ChatGPT

Integrating Fitness Tracker Data into the GlyNa Prototype

Manual vs. Automated Activity Tracking

GlyNa's approach to incorporating exercise data can balance **manual input** with **automatic data sync from devices**. Since real-time streaming isn't critical, users can manually log their activity (e.g. steps or exercise duration) unless they own a compatible fitness tracker. This avoids heavy continuous data requirements while still capturing key info. If a user *does* have a smartwatch or pedometer that can sync data, the app can periodically import that data (say after a walk or at day's end) rather than requiring live updates. This way, **manual entry remains the default**, but those with devices get the convenience of automated logging. The synced data can then be visualized through in-app graphs or exported (e.g. as CSV) for analysis. Many fitness platforms even let users export their data in CSV form – for example, Fitbit lets users download daily steps, calories, sleep, etc., up to one month at a time (premium account needed for direct CSV) [1] [2]. Such data, once in GlyNa, can feed into trend graphs or combine with glucose info to tell a "story" of lifestyle impact.

Using the Smartphone as a Pedometer (Custom Solution)

One accessible, cost-free option is leveraging **smartphone sensors as a pedometer**, eliminating the need for any external device. Modern phones have built-in motion sensors and software APIs that count steps. For instance, Android provides a **Step Counter sensor** (`Sensor.TYPE_STEP_COUNTER`) which tracks the number of steps taken since device boot [3]. iPhones similarly use the Core Motion framework to count steps via the phone's accelerometer. In fact, Apple's HealthKit platform *automatically tracks steps* using the iPhone's sensors (and merges data from Apple Watch when available) [4]. This means even without a wearable, a phone carried in the pocket can act as a pedometer.

From an implementation standpoint, GlyNa could **"create our own" step-counting feature** for novelty and control. There are open-source algorithms available that demonstrate how to detect steps from accelerometer data. For example, researchers have developed an open-source step counting algorithm for the PineTime smartwatch, showing that an open approach can achieve accuracy comparable to proprietary solutions [5]. This kind of algorithm (based on peak detection in sensor signals [6]) could be adapted to run on a smartphone. However, we don't need to reinvent the wheel entirely – many developer libraries already wrap the phone's pedometer capabilities. In a cross-platform Flutter app, we can use existing plugins (such as the `pedometer` plugin) to access the **built-in step sensors on iOS/Android** for continuous step counting [7]. This gives us a ready-made, battery-efficient pedometer that logs steps locally.

Bottom line: The app can incorporate a custom pedometer mode using the phone itself – a highly accessible, zero-cost solution aligning with GlyNa's goal of working *"independently of expensive devices"* [8]. This provides novelty (our own implementation) while keeping things simple for users without any wearable.

Integrating with Health & Fitness Platforms (Fitness App Connections)

To support users who *do* have fitness trackers, GlyNa can connect to existing health platforms for automated data import. Rather than integrating each device from scratch, we prioritize linking with **popular health data platforms** which aggregate data from various wearables:

- **Apple HealthKit (iOS):** Apple's HealthKit is a secure, central repository on iPhones that stores health and fitness data from the phone and any paired Apple Watch or other apps [4]. With the user's permission, our app can read data like step count, walking/running distance, heart rate, etc. from HealthKit [9]. HealthKit already *automatically logs steps* from the iPhone and Apple Watch [4], and many third-party apps can contribute data to it. GlyNa can leverage this by reading the user's step totals and exercise minutes from HealthKit instead of requiring separate device APIs. Apple HealthKit is widely used and **free** for developers, and it comes built-in on iOS (used by millions of users) [10]. Apple's Health app even provides interactive charts for the user's data [4], and while that visualization is in Apple's app, GlyNa could similarly present the fetched data in its own dashboard.

- **Android Health Connect:** On Android, Google is moving toward **Health Connect** as the unified platform for fitness data. Health Connect (now integrated into Android 14+ and available via an app on older versions) acts as a centralized, on-device store for health and workout data [11] [12]. Various apps and device vendors can read/write to Health Connect with user consent. For instance, leading fitness apps like MyFitnessPal, Samsung Health, and even Fitbit have integration with Health Connect [13]. This means if a user has a Wear OS watch or a Fitbit device syncing to its app, those step counts can be shared into Health Connect. GlyNa can then read the step count or activity data from Health Connect's repository instead of dealing directly with each device's API. The benefit is similar to HealthKit: one integration gives access to many sources. Notably, Google Fit (the older Google fitness cloud API) is being deprecated in favor of this on-device model [14] [15], so focusing on Health Connect ensures forward-compatibility. Like HealthKit, Health Connect is **free and used by millions** (as it's becoming a default on Android) [10]. It keeps data local and private by design, which aligns with user privacy expectations [16].

