# Summary

This backend is design with 3 keys traits in mind.

Firstly, it is designed to be scalable as per the requirements its needs to be able to handle between 5-1000 clients this means it needs to be cost effective and cheap to run at 5 clients but also scale effectively up to 1000 making an application with pay as you go/by use features would be optimal.

Secondly, It also has be customizable this is best achievable with NoSQL structures that firebase provides such as Json or their collections system which although more complicated does provide higher querying capacities than Json as well as costing less to run.

Thirdly, support & maintenance, due to the implementation being over a short period of time then being handed off our team only will work on it for a small period of time and will not be able guarantee long term support therefore it is important that the backend relies on a service that can update with legacy support. This is also important for establishing connections between service and keep the product simple and understandable so it can be picked up by a different team.

# Overview

It use case is for a IoT system and so it needs to be able to store media and key data such as meta data, this would include things like its model, manufacturer and specification or any other chosen features the client desires. Installation details like its location the date it was installed, last recorded battery life, this could also include media such as images of the device in its setup in environments as well as its QR code. It will also need to be able to store documents to give more in depth details about the item and its setup this would include it certifications, software version, etc.

How can we achieve this?

We will be using Firebase who provide a multitude of service that would be appropriate for this product based on both technical requirements as well as conforming to other preferable qualities such as scalability requested by the client. We will need setup Firebase and configure it and layout a design plan for how our program will structure itself withy Firebase once this is complete the backend can start to be programmed. We will take a modular approach focussing on reusable functions that can take in a wide variety of arguments necessary for the programs flexibility while avoiding messy code. The primary features Firebase we will be using is the Cloud Firestore database, Firebase Cloud Storage for media, Firebase Authentication and messaging/notifications.

Scope and Objectives

* Scope:
  + Enable storage and management of IoT device metadata, media, and technical documentation.
  + Support user-defined database structures for custom fields and configurations.
  + Provide secure authentication and notifications.
* Objectives:
  + Create a scalable backend to support 5–1,000 users with minimal maintenance.
  + Ensure flexibility using Firestore's NoSQL structure and Firebase services.
  + Build reusable and modular functions for maintainable and adaptable code.

Proposed Methodologies

1. Agile Development:
   * Iterative development cycles to ensure continuous improvement and feedback.
   * Early delivery of key functionalities for client validation.
2. Modular Programming:
   * Develop reusable functions and APIs for CRUD operations, authentication, and storage.
   * Use parameterized methods for handling a variety of user inputs.
3. Cloud-Native Design:
   * Implement serverless functions via Firebase Functions for scalability.
   * Use Firebase Hosting for efficient and secure deployment.
4. Cost Optimization:
   * Design efficient database queries to minimize Firestore read/write costs.
   * Compress and optimize media files before storage.

# Work Break Down Structure

|  |  |  |
| --- | --- | --- |
| **Stage** | **Sub Stages** | Completion |
| **1.0 Project Planning** | 1.1 Requirements Gathering  1.2 Firebase Initial Config and Setup  1.3 Project Timeline Finalization | 90% |
| **2.0 Backend Development** | 2.1 Database Design   * 2.1.1 Define Collections and Documents * 2.1.2 Establish Indexing and Querying Rules   2.2 Backend API Development   * 2.2.1 CRUD Operations * 2.2.2 Modular Function Implementation * 2.2.3 Error Handling and Logging | 5% |
| **3.0 Feature Integration** | 3.1 Authentication   * 3.1.1 Configure Sign-In Options (Email/Password, Google Login, etc.) * 3.1.2 Implement MFA (if required)   3.2 Notification System   * 3.2.1 Integrate Firebase Cloud Messaging * 3.2.2 Develop Notification Triggers   3.3 Media and Document Storage   * 3.3.1 Implement Firebase Storage for Media * 3.3.2 Link Media and Documents to Firestore | 0% |
| **4.0 Testing and Optimization** | 4.1 Unit Testing  4.2 Performance Testing   * 4.2.1 Database Query Optimization * 4.2.2 Storage Cost Efficiency   4.3 Security Testing | 0% |
| **5.0 Deployment and Handover** | 5.1 Create Host for App  5.2 Provide Documentation for Handover  5.3 Client Training Session? | 0% |

