Pseudocode

# User Class

CLASS User

// Properties

PRIVATE userId: String

PRIVATE email: String

PRIVATE passwordHash: String

PRIVATE username: String

PRIVATE firstName: String

PRIVATE lastName: String

PRIVATE profilePictureUrl: String

PRIVATE studyLevel: Enum (HIGH\_SCHOOL, UNIVERSITY, PROFESSIONAL)

PRIVATE yearLevel: Integer

PRIVATE dateOfBirth: Date

PRIVATE isActive: Boolean = TRUE

PRIVATE isPremium: Boolean = FALSE

PRIVATE emailVerified: Boolean = FALSE

PRIVATE createdAt: Timestamp

PRIVATE updatedAt: Timestamp

PRIVATE lastLogin: Timestamp

PRIVATE currentStreak: Integer = 0

PRIVATE timezone: String = "Australia/Melbourne"

// Constructor

CONSTRUCTOR(userData)

SET this.userId = generateUUID()

SET this.email = userData.email

SET this.username = userData.username

SET this.firstName = userData.firstName

SET this.lastName = userData.lastName

SET this.studyLevel = userData.studyLevel

SET this.dateOfBirth = userData.dateOfBirth

SET this.createdAt = getCurrentTimestamp()

SET this.updatedAt = getCurrentTimestamp()

CALL validateUserData()

END CONSTRUCTOR

// Validation

PRIVATE FUNCTION validateUserData()

IF email NOT matches email regex THEN

THROW ValidationError("Invalid email format")

END IF

IF username length < 3 OR username length > 50 THEN

THROW ValidationError("Username must be 3-50 characters")

END IF

IF calculateAge() < 13 THEN

THROW ValidationError("User must be at least 13 years old")

END IF

END FUNCTION

// Authentication Methods

PUBLIC FUNCTION setPassword(plainPassword)

IF plainPassword length < 8 THEN

THROW ValidationError("Password must be at least 8 characters")

END IF

IF NOT containsUppercase(plainPassword) OR NOT containsLowercase(plainPassword) OR NOT containsNumber(plainPassword) THEN

THROW ValidationError("Password must contain uppercase, lowercase, and number")

END IF

SET this.passwordHash = bcrypt.hash(plainPassword, 10)

SET this.updatedAt = getCurrentTimestamp()

END FUNCTION

PUBLIC FUNCTION verifyPassword(plainPassword)

RETURN bcrypt.compare(plainPassword, this.passwordHash)

END FUNCTION

PUBLIC FUNCTION login(password)

IF NOT this.isActive THEN

THROW AuthError("Account is deactivated")

END IF

IF NOT verifyPassword(password) THEN

THROW AuthError("Invalid credentials")

END IF

SET this.lastLogin = getCurrentTimestamp()

CALL updateStreak()

RETURN generateAuthToken()

END FUNCTION

# Study Streak Tracking Feature

CLASS StudyStreakManager

PRIVATE database: DatabaseConnection

PRIVATE notificationService: NotificationService

PRIVATE achievementService: AchievementService

// Main function called when a study session is completed

PUBLIC FUNCTION updateStreakOnSessionComplete(userId, sessionId)

// Get the completed session details

SET session = database.getSession(sessionId)

// Verify session was actually completed

IF session.status != 'completed' THEN

RETURN FALSE

END IF

// Get user's current streak data

SET streakData = database.getStudyStreak(userId)

IF streakData IS NULL THEN

// First time studying - create new streak record

CALL createNewStreak(userId)

ELSE

// Update existing streak

CALL updateExistingStreak(userId, streakData)

END IF

// Check for streak-related achievements

CALL checkStreakAchievements(userId)

RETURN TRUE

END FUNCTION

// Create a new streak record for first-time users

PRIVATE FUNCTION createNewStreak(userId)

SET newStreak = {

streakId: generateUUID(),

userId: userId,

currentStreak: 1,

longestStreak: 1,

lastStudyDate: getCurrentDate(),

streakStartDate: getCurrentDate(),

updatedAt: getCurrentTimestamp()

}

database.insertStudyStreak(newStreak)

// Send first streak notification

notificationService.send(userId, {

type: 'achievement',

title: '🔥 Streak Started!',

message: 'You\'ve started your study streak! Keep it up tomorrow.'

