CAMPUS BOOKING SYSTEM

Metrics Definition Document

Phase 1 · v1.0 · CBS-MET-001

Purpose

This document defines five measurable quality metrics for the Campus Booking System. Each metric is described with its purpose, ISO/IEC 25010 quality characteristic, collection method, target threshold, and a pseudocode algorithm demonstrating how the metric is computed. These metrics will be instrumented in the production system and approximated in the prototype using mock data simulation.

MET-01 — Booking Completion Rate (BCR)

Definition

The percentage of initiated booking sessions that result in a confirmed reservation. A low BCR indicates friction in the booking flow, unclear UI, or availability issues causing users to abandon before confirming.

Pseudocode — BCR Calculation

MET-02 — Resource Utilisation Rate (RUR)

Definition

The percentage of available resource time slots that are booked and used across a given period. Measures how effectively the university’s rooms, equipment, and tutors are being put to use. Low RUR signals under-promoted resources; very high RUR signals capacity shortfall.

Pseudocode — RUR Calculation

MET-03 — Mean Time to Complete Booking (MTCB)

Definition

The average elapsed time from when a user opens the booking flow for a resource to when they receive a booking confirmation. Directly measures the efficiency and usability of the booking interface. High MTCB indicates UX friction, slow loading, or confusing navigation.

Pseudocode — MTCB Calculation

MET-04 — No-Show Rate (NSR)

Definition

The percentage of confirmed bookings where the user did not attend or check in within the grace period after the start time. No-shows waste scarce campus resources and indicate either a reminder system failure or a user commitment problem. High NSR may also indicate users are over-booking as a precaution.

Pseudocode — NSR Calculation

MET-05 — User Satisfaction Score (USS)

Definition

A composite score derived from post-session star ratings submitted by students across all booking types. The USS reflects perceived quality and usability of both the booking system and the resources themselves. It is the most direct measure of quality in use from the student perspective.

Pseudocode — USS Calculation

Document ID: CBS-MET-001

Version: 1.0

Requirements Lead: Umaya Hassan

Phase: Phase 1

Metric ID	Name	ISO/IEC 25010	Target	Owner
MET-01	Booking Completion Rate (BCR)	Reliability, Usability	≥ 85%	Bassel Taleb
MET-02	Resource Utilisation Rate (RUR)	Performance Efficiency	≥ 70%	Jason Myers
MET-03	Mean Time to Complete Booking (MTCB)	Usability, Performance	≤ 90 seconds	Syeda Ahmed

Metric ID	Name	ISO/IEC 25010	Target	Owner
MET-04	No-Show Rate (NSR)	Reliability, Accountability	≤ 10%	Umaya Hassan
MET-05	User Satisfaction Score (USS)	Usability, Quality in Use	≥ 4.0 / 5.0	Syeda Ahmed

ISO/IEC 25010	Reliability (Maturity), Usability (Operability)
Collection Method	Event tracking: log BookingStarted and BookingConfirmed events per session ID
Target Threshold	≥ 85% of initiated booking sessions reach confirmation
Measurement Frequency	Daily rolling 7-day average; spike alert if BCR drops below 75%

```
FUNCTION calculateBookingCompletionRate(timeWindow): // Fetch all booking sessions in the given time window sessions = database.query( SELECT session_id, status, resource_type, started_at FROM booking_sessions WHERE started_at BETWEEN timeWindow.start AND timeWindow.end ) totalInitiated = COUNT(sessions WHERE status IN ['started', 'confirmed', 'abandoned']) totalConfirmed = COUNT(sessions WHERE status = 'confirmed') IF totalInitiated = 0 THEN RETURN { bcr: null, reason: 'No sessions in period' } END IF bcr = (totalConfirmed / totalInitiated) * 100 // Segment by resource type for drill-down analysis FOR EACH resourceType IN ['room', 'equipment', 'tutor'] DO typeInitiated = COUNT(sessions WHERE resource_type = resourceType) typeConfirmed = COUNT(sessions WHERE resource_type = resourceType AND status = 'confirmed') typeBCR[resourceType] = (typeConfirmed / typeInitiated) * 100 END FOR IF bcr < 75 THEN triggerAlert('BCR_LOW', bcr, timeWindow) END IF RETURN { bcr: ROUND(bcr, 2), byType: typeBCR, period: timeWindow } END FUNCTION
```

ISO/IEC 25010	Performance Efficiency (Resource Utilisation)
Collection Method	Compare total available slots vs. confirmed bookings per resource per day
Target Threshold	≥ 70% average utilisation; flag any resource below 30% for review
Measurement Frequency	Weekly report; daily dashboard widget for admin overview

```
FUNCTION calculateResourceUtilisationRate(resourceType, dateRange): resources = database.query( SELECT resource_id, name, total_slots_available FROM resources WHERE type = resourceType AND is_active = true ) results = [] FOR EACH resource IN resources DO availableSlots = resource.total_slots_available * dayCount(dateRange) confirmedBookings = database.query( SELECT COUNT(*) FROM bookings WHERE resource_id = resource.resource_id AND status = 'confirmed' AND booking_date BETWEEN dateRange.start AND dateRange.end ) rur = (confirmedBookings / availableSlots) * 100 results.APPEND({ resource: resource.name, utilisation: ROUND(rur, 2), flag: rur < 30 ? 'UNDER_UTILISED' : rur > 95 ? 'NEAR_CAPACITY' : 'OK' }) END FOR overallRUR = AVERAGE(results.map(r => r.utilisation)) RETURN { overall: ROUND(overallRUR, 2), breakdown: results, period: dateRange } END FUNCTION
```

