**СТО (Дырночкин А.А.)**

1. **Реализация всех задумок СРО в виде продуманной архитектуры проекта.**

В качестве архитектуры приложения мною была выбрана микросервисная архитектура, потому что она имеет следующие преимущества:

**Частичное развёртывание** Микросервисы позволяют по мере необходимости обновлять приложение по частям.

**Доступность. У** микросервисов доступность выше: даже если один из них сбоит, это не приводит к сбою всего приложения.

**Сохранение модульности.** Сохранять модульность и инкапсуляцию может быть непросто, несмотря на правила SOLID. Однако микросервисы позволяют гарантировать отсутствие общих состояний (shared state) между модулями.

**Мультиплатформенность** Микросервисы позволяют использовать разные технологии и языки, в соответствии с вашими задачами.

**Легкий порог вхождения**. Не требуется изучать структуру всего приложения, достаточно изучить структуру одного сервиса.

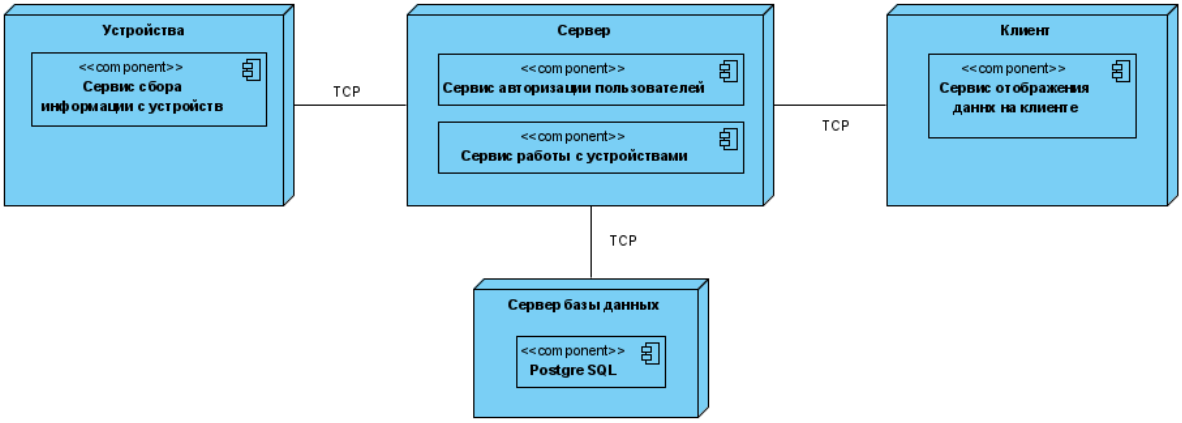


Рис. 1 Диаграмма развертывания

Пакет данных, получаемый с устройств сформирован в формате JSON. JSON – легко читаемый как человеком, так и компьютером формат обмена данными. Для внесения информации из такого пакета данных в базу данных требует его разбора. Приложение будет включать в себя класс разбора сообщений JSON формата.

Сервис сбора информации с устройств – микросервис, отвечающий за сбор информации с устройств и отправка этих данных на сервер, для последующей обработки.

Сервис авторизации пользователей – микросервис, отвечающий за авторизацию и регистрацию пользователей в системе.

Сервис работы с устройствами – микросервис отвечающий за обработку информации поступающей от устройств системы.

Сервис отображения данных на клиенте – микросервис для отображения данные на клиенте.

В базе данных будет храниться информация о пользователях системы, информация об их умных домах (у одного пользователя 1 или несколько умных домов, также у одного дома может быть несколько владельцев).

Также в базе данных будет храниться вся информация о доступных устройствах и конкретных параметров настройки для каждого умного дома.

