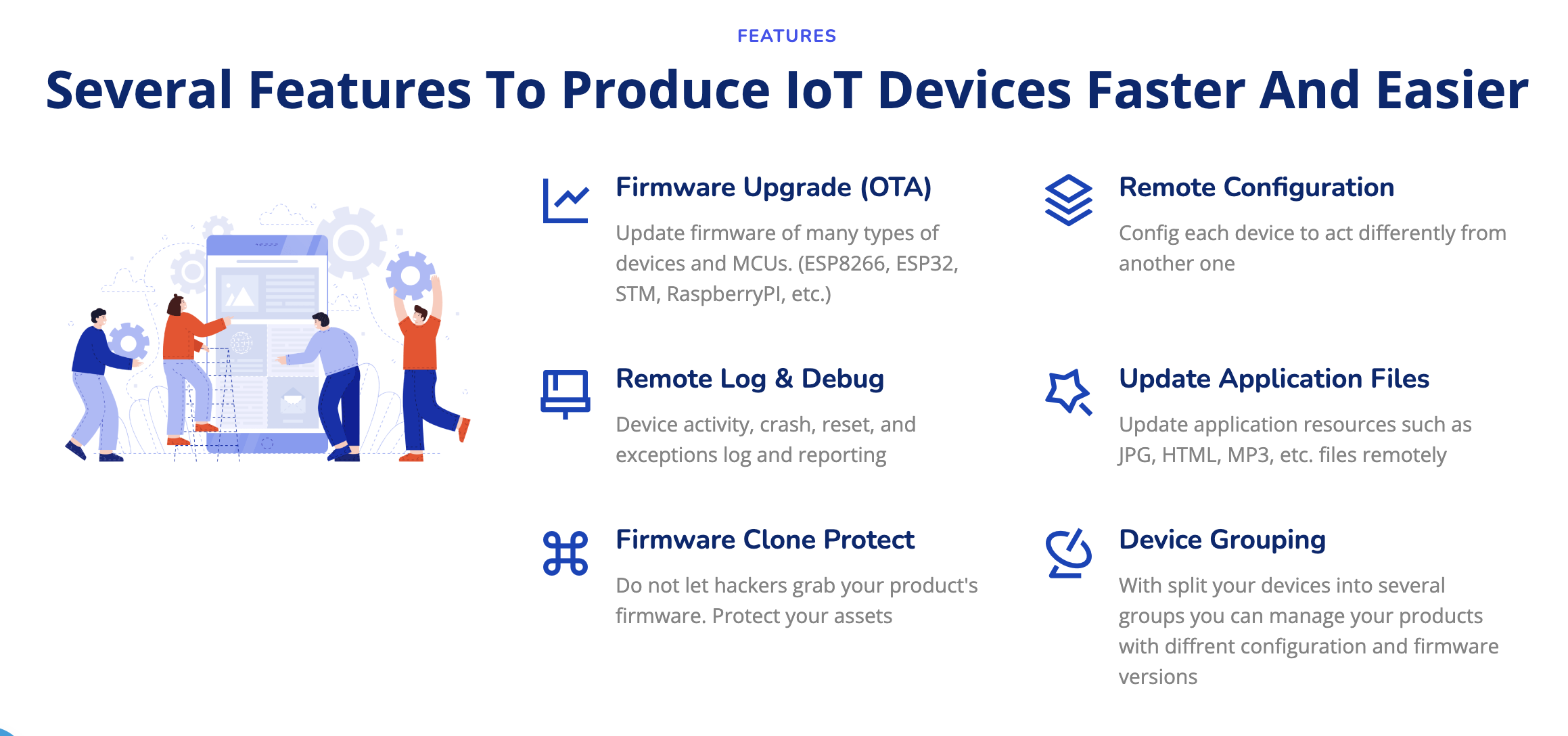


Рис 1.5. Представлений решений OTAdrive, представленных на официальном сайте.

