**TP1 Test Cases**

**Team <n>**  
**Purpose:** Define automated tests (unit + integration + semi-automated UI) that verify all input validation and required behaviors for TP1. Each case states **what** is tested and **how** it is tested. These build on and extend the provided tests (e.g., the style of PasswordEvaluationTestingAutomation).

**1) Test Strategy & Scope**

**• Levels:**

**◦ Unit tests:** Validation helpers, controller validation gates, database uniqueness checks.

**◦ Integration tests:** Controller ↔ Model ↔ Database flows (create user, update user, role dispatch).

**◦ UI/Semi-automated checks:** Form wiring, error message display, disabled/enabled actions, happy paths.

**• Coverage Goals:**

◦ 100% of validation rules from the Input Validation Plan.

◦ Positive and negative paths for every input field.

◦ Boundary values, Unicode/whitespace, duplicate/exists checks, security hardening (no processing on invalid).

**2) Naming Conventions (JUnit examples)**

**• Unit (validators):** ValidationTest\_isEmail\_validCases(), ValidationTest\_isStrongPassword\_rejectCommon()

**• Controller (integration):** NewAccountControllerTest\_createUser\_success(), UserUpdateControllerTest\_changeEmail\_duplicateFails()

**• UI wiring (semi-automated):** LoginUITest\_invalidEmail\_showsInlineError()

**3) Traceability Matrix (Validators → Test IDs)**

• isNonEmpty → **VAL-001**, **VAL-002**

• maxLen → **VAL-010**

• isEmail → **VAL-020** … **VAL-026**

• isStrongPassword → **PWD-100** … **PWD-108**

• isUsername → **USR-050** … **USR-055**

• isPhone → **PHN-070** … **PHN-074**

• isInSet → **SET-090** … **SET-093**

• isPositiveInt → **NUM-110** … **NUM-112**

• isSafeFreeText → **TXT-130** … **TXT-134**

**• Uniqueness checks** (username/email) → **DB-200** … **DB-206**

**4) Test Cases**

**Format:**  
**ID** – **Title**  
**Feature** | **Preconditions** | **Inputs** | **Expected** | **How Tested**

**4.1 Validation Helpers (Unit)**

**VAL-001 – NonEmpty accepts normal text**  
Feature: isNonEmpty  
Pre: none  
Inputs: "abc"  
Expected: true  
How: JUnit unit test

**VAL-002 – NonEmpty rejects null/blank**  
Feature: isNonEmpty  
Pre: none  
Inputs: null, "", " "  
Expected: false  
How: JUnit unit test

**VAL-010 – maxLen trims and enforces**  
Feature: maxLen(s,40)  
Pre: none  
Inputs: "a"\*40 → true, "a"\*41 → false, " a " → true  
Expected: boolean as above  
How: JUnit

**VAL-020 – isEmail accepts common valid**  
Feature: isEmail  
Pre: none  
Inputs: user@example.com, u.x+tag@sub.domain.org  
Expected: true  
How: JUnit

**VAL-021 – isEmail rejects invalid forms**  
Feature: isEmail  
Inputs: user@domain (no TLD), user@.com, user@@x.com, user x@y.com  
Expected: false  
How: JUnit

**VAL-026 – isEmail handles Unicode local**  
Feature: isEmail  
Inputs: olé+tag@ejemplo.es  
Expected: true (if allowed by subset) or clearly documented false  
How: JUnit (document rule decision)

**USR-050 – isUsername valid set**  
Feature: isUsername  
Inputs: alice, a\_b, a.b2, Alice\_01  
Expected: true  
How: JUnit

**USR-051 – isUsername invalid**  
Feature: isUsername  
Inputs: \_alice (no leading letter), a..b (double dot), a (too short), "a"\*40 (too long)  
Expected: false  
How: JUnit

**PWD-100 – Strong password baseline**  
Feature: isStrongPassword  
Inputs: Abcdefgh1!jk (12+, upper/lower/digit/symbol)  
Expected: true  
How: JUnit