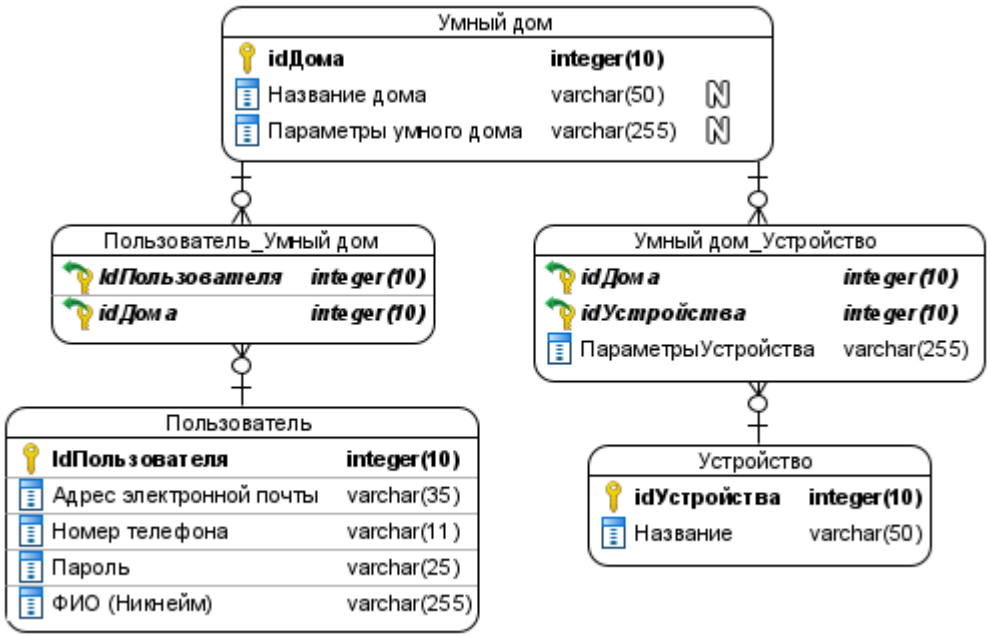


Рис 2. Диаграмма базы данных

## 2. Выбор технологий для проекта

### Выбор технологий для сервиса сбора информации с устройств

•C/C++. Объектно-ориентированный язык программирования, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных и т.д. Является одним из самых распространенных языков программирования, широко используется не только в разработке программного обеспечения, но и в драйверах разнообразных устройств. Хорошо подходит для программной среды Arduino, так как изначально язык программирования устройств Arduino основан на C/C++. На данный момент это самый удобный способ запрограммировать микроконтроллер.

•PERL. Высокоуровневый интерпретируемый динамический язык программирования общего назначения. Особенностью языка считается его богатые возможности работы с текстом, в том числе работа с регулярными

выражениями. Практичен и легок в использовании. Требует дополнительной конфигурации среды разработки.

•Python. Высокоуровневый язык программирования, ориентирован на повышение производительности разработчика, путем уменьшения синтаксиса. Python портирован практически на все платформы. Также, как и Perl требует дополнительной настройки. Конечные функции ограниченны.

В качестве языка программирования для умных устройств, будет использоваться язык C/C++, так как он неограничен в функционале, не урезаны библиотеки и не требует дополнительной настройки в среде разработки «Arduino».

**Выбор базы данных.**

В качестве базы данных была выбрана СУБД Postgres, т.к. бесплатная, есть много информации по настройке в интернете, есть опыт работы с ней.

**Выбор технологий для мобильного приложения**
**Kotlin**

Для разработки Android-приложений Google рекомендует использовать именно язык Kotlin, который появился только около 3 лет назад. Ожидается, что создание новых стандартных инструментов, например, библиотек, будет ориентировано на Kotlin.

Это язык с открытым исходным кодом и в нем собрано все лучшее из языков Java, Scala, TypeScript. В то же время Kotlin обладает рядом преимуществ. Среди них интуитивно понятный синтаксис и соблюдение последовательности, что улучшает производительность программистов. У Kotlin высокая совместимость с Java и его библиотеками. Правила создания кода помогают разработчикам избежать даже незначительных ошибок, которые сложно выявить до запуска программы, например таких, как NullPointerException. Язык обладает функциями расширения и автоматического выявления типов данных. Сторонники языка хвалят Kotlin за краткость, качество и читаемость.

Kotlin практически лишен недостатков, поэтому быстро набирает популярность в среде Android-разработчиков. На него уже перевели свои продукты Uber, Atlassian, Pinterest.

**Java**

Java признан языком официальной среды Android Studio, что дает доступ к огромному числу инструментов. Также для Java разработано много библиотек и руководств и документации.