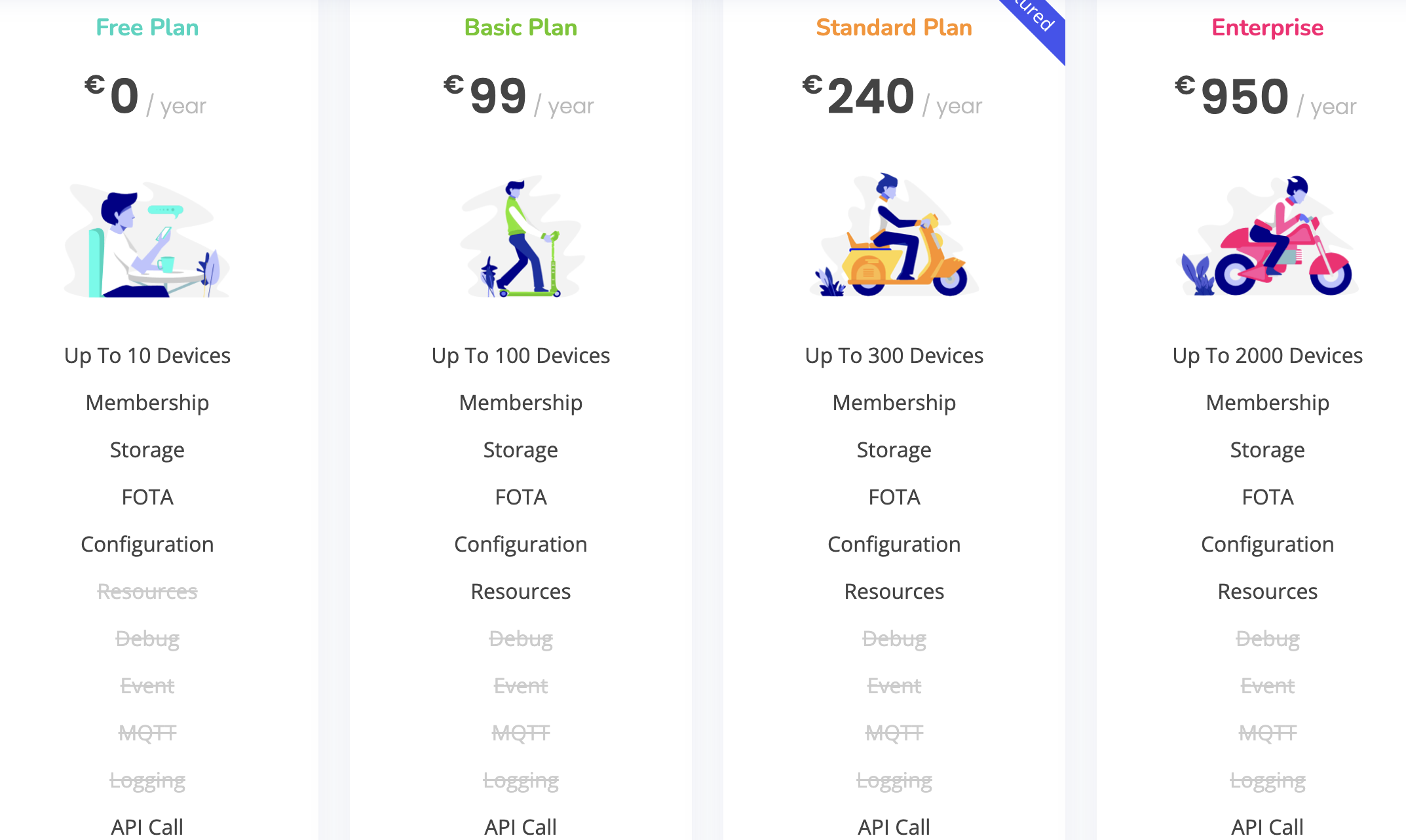


Рис 1.6. Расценки различных планов на услуги OTAdrive, представленный на официальном сайте.

Говоря о OTA в текущей задаче, чаще всего подразумевается именно FOTA (Прошивки «по воздуху»). В автомобильной промышленности обновления прошивки по беспроводной сети (FOTA) относятся к удаленному обновлению или обновлению прошивки автомобиля. Прошивка относится к программному обеспечению низкого уровня, которое управляет компонентами оборудования, от управления двигателем до систем безопасности. Обновления FOTA могут внести важные изменения для повышения производительности, безопасности и даже эффективности использования топлива [5].

В основе FOTA лежит централизованная система управления, которая позволяет автомобильным компаниям контролировать и управлять распространением обновлений. Эта система отслеживает версии прошивок всех устройств и обеспечивает отправку только необходимых обновлений для каждого конкретного автомобиля.

Каждый автомобиль оснащен встроенным ПО, которое управляет процессом приема и установки обновлений. Это ПО не только загружает и устанавливает обновления, но и проверяет их на совместимость и безопасность перед активацией.

