## **1️⃣ Student Use Case: View Upcoming & Past Attended Events**

**Acceptance Criteria:**

* Students should be able to log in using **Email/Student ID and Password**.
* Once logged in, students should be able to:
  + View upcoming **school events**.
  + View their **past attended events**.
* If they are also an **SSG Officer**, their dashboard should also display **attendance scanning features**.
* Students should be able to log out successfully.

## **2️⃣ SSG Officer Use Case: Scan Student Attendance via Face Recognition**

✅ **Acceptance Criteria:**

* SSG Officers should be able to log in using **Email and Password**.
* Once logged in, they should be able to:
  + View **active events** they are assigned to.
  + Click an event to **start attendance scanning**.
  + Use **Face API.js** to scan students' faces during time in, in the middle of the event, and time out to ensure validity of the attendance.
  + If a match is found, mark the student as **present**.
  + If no match is found, display an error message ("Face not recognized").
  + During an event, a student will scan their face three times and if they are unable to complete it, their attendance is not valid and will not be recorded.
  + If multiple scan attempts fail, manually **enter the Student ID**.
* If the SSG Officer is also a **Student**, their dashboard should also display **student-related features** (e.g., event attendance history).
* Officers should be able to log out successfully.

## **3️⃣ Event Organizer Use Case: Create and Manage School Events**

✅ **Acceptance Criteria:**

* Event Organizers should be able to log in using **Email/Student ID and Password**.
* Once logged in, they should be able to:
  + **Create a new event** (Event Name, Date, Location, Assign SSG Officers).
  + View and manage **registered students** for an event.
  + Edit event details **before the event starts**.
  + Delete an event if needed.
* If the Event Organizer is also an **SSG Officer**, their dashboard should also include **attendance scanning features**.
* Organizers should be able to log out successfully.

## **4️⃣ Admin Use Case: Generate Attendance Reports**

✅ **Acceptance Criteria:**

* Admins should be able to log in using **Email/Student ID and Password**.
* Once logged in, they should be able to:
  + View **all school events**.
  + Generate **attendance reports** for any event.
  + Download reports in **CSV format**.
  + Edit **user roles** (e.g., adding/removing SSG Officers or Event Organizers).
* Admins should be able to log out successfully.

**TEST PLAN**

**Project Name:** Valid8 **Version:** 1.0  
**Test Plan ID:** V8-TEST-001  
**Prepared By:** Iryl Jean C. Cadalin  
**Date:** February 19, 2025

## **1. Introduction**

This test plan ensures that the Valid8 functions correctly and meets the sponsor’s requirements. The system allows SSG Officers to record attendance using face recognition and enables students to track their event participation. The system also includes administrators who generate attendance reports and an event organizer that manages and creates events.

## **2. Test Objectives**

* Verify that face recognition accurately detects and logs attendance.
* Ensure role-based access control works properly for students, SSG officers, and admins.
* Validate event attendance logging (time-in/mid-event checking/time-out).
* Prevent duplicate check-ins or unauthorized logins.
* Ensure attendance reports are generated correctly for admins.

## **3. Scope of Testing**

### **In-Scope Features:**

✅ Face recognition-based attendance logging  
✅ Time-in, mid-event checking, and time-out functionality  
✅ Student attendance tracking  
✅ Role-based access control (Student, SSG Officer, Admin)  
✅ Attendance reporting for administrators  
✅ Integration with backend and database

## **4. Test Approach**

* **Manual Testing** → UI interactions, form validations, attendance logs.
* **Automated Testing** → API endpoints (attendance submission, reporting).
* **Database Testing** → Ensure correct data storage in PostgreSQL.