By prioritizing these platforms, GlyNa stays **device-agnostic** – a user's steps could come from their phone, Apple Watch, a Galaxy Watch, a Fitbit, etc., but as long as those sync with Apple Health or Health Connect, our app can retrieve the data in one unified way. This approach satisfies the "latter" option of connecting to a fitness app/platform instead of a direct device link, simplifying development.

Wearable Devices with Open or Free APIs

For completeness and to accommodate other devices, we researched which **wearables offer open or free APIs** for third-party apps. We will prioritize solutions that don't lock users into paid services or proprietary barriers (open-source or free options), but also mention popular devices that may require extra steps:

- **Fitbit:** Fitbit provides a well-documented **Web API** that developers can use at no charge [17]. Once the user authorizes our app, we can programmatically request their Fitbit data (daily steps, active minutes, heart rate, etc.) via RESTful endpoints [17]. Fitbit's API has endpoints for daily summaries and even detailed intraday data (down to minute-by-minute steps or heart rate), though high-resolution data might require a special request/token from Fitbit [18]. The key point

is that **any developer can register an app and use Fitbit's API for free**, making it a friendly option for integration. For example, many researchers and hobbyists use Fitbit's API to pull their data into custom apps or dashboards [19]. The user would need to log in with their Fitbit account to grant permission, then GlyNa could regularly fetch their latest step count and exercise info. (Fitbit also allows direct CSV export of data via their website, but only with a Premium subscription [1] – using the API is a way around that limitation, letting us access data without the user paying for exports.)

- **Polar:** Polar (known for heart-rate focused wearables) has an **Open AccessLink API** that is free for developers. In 2017 Polar opened up their Polar Flow data platform via this API [20]. It allows read-access to a user's uploaded workouts and daily activity data (including step counts, calories, sleep, etc.) once the user authorizes the app [20] [21]. The Polar API uses a standard OAuth2 flow for authentication and returns data in JSON (workouts can be fetched in common formats like TCX/GPX) [22]. Notably, this is a **cloud-based API** (Polar Flow is Polar's cloud), so the user would need to sync their device with the Polar Flow app/cloud, and then GlyNa can pull from there. Polar's initiative to open their ecosystem is developer-friendly, especially since other major manufacturers historically had more restricted APIs [23]. In short, any user with a Polar watch or heart band can authorize GlyNa to fetch their data through Polar's free API.

- **Withings:** Withings (which makes smart scales and hybrid watches like Steel HR) offers a **public API** accessible to individuals and companies without special contracts [24]. Developers can register an application on Withings' developer portal and obtain credentials. Once a user logs in and consents, the API provides access to their data from Withings devices – including activity (steps, distance), heart rate, sleep, and body measurements. Withings's API is also OAuth2 based. This is a good *free API* option, though it's more relevant if users have Withings devices. (Note: Withings's own app can export data but with limits – e.g., weight CSV export only – so using their API can retrieve the fuller set of data [1].)

- **Open-Source Wearables:** A truly **open-source device** approach is possible for enthusiasts. For example, **Bangle.js** is an open, hackable smartwatch that runs JavaScript and comes with sensors (accelerometer, heart rate, etc.) and a pedometer app built-in [25]. It doesn't rely on any proprietary cloud – data can be accessed directly by connecting to the watch via Bluetooth. Similarly, **PineTime** (by Pine64) is an open-hardware smartwatch running open-source firmware. Developers can run custom step-count algorithms on it (as mentioned, PineTime was used in research to test an open algorithm [26]). These devices appeal to the open-source ethos: users can use them *without* vendor lock-in. In fact, projects like **Gadgetbridge** (an open-source Android app) enable connecting to devices like Pebble, Mi Band, Amazfit, and others **without the official apps or cloud** [27]. Gadgetbridge keeps all data local and private, and can collect sensor data from the device for the user [28]. For GlyNa, targeting Bangle.js or PineTime users might be niche, but it demonstrates our commitment to open solutions. If a user had one of these, we could, in theory, integrate by communicating with the device's open protocols or by reading the data Gadgetbridge saved on the phone. This isn't a mainstream need, but mentioning it earns novelty points and aligns with prioritizing open-source tools.