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

**1.1.3 Vendor-Specific Update Tools**

Если говорить про систему обновления ПО встраиваемых устройств для автосервисов, стоит учитывать также официальные системы обновления вендоров. Это инструменты для обновления программного обеспечения, разработанные и поддерживаемые конкретными производителями автомобилей или оборудования. Эти инструменты предназначены специально для обновления встроенных систем и компонентов в автомобилях этого производителя. Использование таких инструментов имеет как преимущества, так и недостатки.

Примерами таких систем являются Ford SYNC, Tesla Software Updates.

Плюсы Vendor-Specific Update Tools:

* Оптимизированы под конкретное оборудование, эти инструменты разработаны специально для определенных моделей или марок, обеспечивая высокую совместимость и стабильность работы.
* Часто включают функции или опции, специально предназначенные для использования в конкретных моделях или системах, улучшая тем самым их функциональность и производительность.
* 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
* Вендорские решения часто предоставляю меньше возможностей для персонализации процесса обновления по сравнению с более универсальными или открытыми решениями

**1.1.4 Open Source Update Solutions**

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.

Примером такого инструмента можно назвать SWUpdate.

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.

**1.2 Требования к системе**

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.

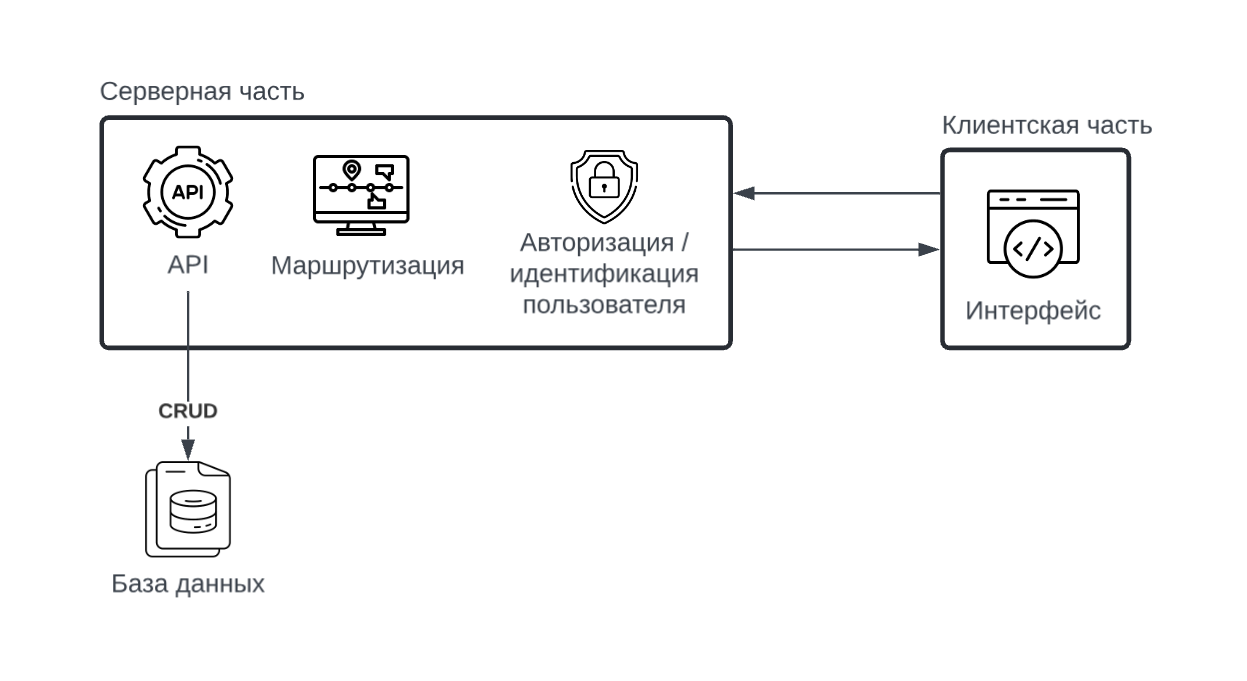


Рис. 1.7 Структура сервера обновлений

**1.3 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**
       1. **Язык программирования**

Начнем с выбора языка программирования для серверной части приложения. Python является одним из наиболее популярных языков программирования для разработки веб-сервисов [6].

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.

