

Contents

1	Architectural Diagram	1
1.1	1. Document Control	1
1.2	2. Overview	1
1.2.1	2.1 MVP Scope (Current Phase)	1
1.2.2	2.2 Future Phases	1
1.3	3. Context Diagram	2
1.4	4. Component Diagram	2
1.4.1	4.1 MVP Components (Current Phase)	2
1.4.2	4.2 Future Components (Post-MVP)	2
1.5	5. Deployment Diagram	2
1.6	6. Data Flow Diagram	2
1.6.1	6.1 MVP Data Flows (Current Phase)	2
1.6.2	6.2 Future Data Flows (Post-MVP)	3
1.7	7. Security Considerations	3
1.7.1	7.1 Role-Based Access Control (RBAC) & Security	3
1.7.2	7.2 Transactions	3
1.7.3	7.3 Build & Performance	3
1.8	8. Development Configuration	3
1.8.1	8.1 Database Accounts	3
1.8.2	8.2 Configuration Management	4

1 Architectural Diagram

1.1 1. Document Control

- **Version:** 2.1
- **Author:** Henry Huerta
- **Date:** 2025-11-3
- **Reviewers:** Prof. Arnold Lau, T.A. Sneha Bhandari

1.2 2. Overview

FitDB is a small, well-structured Flask + MySQL system emphasizing database design and auditable transactions.

1.2.1 2.1 MVP Scope (Current Phase)

To Start: - one gym location - front desk manager role enabled - basic user and member account management

To Address (MVP): - **User account creation** - Users can create accounts (username, email, password)
- **Member account registration** - Front desk managers can create member accounts - **Access card issuance** - Front desk managers can issue access cards to members - **Auditing** - All operations logged immutably

To Avoid (MVP): - Payroll and payment handling

Note: See MVP_SCOPE.md for detailed MVP clarification.

1.2.2 2.2 Future Phases

To Address (Post-MVP): - Bookings (session bookings, cancellations) - Trainer availability management
- Check-ins at gym locations - Equipment tracking and maintenance - Reporting and analytics dashboards

Extensibility Goal: - The schema and services are designed to scale to many gyms, trainers, members, and overlapping class sessions - No schema changes needed when adding future features

1.3 3. Context Diagram

Actors → System: Actors and their primary views/capabilities:

- Member / Plus Member (via Member portal):
 - See personal information
 - Book classes / View bookings (plus only)
- Trainer (via Trainer portal):
 - View class rosters
 - (Provide availability to manager — possibly added later)
- Manager (via Manager console):
 - Manage equipment inventory / allocations
 - Override class rosters
 - Ban members
- Admin (via Admin console):
 - Global configuration
 - Auditing

External systems (possibly added later): - AWS integrations

1.4 4. Component Diagram

1.4.1 4.1 MVP Components (Current Phase)

- **Auth** Flask: session auth with role/permission checks
- **Membership** (MVP): user accounts, member accounts, access cards
- **Audit**: append-only audit log via DB triggers

1.4.2 4.2 Future Components (Post-MVP)

- **Membership** (Full): membership plans, check-ins, member photos
 - **Scheduling**: trainers, class sessions, bookings
 - **Equipment**: inventory, maintenance status, session equipment allocations
 - **Reporting**: denormalized views (utilization, class fill, equipment demand)
-

1.5 5. Deployment Diagram

- **Dev**: Flask + MySQL
 - **Prod (optional)**: AWS
-

1.6 6. Data Flow Diagram

1.6.1 6.1 MVP Data Flows (Current Phase)

A) User creates account 1. User enters username, email, password 2. System validates uniqueness and encrypts password 3. System creates USER record + triggers write to audit log 4. Return account creation confirmation

B) Front desk manager creates member account 1. Front desk manager selects user (by username/ID) 2. Front desk manager selects membership plan and gym 3. System creates MEMBER record + assigns role + triggers write to audit log 4. Return member registration confirmation

C) Front desk manager issues access card 1. Front desk manager selects member 2. Front desk manager enters card UID and selects gym 3. System creates ACCESS_CARD record with “active” status + triggers write to audit log 4. Return access card issuance confirmation

1.6.2 6.2 Future Data Flows (Post-MVP)

D) Plus member books a class 1. Auth check based on role 2. Validate capacity and equipment constraints in a transaction 3. Insert booking + triggers write to an audit log 4. Return booking confirmation + reporting views update automatically

E) Manager publishes sessions (from trainer availability) 1. Manager chooses a class template + availability block 2. System creates class session (rows) for the window; validates equipment pool 3. Manager can override capacity or reassign trainer if conflicts arise 4. Manager can ban members if needed

F) Trainer updates availability 1. Trainer proposes weekly recurring blocks (e.g., Tue 10:00–12:00) 2. Conflicts checked against existing sessions 3. Changes recorded in the audit log

G) Equipment tracking and maintenance 1. Manager tracks equipment inventory 2. Manager tracks equipment maintenance 3. Manager tracks equipment allocations —

1.7 7. Security Considerations

- using MySQL roles (member → plus member inherits; trainer; manager; admin)
- try to counter SQL injection risks from dynamic SQL strings
- PII: hashed passwords and encrypt membership photo
- audit trail for all state-changing operations

1.7.1 7.1 Role-Based Access Control (RBAC) & Security

- app-level decorators AND MySQL roles: `member` → `plus_member` (inherits), `trainer`, `manager`, `front_desk` (subset of manager focused on check-ins), `floor_manager` (subset of manager focused on equipment), `admin`.
- use least privilege grants; sensitive tables via views/procedures where helpful.
- passwords: encrypted somehow; PII protected; member photos stored as paths
- audit: DB triggers on INSERT/UPDATE/DELETE to an append-only `AuditLog` table; updates to audit rows are blocked.
- SQL injection protection via parameterized queries

1.7.2 7.2 Transactions

- **booking:** atomic insert with capacity guard and equipment check; on failure → rollback with clear error.
- **publish sessions:** atomic generation of sessions from availability with equipment validation; rollback on conflict.

1.7.3 7.3 Build & Performance

- use `EXPLAIN/ANALYZE` to show query plans.
-

1.8 8. Development Configuration

1.8.1 8.1 Database Accounts

The build script (`sql/build.sql`) creates two database users: - `fitdb_admin@‘%’` (password: `change-me-admin`): Admin account for development with full privileges - `fitdb_app@‘%’` (password: `change-me`): Application account for Flask app database access

1.8.2 8.2 Configuration Management

- **MVP:** Database configuration via command-line arguments or environment variables
- **Post-MVP:** .env file support will be added for easier local development configuration