- **Garmin (and others):** Garmin is a popular brand for fitness watches, but unfortunately **Garmin's APIs are not readily open to individual developers**. Garmin does have a web API (Garmin Connect API), but it's officially limited to approved business partners and requires an application (they often reject personal/hobby use) [29] [30]. In fact, Garmin's developer program FAQ explicitly states no personal use – they want a company and a privacy policy to even consider approval [29]. There are no fees, but it's for "business use" only [31]. As a workaround, some users resort to third-party solutions: for example, Garmin can sync activities to Strava, and

one could then use Strava's API to get those activities (though Strava would only have workouts, not daily steps). Some open-source community tools (like **python-garminconnect** or **GarminDB**) exist to scrape one's own Garmin data by logging in programmatically [32], but those aren't official. For our purposes, we will **mention Garmin for completeness** but not rely on it due to these restrictions. If a user has a Garmin watch, the most feasible integration is if they connect their Garmin app to Apple Health or Google's Health Connect. Notably, Garmin's iOS app can sync data into Apple Health (steps, calories, etc.), and on Android one can use a tool like "Health Sync" to copy Garmin data to Google's platform [33]. Thus, we indirectly cover Garmin by leveraging the health platforms above, rather than a direct API. (Other brands like **Samsung** have their Samsung Health platform; Samsung Health does now integrate with Health Connect on Android [34], and on iOS, Samsung's wearables are really Wear OS watches now, or else they don't support iOS much. **Suunto** had an API but barely accessible to outsiders [35], and **Jawbone** is defunct. So, the major ones have been covered.)

In summary, **open or free APIs** we'd focus on are: **Fitbit, Polar, Withings** (widely used, easy API access) and support **standard platforms** (HealthKit/Health Connect) that cover Apple, Google, Samsung, etc. We also acknowledge truly open-source gadget options for the sake of novelty and completeness. By prioritizing these, we ensure that GlyNa can pull in activity data without forcing users to buy specific devices or pay for data access. Our app remains flexible: it works with a phone alone, but can plug into a user's device data if they have it – emphasizing the project's values of accessibility and personalization [8].

Prototyping the Integration (Leveraging n8n)

To rapidly implement and test these data integrations, we can use **n8n**, a no-code/low-code workflow automation tool. **n8n is open-source** and allows creating custom workflows that connect various services through a visual interface [36]. This is perfect for prototyping, especially since we want to experiment with pulling data from different APIs and perhaps sending it to our database or app.

Using n8n, we could for example: - **Connect to the Fitbit API** in a workflow: n8n has built-in support (via HTTP nodes or specific integrations) to authenticate with Fitbit and fetch data [37]. We could schedule an n8n workflow to retrieve a user's daily step count every evening and then push it to Firebase or a Google Sheet for analysis [38] [39]. This lets us test the end-to-end flow (from Fitbit to our app's datastore) without writing a bunch of code upfront. - **Integrate Google Fit/Health Connect or Apple Health:** While Apple HealthKit is on-device (so n8n might not directly pull from a phone easily), we can simulate by using Health Connect's data if it's exposed or use an intermediate like Google Fit's cloud (noting that the Google Fit REST API is deprecated, an n8n approach might instead use a companion Android app). Alternatively, n8n could be used to aggregate data from multiple sources: e.g., combine Fitbit + Strava + manual inputs into one place. In fact, one developer used n8n for a fitness side-project to connect **Strava, Fitbit, Google Fit, Telegram, Google Sheets, Notion** and more, automating things like sending motivational messages and logging workouts [36] [40]. This shows how powerful n8n can be in the fitness context. - **No-Code Glucose & Activity Nudges:** We could even use n8n to prototype the logic for GlyNa's "nudges." For instance, if we have meal data and step data available, an n8n workflow could trigger a Telegram or email message ("Reminder: take a 10-min walk!") if the user's post-meal glucose prediction is high and their step count is still low. This kind of **event-based automation** is a key feature of n8n [41]. It enables quick iteration on our idea of timely, data-driven nudges without having to build the entire backend from scratch during prototyping.

From a learning perspective, incorporating n8n is valuable because it will expose us to how these APIs work in practice, and we can **manually inspect the data flow**. Since n8n can run locally or in the cloud,

we have flexibility in testing. We can start it locally with Docker or npm (it's straightforward to set up [42]) and use the drag-and-drop interface to create workflows [43]. Given that the team is new to n8n, starting with a simple workflow like "fetch today's Fitbit steps and write to Firestore DB" would be an educational exercise. This aligns with the goal of learning n8n through hands-on use in our project's context.