**1.3.1.2 Веб-фреймворки**

Python предлагает различные фреймворки для веб-разработки, каждый из которых имеет свои особенности, сильные и слабые стороны. Рассмотрим наиболее популярные фреймворки: Django, Flask, FastAPI.

**Django**

Django — это фреймворк на базе Python, разработанный по модели MTV (Model-Template-View). Основная задача Django — облегчить создание сложных веб-сайтов и приложений, работающих с базами данных. Фреймворк фокусируется на возможности интеграции и повторном использовании компонентов, минимальном объеме кода и ускорении процесса разработки [7].

Python широко используется с Django, в том числе и для работы с файлами и настройками. Фреймворк предоставляет удобный интерфейс для создания, чтения, обновления и удаления данных, а также включает функции самодиагностики, повышающие удобство работы.

Django также поддерживает SEO-оптимизацию, обеспечивая лучшую видимость сайтов в поисковых системах, и предлагает высокий уровень безопасности для пользователей. Благодаря своей универсальности и множеству полезных функций, Django помогает быстро развивать новые проекты.

Однако Django облагает и недостатками. Можно выделить архаичную архитектуру. Хотя Django и преследует модель MTV, структура приложения может превратиться в настоящий хаос, состоящий из сотен файлов, в котором невозможно будет разобраться.

Помимо этого, Django имеет плохую поддержку асинхронного кода. Независимо от того, что поддержка асинхронных «вьюх» уже добавлена в Django, многие функции до сих пор требуют перевода в асинхронность некими «костылями» [8].

**Flask**

Этот фреймворк известен как микрофреймворк из-за его легковесной структуры. Его ключевое преимущество перед конкурентами — высокий уровень кастомизации [9].

Flask предоставляет базовый набор функций, однако он разработан таким образом, что позволяет разработчикам интегрировать множество дополнительных библиотек и плагинов для расширения его возможностей. Среди возможных добавлений — расширения для валидации форм, объектно-реляционные мапперы, системы аутентификации и механизмы для обработки загрузок, а также другие инструменты, улучшающие функциональность.

Однако, первая настройка проекта на Flask может занять значительное время. В отличие от таких систем, как Django, где основные компоненты предоставляются "из коробки", данный фреймворк требует установки множества функциональных возможностей через сторонние расширения, что удлиняет начальный этап разработки.

Применение дополнительных модулей не всегда выгодно. Встречаются случаи, когда интегрированные библиотеки со временем перестают поддерживаться на рынке. Это может привести к необходимости либо самостоятельно поддерживать такие библиотеки, либо искать их замену, что увеличивает общую стоимость поддержки приложения.

Кроме того, использование сторонних модулей может вносить элементы неопределенности в систему безопасности. В процессе разработки существует риск случайного подключения вредоносного модуля, что может стать причиной серьезных проблем в проекте.

**FastAPI**

FastAPI является современным фреймворком для создания веб-API на Python, который быстро набирает популярность благодаря своим особенностям и производительности.

FastAPI построен на Starlette для обработки запросов, что делает его одним из самых быстрых веб-фреймворков на Python. Это особенно важно для приложений, требующих высокой производительности и малой задержки Фреймворк нативно поддерживает асинхронное программирование с использованием ключевых слов async и await, что позволяет легко реализовывать асинхронную обработку. FastAPI автоматически генерирует документацию (Swagger UI и ReDoc) для API, используя стандарты OpenAPI. Это значительно упрощает процесс создания и поддержки документации. Также, немаловажно будет уточнить, что FastAPI использует современные функции Python, такие как типизация с помощью Pydantic. Это обеспечивает более строгую проверку типов во время выполнения и улучшает интеграцию с современными IDE.

Несмотря на все это, FastAPI представлен на рынке сильно меньше его конкурентов и по тому имеет меньшее сообщество и меньше сторонних библиотек, однако поддержка FastAPI сторонники разработчиками растет, все больше команд начинают считаться с этим фреймворком и учитывать его при разработке решений для веб разработки.

Хотя у FastAPI нет такого богатого набора инструментов «из коробки», как у того же Django, он предоставляет множество решений для собственной реализации в своей документации, например, базовый модуль для авторизации с JWT токенами подробно расписан в документации к фреймворку [10].

