**M2V2**: **Contributions by Shivani Bokka (Database Administrator)**

**3. Data Definitions**

**1. User**

Users are the people who interact with the app. Each person has a profile that includes their username, first and last name, email, gender, and a securely encrypted password. The system also tracks the date the user joined (created\_at). Users can create custom habits and participate in wellness circles.

**2. Habit**

This is an abstract table that serves as the general structure for all habits. It does not store any attributes itself but acts as a parent to both Custom\_Habit and Defined\_Habit. Its purpose is to provide a unified structure for different types of habits that can be tracked in the system.

**3. Custom\_Habit**

Custom habits are routines defined by users themselves. Each habit includes a descriptive name, the value recorded (like “2L” for water or “30 mins” for a workout), and the timestamp of when it was recorded. Every custom habit is tied to the user\_id of the person who created it. This model directly inherits from the abstract Habit class.

**4. Defined\_Habit**

Defined habits are preconfigured routines that come built into the app, such as “Drink Water” or “Morning Walk.” These habits inherit from the abstract Habit entity and do not contain any additional attributes. They offer users a quick-start option and promote uniformity across wellness circles.

**5. Wellnest\_Circle**

A Wellnest Circle is a group created to support shared health goals. Each group can have only one goal assigned to it. Users can join multiple wellness circles—either by creating one themselves or by joining a group started by another user.

**6. Joins**

This is an associative entity that links users to wellness circles. Since users can belong to many groups and groups can have many users, this table handles the many-to-many relationship cleanly.

**7. Leaderboard**

The leaderboard keeps track of each user’s score within a specific wellness circle. It records the total score for each user, the group it belongs to, and the last time the leaderboard was updated. This brings in a sense of progress and healthy competition among users.

**8. Streak**

Streaks monitor how consistently users stick to a habit. For each user-habit pair, we record the current streak length, the longest streak they've ever had, and when the streak was last updated. It rewards users for forming long-lasting routines.

**9. WaterIntake**

This table captures daily water consumption. For every log, it stores how much water was consumed (in milliliters), the user’s ID, and the timestamp of the entry.

**10. FoodIntake**

This tracks what the user eats and when. It records the calorie count, timestamp, and type of meal—whether it’s breakfast, lunch, or dinner. Each entry also ties back to the user who logged it.

**11. SleepLog**

Sleep logs help track rest. Each entry records the user’s ID, when they fell asleep, when they woke up, and the total number of hours slept. This data helps users monitor sleep patterns over time.

**12. WorkoutLog**

Workout logs track physical activity. For each session, it stores the user’s ID, how long they exercised, and when it happened. This supports habit tracking around fitness and physical wellbeing.

**------------------------------------------------------------------------------------------------------------------**

**6.1 High-Level Database Architecture**

**Initial Database Requirements**

Our app's backend database is designed to support a holistic wellness tracking platform. It allows users to log daily health activities such as water intake, food intake, sleep duration, and workouts. Additionally, users can create or join group wellness challenges called “Wellnest Circles,” where progress is tracked and visualized through a leaderboard.  
Each user can also define and track personal habits. The system supports both solo and group engagement, with no requirement for users to join a group in order to benefit from the app’s features. A streak system is also maintained to encourage consistency.

The database emphasizes:

* **Time-based logs** for accurate trend tracking
* **Normalized structure** for scalable and efficient data access
* **Foreign key constraints** to maintain referential integrity
* **Support for leaderboard logic** in a multi-user group setting

**DBMS Selection**

We selected **MySQL**, hosted on an **AWS EC2** instance, as our backend DBMS.  
MySQL offers robust support for relational modeling, foreign key enforcement, ACID compliance, and indexing, which are essential for managing user-specific logs and real-time group statistics.  
It integrates smoothly with Django ORM, making backend development more efficient and secure.

**Database Organization:**

Our database has ten main tables: User, Custom\_Habit, FoodIntake, Joins, Leaderboard, SleepLog, Streak, WaterIntake, Wellnest\_Circle, and WorkoutLog. These tables are connected using foreign keys to keep everything organized and related. Every log is saved with a timestamp (where necessary) so we can track progress over time in a clear and structured way.