**PWD-101 – Strong password rejects weaknesses**  
Feature: isStrongPassword  
Inputs: short1!A (<12), alllowercase12!, NOLOWERCASE12!, NoDigits!!!!, NoSymbols1234  
Expected: false  
How: JUnit

**PWD-108 – No leading/trailing spaces**  
Feature: isStrongPassword  
Inputs: " Abcd1234!xyz"  
Expected: false  
How: JUnit

**PHN-070 – isPhone valid formats**  
Feature: isPhone  
Inputs: "(480) 555-1212", "480-555-1212", "+1 480 555 1212"  
Expected: true  
How: JUnit

**PHN-074 – isPhone invalid**  
Feature: isPhone  
Inputs: "+1 (abc) def-ghij", "12345" (too short), "+"\*20  
Expected: false  
How: JUnit

**SET-090 – isInSet accepts only allowed**  
Feature: isInSet  
Pre: allowed = {STUDENT, STAFF, ADMIN}  
Inputs: STUDENT, TEACHER  
Expected: true, false  
How: JUnit

**NUM-110 – isPositiveInt**  
Feature: isPositiveInt  
Inputs: "1" → true, "0" → false, "-3" → false, "001" (policy?)  
Expected: per rule  
How: JUnit

**TXT-130 – isSafeFreeText length and control chars**  
Feature: isSafeFreeText  
Inputs: "Hello" → true, "\x00" → false, "a"\*1001 → false  
Expected: as above  
How: JUnit

**4.2 Login (Controller + Semi-Automated UI)**

**LOG-200 – Valid credentials login success**  
Feature: Login  
Pre: user exists with hash; account active  
Inputs: username/email + correct password  
Expected: controller returns success; next view shown  
How: Controller integration + UI check

**LOG-201 – Invalid credentials generic error**  
Feature: Login security messaging  
Pre: user exists  
Inputs: wrong password  
Expected: “Invalid credentials” (no user existence leak); no navigation  
How: Controller + UI message assertion

**LOG-202 – Rate limiting after N failures**  
Feature: Throttling  
Pre: same IP/user tries N failed logins  
Inputs: repeated wrong password  
Expected: backoff/lockout message; no processing  
How: Controller integration (simulate attempts)

**4.3 New Account (Controller + Integration)**

**NEW-300 – Create user happy path**  
Feature: Registration  
Pre: unique username/email  
Inputs: valid names, username, email, strong password, role  
Expected: user persisted; success message  
How: Controller integration; DB assert

**NEW-301 – Duplicate username**  
Feature: Uniqueness  
Pre: username already in DB  
Inputs: same username  
Expected: validation error; no insert  
How: Controller + DB assert

**NEW-302 – Duplicate email**  
Feature: Uniqueness  
Pre: email already in DB  
Inputs: same email  
Expected: validation error  
How: Controller + DB assert

**NEW-303 – Weak password rejected**  
Feature: Password policy  
Pre: none  
Inputs: weak password  
Expected: inline error; blocked  
How: Controller + UI

**NEW-304 – Invalid role tampering**  
Feature: Role set validation  
Pre: allowed roles fixed  
Inputs: hidden field “role=SUPERADMIN”  
Expected: rejected  
How: Controller test

**4.4 Update Account (Controller + Integration)**

**UPD-400 – Change names/email happy path**  
Feature: Profile update  
Pre: user logged in; new email unique  
Inputs: valid new names/email  
Expected: updated in DB; confirmation  
How: Controller + DB assert

**UPD-401 – Change email to duplicate fails**  
Feature: Uniqueness  
Pre: other user with target email exists  
Inputs: duplicate email  
Expected: error; no update  
How: Controller + DB assert

**UPD-402 – Change password requires current**  
Feature: Password change policy  
Pre: user logged in  
Inputs: wrong current password, valid new password  
Expected: error; no change  
How: Controller

