# Verification & Validation

***Objectives***

1. User Login Performance
    a. Objective: Verify that users can log in with valid credentials within 2 seconds 95% of the time and can successfully create an account if not already registered
    b. Linked Requirement: FR-1
2. Contextualized Fitness Plan Generation
    a. Objective: Validate that the conversational interface generates a personalized fitness plan within 30 seconds per session and requires ≤ 10 follow-up questions on average
    b. Linked Requirement: FR-2
3. Dashboard Data Accuracy & Updated Speed
    a. Objective: Confirm that the fitness dashboard updates metrics and workout snapshots within 2 seconds after data entry or workout completion
    b. Linked Requirement: FR-3
4. Video Upload & Computer Vision Feedback
    a. Objective: Ensure that exercise videos can be uploaded within 5 seconds 95% of the time, and the system provides a form score feedback within 30 seconds 85% of the time
    b. Linked Requirement: FR-6
5. System Accuracy in Form Analysis
    a. Objective: Verify that the computer vision Form Score algorithm produces results ≥85% agreement with trained human evaluators
    b. Linked Requirement: NFR-2
6. Chatbot Responsiveness and Correctness
    a. Objective: Validate that the AI coach responds to prompts of up to 5,000 words within 5 seconds 80% of the time, and generates responses that are safe, relevant, and actionable
    b. Linked Requirement: FR-4
7. System Usability and Navigation
    a. Objective: Ensure new users can complete main tasks (login, workout access, video upload, dashboard review) within ≤ 3 clicks per subsystem, with an average intuitive rating ≥ ¾
    b. Linked Requirement: NFR-1
8. Security and Data Protection
    a. Objective: Confirm that all user data (profiles, workouts, uploaded videos, LLM prompts) is protected against malicious inputs and unauthorized access, with 0 critical vulnerabilities detected
    b. Linked Requirement:
9. System Performance and Reliability
    a. Objective: Verify that the system maintains consistent processing speed for video analysis ≥20 FPS, handles multiple sessions without crashes, and reliably updates all user metrics
    b. Linked Requirement: FR-3, FR-6, NFR-2

***Prioritization***

| **Product Capability** | **Priority (High, Medium, Low)** | **Justification** |
|---|---|---|
| User authentication & profile management | **High** | If login/authentication fails, no other flows can be reached. Very high use impact and security impact. |
| Workout dashboard | **High** | Central to the app, high user impact and critical for demonstrating value of the product. |
| Workout snapshot & session orchestration | **High** | Directly tied to key user flows in the SDP. If this is broken, users cannot complete their primary task. |
| Video upload, storage and retrieval | **HIgh** | Hard dependency for CV and FormScore because it involves larger payloads, network variability, and privacy concerns. |
| Computer Vision FormScore algorithm | **High** | Hardest technical component, high risk with ML/CV tuning and central to the app. |
| Chatbot coaching & goal-setting | **High** | Core capability of the app, depends on prompt design, latency, and safety. High user impact and integration complexity. |
| System performance & latency | **High** | Non-functional but critical because poor performance or slow chatbot feedback will make the app feel broken even if functionally correct. |
| Security & privacy controls | **High** | Directly tied to NFRs and mishandling data or injections is high-severity risk. |
| Data persistence & consistency | **Medium** | Important for long-term value, technical risk is moderate, failures degrade trust over |

| | | |
|---|---|---|
| | | time. |
| Wearable / external data integration | **Medium** | Enhances personalization but is not required for MVP workflows. Has integration risks. |
| Usability refinements | **Medium** | Directly supports NFR-1, but basic flows will still work without it being perfect. |
| Logging, monitoring, and validation result archiving | **Medium** | Important for debugging, audits, and regression tracking. Medium risk but high long-term benefit. |
| Admin/maintenance tooling | **Low** | Useful for team operations but not required for initial user value or core demos. Low user impact and lower risk. |

***Full RVTM***

Link to spreadsheet (https://docs.google.com/spreadsheets/d/1LlwYnhGP1OuHE_82G8SioDMWBc-v7PQ38XRp5suAcwY/edit?usp=sharing)

***Test Approach***

Our overall testing strategy is layered. We will first start with automated unit tests, build up to integration and system/end-to-end tests, and then validate usability through user-acceptance testing. All of these aspects are tied back to the RVTM so that each FR/NFR has at least one corresponding test case.

**Unit, System, and Integration Testing**

- **Automated Unit Testing**
  - Implemented with Pytest in /tests/units.
  - Focus: individual modules in isolation such as FormScore calculation, API handlers, database access layer, LLM adapter, auth helpers
  - External dependencies like the LLM, AWS, CV model, and database are mocked so that units are deterministic.
  - All unit tests run automatically on every push through GitHub Actions as part of the CI pipeline.