**Entities, Attributes, Relationships, and Domains**

* User: Stores personal data and health metrics of each registered user. Acts as the central entity.
* Custom\_Habit & Defined\_Habit: Support user-defined and pre-defined habits respectively.
* WaterIntake, FoodIntake, WorkoutLog, SleepLog: Time-stamped logs of user health behavior.
* Wellnest\_Circle: Represents a group wellness challenge with a name, description, and creation time.
* Leaderboard: Tracks user scores in each group, supporting performance comparisons.
* Streak: Tracks a user's ongoing success in habit consistency.
* Joins: A bridge table linking users to the Wellnest Circles they've joined.

**Key relationships:**

* One User can have many logs (1-to-many).
* One User can belong to many Wellnest\_Circles (many-to-many via Joins).
* One Leaderboard entry uniquely combines a User and a Wellnest\_Circle (unique constraint).
* All logs and relational tables use foreign keys to maintain referential integrity with the User table and others where applicable.

|  |  |
| --- | --- |
| **Entity** | **Key Attributes** |
| User | id, first\_name, last\_name, gender, username, email, encrypted\_password, created\_at |
| Habit | abstract – inherited by subtypes |
| Custom\_Habit | id, user\_id, name, timestamp, recorded\_value |
| Defined\_Habit | inherits from Habit (no additional fields) |
| WaterIntake | id, user\_id, amount\_ml, timestamp |
| FoodIntake | id, user\_id, calories, type\_of\_meal, timestamp |
| WorkoutLog | id, user\_id, duration, timestamp |
| SleepLog | id, user\_id, duration, timestamp |
| Wellnest\_Circle | id, name, goal\_description, goal\_type, goal\_value |
| Joins | user\_id, group\_id — associative entity for User ↔ Wellnest\_Circle relationship |
| Streak | id, user\_id, habit\_id, current\_streak, longest\_streak, last\_updated |
| Leaderboard | id, user\_id, group\_id, score, last\_updated |

**Entity-Relationship Diagram (ERD)**

An ERD was created using [draw.io](https://draw.io) to represent the entities, their attributes, and how they relate. The diagram highlights the key functionalities of the system, such as user logging, group engagement, and progress tracking. Foreign key constraints ensure data integrity and enforce the rules of interaction between different parts of the database.

(INSERT ERD DIAGRAM HERE)

**Media Storage**

For this project, we do not plan to store or handle any user-uploaded media files such as images, video, audio, or GPS data. Instead, users will be able to select from a set of predefined profile icons. These icons are handled as static assets on the frontend, with only a reference (e.g., file name or ID) stored in the database. This approach keeps the system efficient while still providing basic personalization features.

**------------------------------------------------------------------------------------------------------------------**

**M2V2 – 6.2 Backend Architecture**

A diagram of a microservice

AI-generated content may be incorrect.**Scalability Diagram**

**Architecture Summary**

Our backend system is designed with future scalability, modularity, and security in mind, following microservices principles and anticipating production-level deployment needs. The current implementation lays the groundwork for future enhancements including containerization, replication, caching, and more robust load handling.

**Microservices Architecture**

We follow a microservices architecture, where different backend functionalities are separated into independently manageable services:

* **User Microservice**: Responsible for user registration, authentication, and profile management.
* **Group Microservice**: Handles creation and management of user groups.
* **Future Microservice**: Placeholder for upcoming features (e.g., notifications, analytics).

This design allows us to independently scale, test, and deploy each component as our needs evolve.

**Load Balancers (Planned for Future)**

Although not currently deployed, we plan to integrate **Nginx as a reverse proxy and load balancer**. In the future, this will:

* Distribute traffic evenly across microservice instances,
* Enable SSL termination (HTTPS),
* Enforce API security via session-based authentication,
* Optionally introduce rate limiting for abuse prevention.

For now, all requests are routed directly from the frontend to services through a basic Nginx proxy layer.

**Caching Strategies (Planned for Future)**

We plan to use **Redis** for:

* Session caching to minimize repeated authentication checks,
* Query caching to reduce database load on frequent read operations.

While Redis is represented in our architecture for planning purposes, it is not yet in active use.

**Reliability and Fault Tolerance (Planned with Containers)**

Currently, our services are deployed directly on an **AWS EC2 instance** without container orchestration. As we scale, we plan to use Docker containers to enhance reliability and fault tolerance by:

* Isolating services to prevent one failure from affecting others,
* Enabling automatic restarts of crashed containers,
* Supporting horizontal scaling by spinning up additional container instances.

These improvements will help us make the most of our existing cloud infrastructure and ensure the system can handle higher loads and failures gracefully.