Работа с Java требует высокого уровня абстракций и не терпит упрощений, что делает код длиннее и более громоздким, уменьшает производительность языка. При разработке продуктов следует помнить про исключения, из-за которых приложение может упасть, про конструкторы классов и прочее. С другой стороны, при соблюдении стандартов язык легко

читается и структурируется. К плюсам языка также стоит отнести автоматическое управление памятью, высокий уровень безопасности, многопоточность, портируемость.

**C/C++**

Низкоуровневые языки, которые позволяют писать нативные приложения, игры или другие ресурсоемкие программы.

C/C++ нельзя назвать удобными языками, они сложны в настройках, имеют громоздкие синтаксические конструкции. Их лучше использовать для написания отдельных модулей программы для сложных операций вроде обработки графики или видео.

К плюсам языков также относятся высокая производительность и универсальность. На данный момент у языков огромное сообщество, которое их поддерживает и развивает.

В итоге был выбран язык Kotlin, т.к. это довольно новый язык, который Google рекомендует для разработки android приложений, из-за чего в будущем под него будут писать все больше и больше новых библиотек. В то же время Kotlin имеет интуитивно понятный синтаксис и соблюдение последовательности, что упрощает и ускоряет написание кода. Также у Kotlin высокая совместимость с Java и его библиотеками.

**3. Релизы**

Будет использоваться GitLab/CI. CI — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ.

Основной целевой аудиторией будут являться пользователи из стран СНГ, поэтому релизы будут выпускаться в 01:00 – 03:00.

Бекапы будут создаваться перед обновлением, и храниться 90 дней, после будет происходить автоматическое удаление, и будут оставаться бекапы 1 и 15 числа каждого месяца

4. **Скрипт проведения собеседования**

**Требования к кандидату на вакансию «Разработчик микросервисов для взаимодействия с устройствами»**

- Опыт работы с умными устройствами от 1 года
- Знание языка C++
- Базовые знания ОС и СиТ
- Опыт работы с системой контроля версий Git
- Опыт интеграции систем
- Опыт работы с микросервисами

Будет плюсом:

- Знание SQL
- Опыт в мобильной разработке

**Требования к кандидату на вакансию «Android разработчик»**

- Опыт работы в андроид разработке 1,5 лет
- Знание языка Kotlin
- Знание SQL запросов
- Опыт работы с системой контроля версий Git
- Опыт работы с микросервисами

Будет плюсом:

- Опыт работы с умными домами
- Знание языка Java

**Скрипт:**

**1. Архитектура, паттерны**

- Какие паттерны знаете?
- Считаете ли вы Singleton антипаттерном?
- Опишите принципы MVP. Какие еще есть похожие MV*, в чем разница между ними?
- Объясните принцип DI
- Объясните принципы SOLID
- Объясните принципы Clean Architecture

**2. Android**

- Как можно выявить проблемы в скорости UI и устранить их?
- Какие проблемы были с использованием Dagger?
- Приходилось ли использовать Guard?
- Что такое multidex?
- Приходилось ли сталкиваться с миграцией с Dalvik на новую технологию ART?
- Начиная с какой версии пишете под Android? Какие были сложности с разницей версий?
- Асинхронные механизмы загрузки в Android
- В чем отличие AsyncTask от Thread?
- Минусы AsyncTask
- Опишите, что такое Activity

- Чем Fagment отличается от Activity?
- Разница между Service и IntentService. Пример использования Service.
- Зачем нужен Headless fragment (без View и с setReatinInstance = true)? Приходилось ли использовать?
- Какие новшества были в последней версии Android?
- Как определяете, какой layout надо использовать для смартфона, а какой для планшета?
- Как в коде определите: планшет это или смартфон?
- Пример использования BroadcastReceiver
- Опишите LifeCycle Activity
- Отличия Serializable и Parcelable

**3. Git**

- Разница между pull и fetch
- Разница между merge and rebase

**4. Прочие вопросы**

- Минусы использования сторонних библиотек
- Что вы будете делать, если ваше решение не совпадает с решением коллег или лида?
- Какие свои качества работы в команде вы можете описать?
- У вас есть команда, какие правила вы установите, чтобы писать тесты?