- **Integration Testing**

    - Use Docker to spin up the backend API, database, and LLM mock together
    - Focus: cross-component behavior like login + token handling, workout snapshot updates, CV feedback request lifecycle, and logging of LLM conversations.
    - These integration tests verify that combined services satisfy the functional requirements in the RVTM such as login FR-1, dashboard FR-3, chatbot FR-4, and workout access FR-5
    - Integration tests run in CI on each push and on main-branch merges.

- **System/End-to-End Testing**
    - Full user journeys are exercised in a deployed environment (Docker locally and AWS for staging):
        - New user registration -> chatbot goal setting -> dashboard view -> workout snapshot interaction -> video upload for form feedback.
    - System tests also include the pilot/prototype tests described in the Validation Plan and later User Acceptance Testing with simulated users.
    - These are primarily manual or semi-structured tests, with checklists that trace back to RVTM test cases TC-1.0–TC-1.8.

**Manual vs. Automated Tests**

- **Automated:**
    - Pytest unit tests.
    - Postman/Newman API and workflow tests.
    - CI pipeline checks such as build, test, Docker image build, and basic health checks
    - Repeatable performance checks can be scripted and run in CI or as scheduled jobs.
- **Manual/Semi-Automated:**
    - Pilot tests, observing real users performing exercises, and collecting qualitative feedback.
    - Usability testing such as think-aloud navigation sessions and click-count metrics tied to NFR-1
    - Exploratory security testing beyond scripted malicious inputs such as trying unexpected flows.

**Testing Non-Functional Requirements**

- Usability
    - Conduct guided navigation sessions where users verbally describe their experience; record “Intuitive Rating” (1–5) and number of clicks to reach key subsystems.
    - Target is for an average rating over 3.0 and following the click ceilings per subsystem as defined in TC-1.6.
- Accuracy

        - For CV/FormScore we will compare system FormScore outputs against a “trained person” baseline across a fixed set of exercises and require at least 85% agreement as in TC-1.7.
        - For the LLM we will use curated prompt sets like normal + edge case prompts and verify response quality and correctness manually and via rule-based checks such as no unsafe advice.
    - Performance & Reliability
        - Measure CV pipeline throughput and end-to-end feedback time with upload ≤ 5s, feedback ≤ 30s for target exercise
        - Measure chatbot latency with aim being ≤ 5s for prompts up to 5k words, ≥ 80% of the time and dashboard load time with ≤ 2s in typical scenarios, which will be consistent with FR-3, FR-4, and FR-6 test cases.
    - Security
        - Use a library of malicious inputs like token tampering to test all input paths such as forms, APIs, chatbot.
        - Verify that sensitive user data like profiles, workout history, prompts are encrypted and not exposed, and record vulnerabilities in a binary fashion like in TC-1.8.

**Regression Testing Approach**

- All automated unit and integration tests run via GitHub Actions on every push and pull request. A build cannot be merged if tests or security checks fail.
- When a bug is found we first add a failing unit/integration test that reproduces it, then fix the code, turning that test into a permanent regression guard.
- Postman/Newman collections and performance/security test datasets are versioned in GitHub, and CI stores results
- JIRA is used to track defects, link them back to RVTM requirements and test cases, and ensure that closing a ticket is contingent on passing regression tests associated with that requirement.
- Before any major release we will run a full regression suite with all unit tests, integration tests, core system test scripts, and a set of non-functional checks to ensure that changes have not broken existing capabilities.

***Validation Plan***

To ensure the application meets user needs and performs reliably under realistic conditions, the validation strategy will follow an iterative, feedback-driven approach. Validation activities will occur at multiple points throughout development rather than only at final release. This plan focuses on Pilot/Prototype Testing Procedures and User Acceptance Testing (adapted for simulated users).

**Pilot/Prototype Testing Procedures**

Pilot testing will focus on validating interaction between the computer vision subsystem, the chatbot interface, and feedback flow. This will occur using early prototypes and small-scale human trials (1-3 volunteer users). The purpose is to confirm that the application behaves as expected in a real environment, including camera setup, lighting, movement speed, and user interaction flow.