**Containers (Planned with CI/CD Pipeline)**

Although our code is currently deployed manually or with limited automation, we are planning to use **Jenkins (running inside a Docker container)** to automate builds and deployment:

* GitHub → Jenkins (build & test) → Deployment server.
* This will enable faster CI/CD workflows and consistent deployments.

**Data Replication and Consistency (Planned for Scaling Reads)**

We plan to adopt a **primary-replica database architecture** using MySQL:

* The primary DB will handle writes,
* Read replicas will offload select-heavy operations to boost performance.

This setup is not yet implemented but is part of our scalability roadmap.

**Security Considerations**

We are currently implementing:

* SSL Termination via Nginx,
* Session-based authentication to validate API requests.

We’ve included rate limiting in the architecture diagram as an optional future enhancement to protect the system from misuse. We may also introduce token-based access or API keys for microservice-to-microservice communication when needed.

**UML Class Diagram**

(Insert UML Diagram here)

**------------------------------------------------------------------------------------------------------------------**

**M2V2 - 6.4 High-Level APIs and Main Algorithms**

1. **Overall API Architecture**

Our model is designed using a modular, service-oriented backend structure. The backend is divided into microservices such as the **User Service** and **Group Service**, each responsible for managing specific core functionalities like user data, group challenges, and habit tracking.

* **User Service APIs include:**
* Managing user profiles (creation, update, deletion)
* Logging health metrics and daily routines (water, food, sleep, workout)
* Retrieving personalized user summaries based on activity trends
* **Group Service APIs include:**
* Creating and managing wellness circles
* Allowing users to join and leave groups
* Tracking group-based challenges and progress
* Maintaining leaderboards for community motivation

This service split ensures scalability and clean code organization

1. **Search Functionality**

The system supports a user-facing search bar to look up predefined habits and Wellnest Circles. When a user types a keyword, the backend uses fuzzy matching and keyword normalization to retrieve results.

* **Search endpoint example:**  
  GET /api/search?query=walk  
  This returns a list of matching habit names or group titles such as “Morning Walk”, “Walk 5K Steps”.
* **Search logic:**
  + Normalize text (case-insensitive, trimmed input)
  + Search predefined habits first
  + Then search group names from the database using LIKE
  + Optionally, we can use simple Levenstein distance for fuzzy match improvement if needed

1. **Ranking Algorithm (Leaderboard)**

Leaderboards track user performance inside each Wellnest Circle and ranks participants based on their progress toward the circle’s goal.

* **Ranking endpoint example:**  
  GET /api/leaderboard/:circle\_id  
  This returns users in that group, sorted by their scores in descending order.

**:circle\_id** is a dynamic path parameter that should be replaced with the actual ID of the wellness circle that the user wants to look up

* **Ranking logic:**
  + Each user’s score is computed when a logged habit aligns with the group's goal
  + Higher consistency and longer streaks earn more points
  + Tiebreakers are resolved by last\_updated (more recent activity wins)

1. **Streak Algorithm**

A streak tracks how many consecutive days a user has completed a specific habit.

* **Tracked using:**  
  Streak(user\_id, habit\_id, current\_streak, longest\_streak, last\_updated)
* **Streak logic:**
  + If user logs a habit today → current\_streak += 1
  + If yesterday’s log is missing → current\_streak = 1
  + longest\_streak updates if current\_streak exceeds it

This encourages consistency and builds accountability in habit formation.

1. **Personalized Evaluation Algorithm**

This algorithm powers the "Your Progress" page in the app, where users can view a daily summary of their health routines.

* It aggregates daily logs for water, food, sleep, and workout
* The summary reflects progress toward goals and streak status
* Users can view their log entries for the current and previous days
* The data is fetched on demand—no background evaluation or auto-updates

This feature gives the user a clear and consolidated picture of their daily wellness activities without overwhelming them with detail.

1. **Rating System (Planned Future Enhancement)**

At this stage of our project, we haven’t implemented a formal rating feature. However, it’s something we’ve identified as a valuable addition for future development.

**For example:**

* Users could rate predefined habits and wellness circles (1–5 stars)
* These ratings could influence search results or recommendations
* Community-rated content can improve onboarding for new users

While it's not part of our current model, we plan to explore this feature to make the experience more interactive and community-driven.

1. **API Design Philosophy**

All APIs follow RESTful principles with clean and logical grouping of resources. The backend is structured using Django REST Framework, and endpoints are designed to be reusable, secure, and scalable.

* All data is validated and sanitized before database entry.
* Endpoints return JSON and support pagination where applicable.

**------------------------------------------------------------------------------------------------------------------**