По итогам исследования современных фреймворков, популярных на рынке, можно сказать, что Django является отличным выбором для крупных приложений с полным стеком функций "из коробки", обеспечивая широкие возможности для управления содержимым, пользователями и процессами. Flask предлагает больше гибкости и подходит для проектов, где разработчики хотят полный контроль над стеком технологий, однако это может потребовать большей интеграции сторонних компонентов. FastAPI, с другой стороны, является превосходным выбором для разработки современных веб-API, особенно когда требуется высокая производительность и асинхронная обработка. Его способность быстро генерировать документацию, поддержка современных стандартов Python и встроенная поддержка асинхронности делают его идеальным для проектов, которым необходимы эффективные и масштабируемые API.

FastAPI в моем проекте будет оптимальным выбором. Он обеспечит эффективность разработки и поддержки приложения, а также позволит легко адаптироваться к изменяющимся требованиям и технологиям. FastAPI идеально подходит для проектов, требующих высокой производительности и где важна скорость разработки без потери в качестве и безопасности.

**1.3.1.3 Графовые базы данных**

Графовые базы данных в последние годы значительно укрепили свои позиции на рынке благодаря их уникальной способности эффективно обрабатывать сложные запросы с интенсивными связями данных. Среди различных графовых БД, таких как Neo4j, OrientDB, ArangoDB и Titan, Neo4j выделяется своей производительностью, мощными возможностями и широким применением.

Neo4j является одной из самых популярных и широко используемых графовых баз данных. Она применяется в таких областях, как социальные сети, рекомендательные системы, мошенничество с финансовыми данными, управление сетями и многие другие. Основные преимущества Neo4j:

* Neo4j оптимизирована для работы с графами и обеспечивает высокую скорость обработки запросов, даже когда речь идет о больших объемах данных и сложных запросах.
* Neo4j поддерживает различные типы запросов и обладает мощными возможностями для моделирования данных.
* Богатый язык запросов Cypher – декларативный язык запросов, специально разработанный для графов, позволяет легко и интуитивно формулировать сложные запросы.
* Neo4j поддерживает как вертикальное, так и горизонтальное масштабирование, что важно для обработки растущих объемов данных.
* Обширное сообщество и доступность обучающих ресурсов обеспечивают поддержку на каждом этапе использования базы данных.

Для проекта по разработке системы обновления ПО для встраиваемых устройств, Neo4j предлагает ряд значительных преимуществ, таких как сложные связи между устройствами. Neo4j идеально подходит для управления сложными связями между устройствами, версиями ПО и их зависимостями. Это позволяет эффективно отслеживать, какие устройства нуждаются в обновлении и управлять распространением ПО. Помимо этого, Neo4j облегчает отслеживание различных версий ПО и их взаимосвязей, что критически важно для предотвращения ошибок при обновлении и гарантирования совместимости.

Использование Neo4j также позволяет встроить сложные проверки безопасности и аутентификации на уровне базы данных, что повышает общую защищенность системы, а гибкость Neo4j в моделировании данных и возможность легкого отката изменений помогают минимизировать риски, связанные с распространением некорректных обновлений.

* + 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:

**1.3.2.1 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.

**1.3.2.2 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 отлично подойдет смешанный подход – Server-side + Client-side Rendering, так как, используя серверную отрисовку мы можем быстро отдавать пользователю даже большие массивы данных, при этом не затрачивая дополнительные ресурсы клиентской части и не теряя время на лишних I/O операциях, а с помощью запросов с клиента мы можем динамически менять данные на странице без ее перезагрузки, что будет очень полезно, например, для получения значений для подсказок в полях форм (select’ы). 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.