***Process Steps:***

1. Conduct initial pilot tests with the basic prototype
    a. Users perform a single exercise while standing in front of the camera
    b. System detects movement and sends feedback
2. Observe usability factors during pilots, such as:
    a. How easily users position themselves
    b. Whether feedback appears at appropriate times
    c. Whether the chatbot responds naturally/in the correct sense to questions
3. Record pilot issues (such as slow detection, confusing prompts, misclassification)
4. Refine pose detection and score outputs, latency handling and UI prompts based on findings
5. Conduct follow-up pilots to verify that issues are resolved
6. Continue pilot testing through multiple development iterations, adding more exercises and interaction flows as the system matures

***Success Metrics:***

- Average processing speed ≥ 20 frames per second (FPS)
- Successful task completion rate ≥ 85% (user can start a workout and receive feedback without external help)
- Reduction of major usability issues by ≥ 75% from the first to final pilot round
- Zero critical failures (crashes, frozen feedback loops, etc.) during pilot sessions

**<u>User Acceptance Testing (UAT) Approach</u>**

Because building a large, real user base is going to be challenging due to limited time and resources, UAT will be conducted using simulated user interactions and recorded exercise scenarios. These scenarios are designed to represent realistic workout conditions and common user behaviors. This allows us to evaluate system correctness, feedback quality, and reliability without requiring many live participants.

***Process Steps***

1. Develop a library of prerecorded workout videos (squats, lunges, push-ups, overhead press) performed with both correct and incorrect technique.
2. Run these videos through the AI model to test:
    a. Pose detection accuracy
    b. Identification of form errors
    c. Clarity and relevance of feedback from the NLP component
3. Compare the system's feedback against a predefined "gold standard" created with input from peers familiar with exercise form.

4. Conduct at least one expert review session (such as fitness-experienced student or instructor) to evaluate whether the feedback is safe, useful, and aligned with real coaching expectations.
5. Document discrepancies and implement model refinements.
6. Repeat the scenario tests after each major update to evaluate improvements.

***Success Metrics***

- Pose estimation accuracy ≥ 85% on simulated movement scenarios
- Feedback correctness ≥ 80% agreement with predefined “correct” answers or feedback
- Error detection consistency ≥ 90% across repeated runs
- Expert approval rating ≥ ⅘ for clarity and usefulness feedback

**Validation Approach**

Validation activities will be embedded throughout development, following the continuous cycle of prototyping, simulated testing, feedback review, refinement, pilot testing, and re-evaluating. The early stages are going to prioritize interface clarity and baseline pose detection, the middle stages refine error detection logic and NLP-based feedback, and the later stages will verify performance, safety, and usability. Feedback from simulated users, pilot participants, and experts will guide ongoing adjustments. This approach ensures that the application evolves to meet user needs, supports accurate and safe exercise feedback, and performs reliably under realistic workout conditions.

***Tools, Environments & Test Data Sets***

*Resources*

- Testing Tools
    - Pytest
    - Postman
    - CI/CD Pipeline (GitHub Actions)
- Simulation/Emulation Environments
    - Docker
    - AWS
- Data Sets for Validation
    - Sample Conversational Interface Conversations
    - CV Dataset(s)
    - Security Malicious Input Datasets
- Version Control and Test Management Tools
    - Github
    - Github Actions
    - JIRA

*Tool Configuration*

- Testing Tools
    - Pytest

        - All the unit tests will be located in the /tests/units directory (see GitHub repo for tree)
        - The test suite will run on every push via Github Actions and test functionality to ensure requirements are being met
    - Postman
        - Will store JSON collections for testing in /tests/postman/ directory
        - Will utilize Newman within GitHub Actions to run the API framework tests
    - CI/CD Pipeline (GitHub Actions) Configured To…
        - Install necessary dependencies
        - Run test suite for unit testing
        - Run Postman API Tests
        - Build Docker images
        - Push to AWS
- Simulation/Emulation Tools
    - Docker for local emulation of…
        - API backend
        - Database
        - LLM mock
    - AWS
        - S3 Bucket for Video Storage
        - DynamoDB Table for user tables, workout history table, and other relevant data storage
        - ECS Cluster/Service for backend production and URL accessibility
- Data Sets for Validation
    - Sample Conversational Interface Conversations
        - Sample user prompts dataset for LLM latency and response validation
        - Edge case prompt set to test LLM prompt limits an system response
    - CV Dataset(s)
        - Dataset of number of target exercises to set as “correct form” for CV comparison for FormScore algorithm test
        - Anonymous videos of one-rep exercises to validate FormScore algorithm and validate CV minimum accuracy
    - Security Malicious Input Datasets
        - Set of Malicious Inputs such as SQL injection, Token Tampering, Prompt Injection, etc.
- Version Control and Test Management Tools
    - Github
        - Source control, branching, PR review
    - Github Actions
        - Pytest suite
        - Newman tests
        - Security Check
        - Builds and Deploys to AWS

        - Store results as GitHub Actions artifacts and test logs stored at /validation/archive/<val_num>
    - JIRA
        - Utilize Epics, Sorties, Tasks, Tests and Bugs via a  Dashboard to isolate user roles and keep track of current failures/defects/validation issues

***Updated GANTT Chart***
*https://docs.google.com/spreadsheets/d/1jqt0bqbJDMeF0s4qPc-QQ24W8nhqlXgvxYtmo_-P1sQ/edit?usp=sharing*