## **5. Test Environment**

* **Frontend:** Web (HTML, CSS, JavaScript, Face API.js)
* **Backend:** Java (Spring Boot, AWS Rekognition/OpenCV for face recognition)
* **Database:** PostgreSQL
* **Hosting:** AWS

6. Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Expected Result | Status |
| TC001 | Users log in to the system | * 1. Students open the website.   2. Log in   3. Click “log in” | User log in successfully |  |
| TC002 | Students attended an event using face recognition | 1. Scan students’ faces by SSG OFFICER during time-in, mid-event checking, and time out. | System recognizes face and logs attendance. |  |
| TC003 | Students view history of events attended and upcoming events. | 1. Logs in to the system successfully. | Students view history of events and upcoming events. |  |
| TC004 | Unauthorized user tries to log in | 1. Open system. 2. Scan unregistered face | System denies access |  |
| TC005 | Duplicate attendance prevention | 1. Students logging in two times | System prevents duplicate check in |  |
| TC006 | Time in, mid-event checking, and out logging | 1. System records the complete attendance of the students | System correctly records time in, mid-event checking and out logging |  |
| TC007 | Creation of events and managing it | 1. Event organizer logs in 2. Create, edit events | Organizer successfully created or edited events |  |
| TC008 | Admin generates attendance report | 1. Log in as admin 2. Navigate to reports 3. Generate events attendance report | System generates a report with attendance data |  |

7. TEST SCHEDULE

|  |  |  |
| --- | --- | --- |
| **Testing Phase** | **Duration** | **Assigned Tester** |
| Unit Testing | Week 6 | Developer |
| Integration Testing | Week 7 | Developer and Tester |
| User Acceptance Testing | Week 8 | SSG Officers and Admins |

8. Risks and Mitigations

|  |  |
| --- | --- |
| Risk | Mitigation Strategy |
| Face recognition fails in low light and without internet connection | Allow Manual ID entry and a picture via any camera as proof |
| Students forgets to log time-out | Remind students to log out after event, if failed to do log out, log in or mid-event checking, attendance will be invalid |
| Unauthorized access attempts | Implement role-based authentication |

9. Test Deliverables

* Test Cases
* Bug Reports (If issues are found)
* Final Test Summary Report

**TEST STRATEGY DOCUMENT**

**Introduction**

This document outlines the testing approach for the Valid8 with Face Recognition. The system includes a web application, an AppSheet mobile app, and backend services that integrate face recognition, user authentication, and event tracking. The goal of testing is to ensure the system meets functional, performance, security, and usability requirements.

**Testing Objectives**

* Validate user authentication using face recognition.
* Ensure accurate attendance tracking.
* Verify backend API functionality (Java Spring Boot).
* Test database integrity (MySQL or PostgreSQL).
* Confirm AppSheet integration with the backend.
* Evaluate system performance under high load.
* Identify and fix security vulnerabilities.

**Scope of Testing**

In-Scope

* Functional Testing: Authentication, event creation, attendance tracking, face recognition accuracy.
* Integration Testing: Communication between frontend (Web & AppSheet), backend (Java), and database (MySQL/PostgreSQL).
* Performance Testing: Load testing for high user concurrency.
* Security Testing: Authentication vulnerabilities, database security.
* Usability Testing: Ease of use for SSG officers, students, and administrators.

Out of Scope

* Third-party API testing (e.g., AWS Rekognition, Face API.js).
* Testing on outdated browsers or unsupported devices.

**Testing Approach**

4.1 Testing Levels

* Unit Testing: Developers will test individual modules (e.g., face recognition API, event creation, user authentication).
* Integration Testing: Ensures proper communication between frontend, backend, and database.
* System Testing: Verifies overall functionality meets project requirements.
* User Acceptance Testing (UAT): Conducted with SSG officers and students to validate usability.

4.2 Testing Types

* Black-box Testing: Functional tests without accessing code.
* White-box Testing: Code-level testing (security vulnerabilities, API calls).
* Regression Testing: Ensuring new updates do not break existing functionality.