Conclusion

In conclusion, GlyNa's prototype will support both **manual entry** and **device integration** for physical activity data, with an emphasis on accessible and open solutions. On one hand, the app can function with nothing more than the user's smartphone – tapping into built-in step counters or allowing quick manual logs – ensuring no barriers for users without fancy devices. On the other hand, for users who do have wearables, we'll integrate with major health platforms (Apple HealthKit, Android Health Connect) and directly with select device APIs (Fitbit, Polar, Withings, etc.) to automatically import their data. We've prioritized **open-source and free API options** so that our solution remains low-cost and flexible (no proprietary lock-in). Notably, while some popular devices like Garmin are less accessible to developers, we can still accommodate their users indirectly via the unified platforms.

By creating our own pedometer functionality and leveraging open APIs, we score points for novelty and uphold the project's ethos of affordability and personalization [8]. At the same time, using tools like **n8n** will accelerate our prototyping and allow us to experiment with these integrations quickly. The end result will be a GlyNa prototype that effectively "listens" to a user's lifestyle data – be it entered by hand or collected from a gadget – and converts it into meaningful insights and nudges (with graphs, stats, or CSV exports as needed). This robust yet flexible integration of activity data will enrich GlyNa's capability to guide diabetics in their daily journey toward better health.

Sources:

- Fitbit Web API – Developer FAQ (access to step count, etc.) [17] [44]
- MindSea Blog – Apple HealthKit and Android Health Connect integration [4] [13]
- VideoSDK Guide – Overview of fitness APIs (Fitbit, HealthKit, Garmin, etc.) [45] [46]
- DC Rainmaker – Polar Open AccessLink API announcement [21] [46]
- Quantified Self Forum – Discussion on wearable data exports (CSV, APIs) [1] [2]
- Open-Source Pedometer Research – PineTime wearable step algorithm [26] [5]
- Gadgetbridge F-Droid page – Open source app for wearables (no cloud needed) [27]
- Medium (L. Diaby) – Using n8n to integrate fitness services (Fitbit, Strava, etc.) [36] [40]

---

[1] [2] Which wearables have CSV exports? - Data Formats, Export and APIs - Quantified Self Forum
https://forum.quantifiedself.com/t/which-wearables-have-csv-exports/911

[3] Motion sensors | Sensors and location - Android Developers
https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion

[4] [10] [11] [12] [13] [15] [16] [33] [34] Apple Health, Google Fit: Integration Platforms for Health, Wellness, and Fi
https://mindsea.com/apple-health-android-health-connect-integration-platforms-for-health-wellness-and-fitness/

[5] [6] [26] Open source step counter algorithm for wearable devices
https://www.diva-portal.org/smash/get/diva2:1474455/FULLTEXT01.pdf

7 pedometer - Dart API docs - pub.dev
https://pub.dev/documentation/pedometer/latest/

8 GlyNa-Your-Daily-Diabetes-Companion (1).pdf
file://file-6EZWnGRmCy4MeAdk16LCJx

9 HealthKit | Apple Developer Documentation
https://developer.apple.com/documentation/healthkit

14 45 Fitness API: Ultimate Guide for Building Connected Health Applications - VideoSDK
https://www.videosdk.live/developer-hub/media-server/fitness-api-guide

17 18 19 44 Researchers FAQ – Fitbit Enterprise
https://fitbit.google/enterprise/researchers-faqs/

20 21 22 23 35 46 Polar Announces Open API for App Access to Polar Flow | DC Rainmaker
https://www.dcrainmaker.com/2017/10/polar-flow-api-accesslink.html

24 Public API - Withings
https://developer.withings.com/developer-guide/v3/withings-solutions/app-to-app-solution/

25 Bangle.js - Hackable Smart Watch
https://banglejs.com/

27 28 Gadgetbridge | F-Droid - Free and Open Source Android App Repository
https://f-droid.org/en/packages/nodomain.freeyourgadget.gadgetbridge/

29 30 32 Personal use of Garmin API : r/Garmin
https://www.reddit.com/r/Garmin/comments/114eh9y/personal_use_of_garmin_api/

31 Garmin Connect Developer Program FAQ
https://developer.garmin.com/gc-developer-program/program-faq/

36 38 39 40 41 42 43 Fitness & Challenges tracking with friends using n8n — Side Project (Part 1) | by Lamine DIABY | Medium
https://medium.com/@diaby.lamine/fitness-challenges-tracking-with-friends-using-n8n-side-project-part-1-d8fe4bc46910

37 Fitbit integrations | Workflow automation with n8n
https://n8n.io/integrations/fitbit/