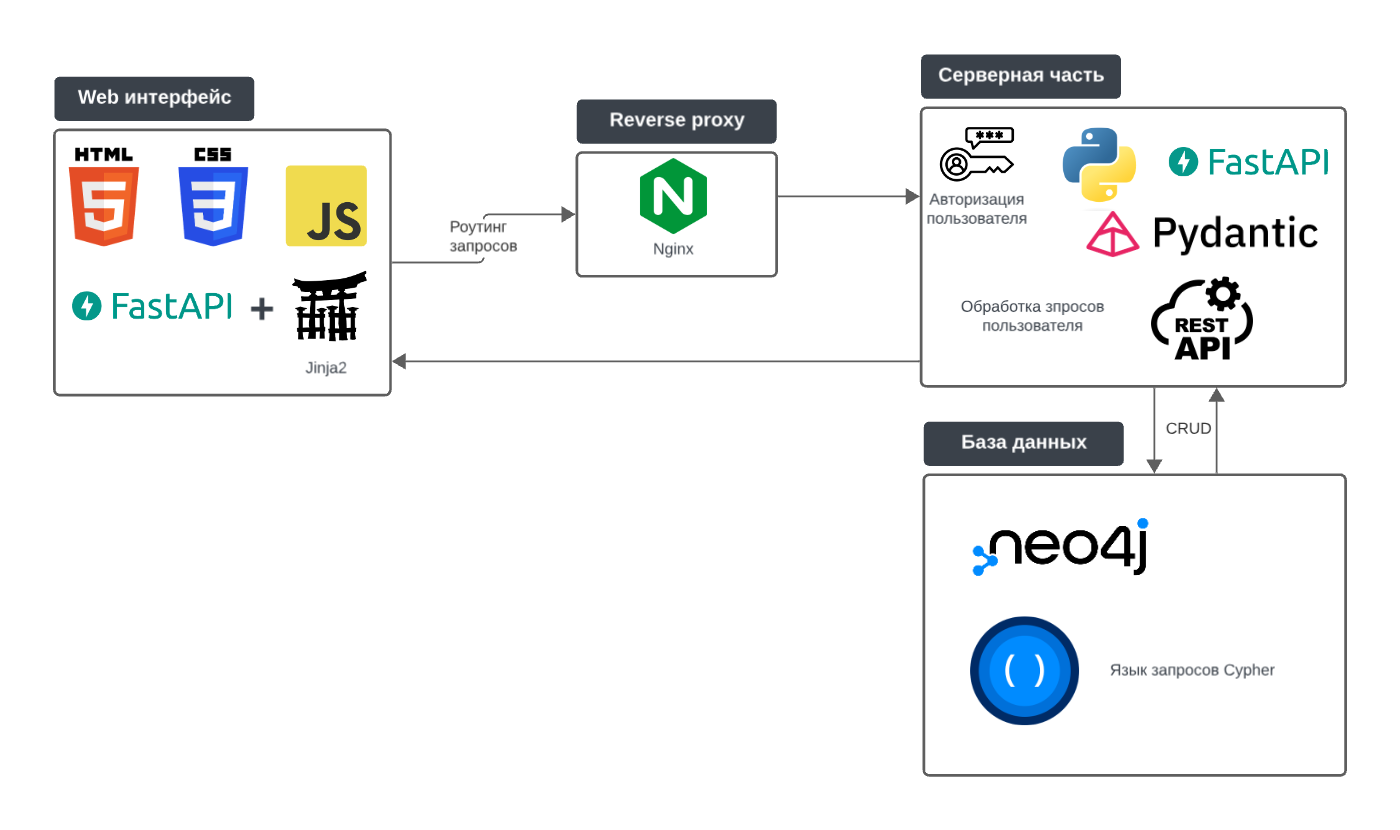


Рис 1.8. Структура приложения с выбранными технологиями

**Список источников информации**

1. Bosch IoT Suite официальная страница. <https://bosch-iot-suite.com/>
2. Bosch IoT Suite Device Management White Paper <https://bosch-iot-suite.com/knowledge-center/white-paper/iot-device-management/>
3. TechTarget OTA update systems <https://www.techtarget.com/searchmobilecomputing/definition/OTA-update-over-the-air-update>
4. OTAdrive официальная страница <https://otadrive.com>
5. CarRentalGateway глоссарий FOTA <https://www.carrentalgateway.com/glossary/fota/>
6. ITProger сравнение популярных языков программирования. <https://itproger.com/news/sravnenie-populyarnih-yazikov-programmirovaniya-dlya-veb-razrabotki>
7. Merehead. Лучшие фрейморвки 2024 <https://merehead.com/ru/blog/best-web-frameworks-2024/>
8. Django 5.0 Документация. Asynchronous support <https://docs.djangoproject.com/en/5.0/topics/async/>
9. Worksolutions. Flask vs Django <https://worksolutions.ru/useful/flask-vs-django-kakoj-web-frejmvork-vybrat/>
10. FastAPI документация <https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/>
11. Stackoverflow. Debouncing in Javascript. <https://stackoverflow.com/questions/75988682/debounce-in-javascript/75988895#75988895>
12. Jinja2 документация. <https://jinja.palletsprojects.com/en/3.1.x/>
13. Neo4j транзакции <https://neo4j.com/docs/python-manual/current/transactions/#explicit-transactions>