**Test Environment**

Frontend: HTML, CSS, JavaScript, Face API.js

Backend: Java (Spring Boot), AWS Rekognition/OpenCV

Database: MySQL/PostgreSQL

Tools: Postman (API testing), Selenium (UI testing), JMeter (performance testing)

Devices: Windows & Mac PCs, Android & iOS devices

Browsers: Chrome, Edge, Firefox

**Test Data**

User Roles: Admin, Event Organizer, SSG Officer, Student

Sample Events: College Orientation, Sports Festival, Seminar

Face Recognition Data: Pre-registered student faces

Invalid Test Data: Incorrect credentials, distorted images for face scanning

**Defect Management**

Bug Tracking Tool: GitHub Issues / Trello

Bug Severity Levels:

Critical: System crash, major security flaws.

High: Authentication failures, incorrect attendance tracking.

Medium: UI misalignment, minor API response delays.

Low: Spelling errors, minor layout issues.

**Test Execution & Reporting**

Sprint-Based Testing: Aligns with Agile sprints.

Daily Standups: Identify blockers and progress.

Test Reports: Shared with the project team weekly.

Final Test Summary: Documenting overall test results before deployment.

**Risks and Mitigation Plan**

|  |  |
| --- | --- |
| **Risk** | **Mitigation** |
| Face recognition in low-light conditions may fail | Improve lighting or adjust detection threshold |
| Database connection issues | Implement automatic reconnection and logging |
| High traffic may slow down authentication | Optimize API calls load testing |
| AppSheet API integration failure | Debug API responses and validate connectivity |

**Approval**

Prepared by: Iryl Jean C. Cadalin

Reviewed by:

Approval date:

**TEST CASES FOR VALID8**

**User Authentication**

**Test Case 1.1:** User Login

Test Case ID: TC-001 Description: Verify that users can log in with valid credentials. Preconditions: User must be registered. Test Steps:

1. Navigate to the login page.
2. Enter a valid username and password.
3. Click the 'Login' button.

Expected Result: User is successfully logged in and redirected to the dashboard. Actual Result: To be filled after testing Status: Pass/Fail

**Test Case 1.2:** Invalid Login Attempt

Test Case ID: TC-002 Description: Verify that login fails with incorrect credentials. Preconditions: User must be registered. Test Steps:

1. Navigate to the login page.
2. Enter an incorrect username or password.
3. Click the 'Login' button.

Expected Result: Error message is displayed, and login is denied. Actual Result: To be filled after testing Status: Pass/Fail

**Face Recognition Attendance**

**Test Case 2.1:** Successful Face Recognition

Test Case ID: TC-003 Description: Verify that a registered user can check in using face recognition. Preconditions: User's face must be registered in the system. Test Steps:

1. Open the attendance system.
2. Click 'Scan Face' button.
3. Align face with camera.
4. System verifies the face.

Expected Result: User attendance is marked successfully. Actual Result: To be filled after testing Status: Pass/Fail

**Test Case 2.2:** Unregistered Face Attempt

Test Case ID: TC-004 Description: Verify that an unregistered face is not allowed to check in. Preconditions: User must not be registered in the system. Test Steps:

1. Open the attendance system.
2. Click 'Scan Face' button.
3. Align an unregistered face with the camera.
4. System verifies the face.

Expected Result: Error message is displayed, and attendance is not marked. Actual Result: To be filled after testing Status: Pass/Fail

**Event Management**

**Test Case 3.1:** Create a New Event

Test Case ID: TC-005 Description: Verify that an event organizer can create an event. Preconditions: Organizer must be logged in. Test Steps:

1. Navigate to the event management page.
2. Click 'Create Event'.
3. Enter event details (name, date, location, description).
4. Click 'Save'.