ISO/IEC 25010	Usability (Operability), Performance Efficiency (Time Behaviour)
Collection Method	Timestamp BookingFlowOpened and BookingConfirmed events; compute elapsed seconds
Target Threshold	≤ 90 seconds mean; p95 ≤ 180 seconds
Measurement Frequency	Continuous; weekly median and p95 report; alert if p95 exceeds 3 minutes

```
FUNCTION calculateMeanTimeToCompleteBooking(resourceType, timeWindow): completedSessions = database.query( SELECT session_id, flow_opened_at, confirmed_at, resource_type FROM booking_sessions WHERE status = 'confirmed' AND resource_type = resourceType AND flow_opened_at BETWEEN timeWindow.start AND timeWindow.end ) IF COUNT(completedSessions) = 0 THEN RETURN { mtcb: null, reason: 'No completed bookings in period' } END IF durations = [] FOR EACH session IN completedSessions DO elapsed = session.confirmed_at - session.flow_opened_at // in seconds // Exclude outliers > 20 minutes (likely user left app and returned) IF elapsed <= 1200 THEN durations.APPEND(elapsed) END IF END FOR durations.SORT(ascending) mean = SUM(durations) / COUNT(durations) median = durations[COUNT(durations) / 2] p95 = durations[FLOOR(COUNT(durations) * 0.95)] IF p95 > 180 THEN triggerAlert('MTCB_HIGH', p95, resourceType, timeWindow) END IF RETURN { mean: ROUND(mean, 1), median: ROUND(median, 1), p95: p95, sampleSize: COUNT(durations) } END FUNCTION
```

ISO/IEC 25010	Reliability (Fault Tolerance), Accountability (Audit)
Collection Method	Compare confirmed bookings vs. check-in events (QR scan / admin mark); 15-min grace period
Target Threshold	≤ 10% across all resource types; tutor sessions ≤ 5%

ISO/IEC 25010	Reliability (Fault Tolerance), Accountability (Audit)
Measurement Frequency	Daily; weekly breakdown by resource type and by user cohort

```
FUNCTION calculateNoShowRate(resourceType, dateRange): CONST GRACE_PERIOD_MINUTES = 15 confirmedBookings = database.query( SELECT booking_id, user_id, resource_id, scheduled_start FROM bookings WHERE resource_type = resourceType AND status = 'confirmed' AND scheduled_start BETWEEN dateRange.start AND dateRange.end ) totalConfirmed = COUNT(confirmedBookings) noShowCount = 0 noShows = [] FOR EACH booking IN confirmedBookings DO graceCutoff = booking.scheduled_start + GRACE_PERIOD_MINUTES checkinEvent = database.query( SELECT * FROM check_ins WHERE booking_id = booking.booking_id AND checked_in_at <= graceCutoff LIMIT 1 ) IF checkinEvent IS NULL THEN noShowCount += 1 noShows.APPEND({ booking_id: booking.booking_id, user_id: booking.user_id }) END IF END FOR nsr = (noShowCount / totalConfirmed) * 100 IF nsr > 10 THEN triggerAlert('NSR_HIGH', nsr, resourceType) // Flag repeat no-show users for admin review repeatNoShows = GROUP(noShows BY user_id WHERE COUNT > 3) flagForAdminReview(repeatNoShows) END IF RETURN { nsr: ROUND(nsr, 2), noShowCount, totalConfirmed, period: dateRange } END FUNCTION
```

ISO/IEC 25010	Usability (User Interface Aesthetics), Quality in Use (Satisfaction)
Collection Method	Aggregate star ratings (1–5) submitted after completed sessions, weighted by recency
Target Threshold	≥ 4.0 / 5.0 overall; individual resources below 3.5 flagged for review
Measurement Frequency	Rolling 30-day average; monthly trend report for Product Owner

```
FUNCTION calculateUserSatisfactionScore(timeWindow, resourceType = 'ALL'): CONST RECENCY_DECAY = 0.95 // Each week older = 5% less weight ratings = database.query( SELECT rating_value, submitted_at, resource_type, resource_id FROM ratings WHERE submitted_at BETWEEN timeWindow.start AND timeWindow.end AND (resourceType = 'ALL' OR resource_type = resourceType) AND rating_value BETWEEN 1 AND 5 ) IF COUNT(ratings) < 10 THEN RETURN { uss: null, reason: 'Insufficient data (< 10 ratings)' } END IF today = CURRENT_DATE() weightedSum = 0 totalWeight = 0 FOR EACH rating IN ratings DO weeksOld = DAYS_BETWEEN(today, rating.submitted_at) / 7 weight = POWER(RECENCY_DECAY, weeksOld) // More recent = higher weight weightedSum += rating.rating_value * weight totalWeight += weight END FOR uss = weightedSum / totalWeight // Per-resource breakdown to identify underperformers resourceScores = GROUP(ratings BY resource_id) .MAP(group => { id: group.resource_id, score: AVERAGE(group.rating_value) }) .FILTER(r => r.score < 3.5) IF COUNT(resourceScores) > 0 THEN flagForAdminReview(resourceScores, 'LOW_SATISFACTION') END IF RETURN { uss: ROUND(uss, 2), ratingCount: COUNT(ratings), flaggedResources: resourceScores, period: timeWindow } END FUNCTION
```