**UPD-403 – New password equals old rejected**  
Feature: Password reuse rule  
Pre: current password P  
Inputs: new password P  
Expected: error  
How: Controller

**4.5 Role Dispatch & Role Homes**

**RDS-500 – Single role auto-dispatch**  
Feature: SingleRoleDispatch  
Pre: user has exactly one role  
Inputs: login  
Expected: routed to that role’s home  
How: Controller

**RDS-501 – Multi-role selection enforced**  
Feature: MultipleRoleDispatch  
Pre: user has 2+ roles  
Inputs: select valid role  
Expected: route to selected role; invalid role rejected  
How: Controller + UI

**ROL1-520 – Role1 required field validation**  
Feature: Role1Home  
Pre: DB seeded with valid Course IDs  
Inputs: invalid course id, long notes  
Expected: errors; no action  
How: Controller (project-specific)

**ROL2-540 – Role2 action validation**  
Feature: Role2Home  
Pre: allowedActions defined  
Inputs: invalid action  
Expected: error  
How: Controller

**4.6 Admin – First Admin & Role Management**

**ADM-600 – First admin creation success**  
Feature: FirstAdmin  
Pre: clean DB (no admins)  
Inputs: valid admin username/email/password  
Expected: admin persisted; flagged as admin  
How: Controller + DB

**ADM-601 – Prevent non-unique admin username/email**  
Feature: Uniqueness  
Pre: admin exists  
Inputs: duplicate  
Expected: error  
How: Controller + DB

**ADM-610 – Assign role success**  
Feature: AddRemoveRoles  
Pre: user without STAFF role  
Inputs: assign STAFF  
Expected: role added; no duplicates  
How: Controller + DB

**ADM-611 – Remove role policy**  
Feature: Role removal guard  
Pre: user has last required role  
Inputs: attempt to remove last required role (per team policy)  
Expected: rejected; message  
How: Controller

**4.7 Security & Hardening**

**SEC-700 – No processing on invalid**  
Feature: Fail-fast  
Pre: none  
Inputs: invalid email, weak password  
Expected: DB untouched; no side effects  
How: Integration assertion (no inserts/updates)

**SEC-701 – Output encoding prevents XSS in messages**  
Feature: XSS hardening  
Pre: user’s display name contains <script>alert(1)</script>  
Inputs: view profile / errors  
Expected: rendered safely as text; no script execution  
How: UI render assertion (semi-automated)

**SEC-702 – Logs exclude sensitive data**  
Feature: Logging discipline  
Pre: turn on test logging  
Inputs: failed login or failed validation  
Expected: logs contain event, not passwords/PII  
How: Inspect captured logs in test

**5) Data Sets / Fixtures**

**• Users:**

◦ alice (STUDENT), bob (STAFF), carol (ADMIN), emails unique, strong passwords

**• Duplicates:**

◦ dupUser, dup@example.com existing for uniqueness tests

**• Role sets:**

◦ Allowed roles {STUDENT, STAFF, ADMIN}; invalid sample SUPERADMIN

**• Domain objects (project-specific):**

◦ Example courses, students as needed by Role1/Role2 tests

**6) Pass/Fail Criteria**

• A test **passes** when expected boolean/result state occurs and **no unintended DB changes** happen.

• A test **fails** if: wrong message text, missing inline error, controller proceeds on invalid input, or DB state differs from expected.

**7) Automation Hooks & Reporting**

**• CI command:** mvn -q -Dtest=\*Test test (or your build tool)

**• Reports:** JUnit XML + coverage report

**• Gate:** Merge blocked if coverage for validation packages < 90% or any test fails.

**8) Maintenance & Extensions**

• When adding/altering fields, **add unit tests** for new validators and **integration tests** for controller usage.

• Keep this document in sync with the Input Validation Plan; update test IDs and traceability matrix accordingly.

**Sign-Off**

Name | ASURITE | Role in Testing | Signature | Date