Expected Result: Event is successfully created and visible in the event list. Actual Result: To be filled after testing Status: Pass/Fail

**Attendance Reports**

**Test Case 4.1:** Generate Attendance Report

Test Case ID: TC-006 Description: Verify that an administrator can generate an attendance report. Preconditions: Users must have checked in at least once. Test Steps:

1. Navigate to the reports page.
2. Select an event.
3. Click 'Generate Report'.

Expected Result: Report is generated with correct attendance data. Actual Result: To be filled after testing Status: Pass/Fail

**AppSheet Integration**

**Test Case 5.1:** AppSheet Sync with Backend

Test Case ID: TC-007 Description: Verify that AppSheet syncs with the backend database. Preconditions: Backend API must be active. Test Steps:

1. Open the AppSheet app.
2. Perform an attendance action (e.g., check-in).
3. Refresh the app.

Expected Result: Data should reflect changes from the backend. Actual Result: To be filled after testing Status: Pass/Fail

**Manual vs. Automated Testing Scope Document**

**Introduction**

The purpose of this document is to define the testing scope for manual and automated testing in the Valid8 with Face Recognition. This ensures efficient test execution, proper allocation of resources, and the identification of areas where automation can provide the most value.

**Testing Strategy**

The testing strategy consists of manual and automated approaches to validate the system's core functionalities, performance, and security.

**Manual Testing Scope**

Certain aspects of the system require manual testing due to their complexity, variability, and need for human validation. These include:

* Face Recognition Accuracy Testing:

Verify the system’s ability to correctly recognize registered faces.

Test with different lighting conditions, angles, and backgrounds.

Validate real-world usage with different facial expressions and accessories.

* UI Usability Testing:

Evaluate the user experience on different devices and screen sizes.

Ensure proper navigation, layout, and responsiveness.

* Security Penetration Testing:

Manually attempt to bypass authentication mechanisms.

Test for potential vulnerabilities in data handling and session management.

* End-to-End Workflow Testing:

Conduct full user scenarios from registration to attendance tracking.

Validate event creation, attendance recording, and report generation.

**Automated Testing Scope**

Automation is used for repetitive and data-intensive testing areas where scripts can efficiently execute tasks.

* API Testing:

Automate validation of authentication endpoints (login, registration, role-based access control).

Test attendance submission and retrieval via REST APIs.

Use tools like Postman or JUnit for automation.

* Performance Testing:

Simulate multiple concurrent users accessing the system.

Measure response times and detect potential bottlenecks.

Use tools like JMeter for load testing.

* Database Integrity Checks:

Ensure proper data storage and retrieval from MySQL.

Validate constraints, relationships, and data consistency.

Automate using SQL scripts and testing frameworks.

**Testing Tools and Technologies**

|  |  |
| --- | --- |
| **Testing Type** | **Tool/Technology** |
| Manual Testing | Real-user execution, Logs, Observation |
| Automated API Testing | Postman, JUnit |
| Automated UI Testing | Selenium |
| Performance Testing | JMeter |
| Database Testing | SQL Queries, JUnit |

**Timeline and Responsibilities**

Testing will be conducted in iterative sprints alongside development. Each sprint will include both manual and automated testing tasks.

|  |  |  |
| --- | --- | --- |
| **Task** | **Testing Type** | **Responsible Person** |
| Face Recognition Testing | Manual | Tester |
| UI Usability Testing | Manual | Tester |
| Authentication API Testing | Automated | Tester |
| Attendance API Testing | Automated | Tester |
| Performance Testing | Automated | Tester |
| Security Testing | Manual | Tester |

**Conclusion**

This document provides a clear breakdown of what will be tested manually and what will be automated. By leveraging automation where applicable, the testing process will be more efficient, while manual testing will ensure real-world usability and security. The goal is to ensure a robust, accurate, and user-friendly attendance system with face recognition.

**Edge Cases and Boundary Conditions**

**Edge case Scenario:**

|  |  |
| --- | --- |
| **Scenario** | **Expected Behavior** |
| **User enters an incorrect password 10 times** | **Account locks temporarily** |
| User submits a blurry/ side profile image | Face recognition rejects the image |
| Multiple users scan faces simultaneously | System queues/processes scans without testing |

**Boundary Conditions**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Test Case** | **Expected Result** |
| Min/Max characters for usernames | 1 char / 50 characters | Should reject/accept accordingly |
| Min/Max image resolution | 100x100px /5000x5000px | System processes valid sizes only |