# Backend Architecture

Database:

I chose Firestore over Firebases traditional Realtime Json database this is for a few reasons but the primary reason is its increased querying capability, Json is notoriously poor for making queries in but Firestore does not have this weakness as even though its Json like in structure it is also similar to SQL how each collection is like a mini table meaning advance queries can still be run with effective time and performance. Its scheme is like Json and is very flexible to accommodate different user priorities when adding devices as well as supporting subcategories allowing for users to sort and manage the database themselves without any need for more technically minded people to step in. Even though Firebases Realtime database is more suited to Realtime requests Firestore still supports and can perform to the project specification. This small disadvantage of being worse at realtime support is negated by how effective Firestore is for querying and over large data set operations and it also make more sense from a cost perspective as the ability to perform more advance queries increase efficiency and reduce the number of requests.

The structure looks like this:

Collection: devices

Document: deviceID123

- name: "Temperature Sensor"

- model: "TS-5000"

- manufacturer: "Acme Corp"

- location: { latitude: 40.7128, longitude: -74.0060 }

- installationDate: "2025-01-01"

- imageURL: "https://example.com/installation.jpg"

- technicalDocs: ["https://example.com/specs.pdf"]

Authentication – Needs further discussion with client as to the specifics of what they want depends if they want email/password, google, etc sign in even token-based sign on method as well as if they want MFA/2FA all supported and easily incorporated through firebase, though some play store features need a token to be setup.

John would prefer some kind of single sign on service like google but if it provides too much challenge then it can be ignored

Smaller connections linked through referenceId to establish connections between other external applications. As well as the ability to keep track of where data has come from when using other services and creating a link table almost like an audit log to track what has been submitted through these services.

# Cost

Firebase can be used in a restrictive free way, this is useful for small project/ use cases however if we want to use their ‘functions’ capabilities which allow for much more serverless control over how request are managed we would need the blaze plan their pay as you go solution.

Free vs Paid

A white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated





A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated(OPTIONAL)

Key points:

1. **Authentication**

* Free tier: 10K monthly active users (MAUs) are free.
* Beyond 10K MAUs: $0.01 per MAU.
* Predicted: £0/per month under our use case

1. **Firestore**

* Free tier:
  + 50,000 reads, 50,000 writes, and 1GB storage per month.
* Beyond free tier:
  + Reads: $0.06 per 100K.
  + Writes: $0.18 per 100K.
  + Storage: $0.18 per GB per month.
* Assumiming each user use is:
  + 10 reads/day
  + 5 writes/day
  + 0.1MB of data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Users | Reads/Month | Write/Month | Data Stored | Cost |
| 5 | 1,500 | 750 | 0.5 MB | $0 |
| 50 | 15,000 | 7,500 | 5 MB | $0 |
| 500 | 150,000 | 75,000 | 50 MB | $1.80 |
| 1000 | 300,00 | 150,000 | 100 MB | $3.60 |

1. **Firebase Storage**

* Free tier:
  + 1GB storage, 50K downloads per month.
* Beyond free tier:
  + Storage: $0.026 per GB per month.
  + Downloads: $0.12 per GB.
* Assuming
  + 1 image (500KB) uploaded/month
  + 3 files downloaded/month

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Users | Data Upload | Data Downloaded | Storage Cost | Download Cost | Total Cost |
| 5 | 2.5MB | 7.5MB | $0 | $0 | $0 |
| 50 | 25MB | 75MB | $0 | $0.01 | $0.01 |
| 500 | 250MB | 750MB | $0.01 | $0.09 | $0.10 |
| 1000 | 500MB | 1.5GB | $0.01 | $0.18 | $0.19 |

**Total Cost**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Users | Authentication | Firestore | Storage | Total |
| 5 | $0 | $0 | $0 | $0 |
| 50 | $0 | $0 | $0.01 | $0.01 |
| 500 | $0 | $1.80 | $0.10 | $1.90 |
| 1000 | $0 | $3.60 | $0.19 | $3.79 |

If we were to conisder function + firebases app hosting feature we woud be look at an additonal 38p a month, although these are all estimation and subject to change based on frequency of use these values were gathered assuming low use.