})

END FUNCTION

// Update existing streak based on study patterns

PRIVATE FUNCTION updateExistingStreak(userId, streakData)

SET today = getCurrentDate()

SET lastStudyDate = streakData.lastStudyDate

SET daysSinceLastStudy = calculateDaysBetween(lastStudyDate, today)

IF daysSinceLastStudy = 0 THEN

// Already studied today - no change needed

RETURN

ELSE IF daysSinceLastStudy = 1 THEN

// Studied yesterday - increment streak

SET streakData.currentStreak = streakData.currentStreak + 1

SET streakData.lastStudyDate = today

// Check if this is a new record

IF streakData.currentStreak > streakData.longestStreak THEN

SET streakData.longestStreak = streakData.currentStreak

CALL celebrateNewRecord(userId, streakData.currentStreak)

END IF

// Milestone notifications

IF streakData.currentStreak IN [7, 14, 30, 60, 100, 365] THEN

CALL sendMilestoneNotification(userId, streakData.currentStreak)

END IF

ELSE IF daysSinceLastStudy > 1 THEN

// Missed days - streak broken

SET oldStreak = streakData.currentStreak

SET streakData.currentStreak = 1

SET streakData.lastStudyDate = today

SET streakData.streakStartDate = today

// Notify about broken streak (if it was significant)

IF oldStreak >= 7 THEN

CALL notifyStreakBroken(userId, oldStreak)

END IF

END IF

SET streakData.updatedAt = getCurrentTimestamp()

database.updateStudyStreak(userId, streakData)

END FUNCTION

// Check if user has earned any streak achievements

PRIVATE FUNCTION checkStreakAchievements(userId)

SET streakData = database.getStudyStreak(userId)

SET userAchievements = database.getUserAchievements(userId)

// Define streak achievements

SET streakAchievements = [

{id: 'first\_week', days: 7, name: 'Week Warrior'},

{id: 'two\_weeks', days: 14, name: 'Fortnight Fighter'},

{id: 'one\_month', days: 30, name: 'Monthly Master'},

{id: 'two\_months', days: 60, name: 'Commitment Champion'},

{id: 'hundred\_days', days: 100, name: 'Century Scholar'},

{id: 'full\_year', days: 365, name: 'Year-long Learner'}

]

FOR EACH achievement IN streakAchievements DO

// Check if streak meets requirement and achievement not already earned

IF streakData.currentStreak >= achievement.days AND

NOT userAchievements.contains(achievement.id) THEN

// Award achievement

achievementService.awardAchievement(userId, achievement.id)

// Special celebration for major milestones

IF achievement.days >= 100 THEN

CALL triggerMajorCelebration(userId, achievement)

END IF

END IF

END FOR

END FUNCTION

// Send milestone notification

PRIVATE FUNCTION sendMilestoneNotification(userId, streakDays)

SET emoji = getMilestoneEmoji(streakDays)

SET message = getMilestoneMessage(streakDays)

notificationService.send(userId, {

type: 'achievement',

title: emoji + ' ' + streakDays + ' Day Streak!',

message: message,

dataJson: {

streakDays: streakDays,

showConfetti: TRUE

}

})

END FUNCTION

// Notify user their streak was broken

PRIVATE FUNCTION notifyStreakBroken(userId, lostStreak)

notificationService.send(userId, {

type: 'system',

title: '😢 Streak Broken',

message: 'You lost your ' + lostStreak + ' day streak. Start a new one today!',

dataJson: {

lostStreak: lostStreak,

motivationalQuote: getRandomMotivationalQuote()

}

})

END FUNCTION

// Get user's current streak for display

PUBLIC FUNCTION getCurrentStreak(userId)

SET streakData = database.getStudyStreak(userId)

IF streakData IS NULL THEN

RETURN 0

END IF

// Check if streak is still active

SET daysSinceLastStudy = calculateDaysBetween(streakData.lastStudyDate, getCurrentDate())

IF daysSinceLastStudy > 1 THEN

// Streak is broken but not yet updated

RETURN 0

ELSE

RETURN streakData.currentStreak

END IF

END FUNCTION

// Get streak status for dashboard display

PUBLIC FUNCTION getStreakStatus(userId)

SET streakData = database.getStudyStreak(userId)

IF streakData IS NULL THEN

RETURN {

currentStreak: 0,

longestStreak: 0,

status: 'no\_streak',

message: 'Start your first streak!',

isAtRisk: FALSE

}

END IF

SET today = getCurrentDate()

SET daysSinceLastStudy = calculateDaysBetween(streakData.lastStudyDate, today)

IF daysSinceLastStudy = 0 THEN

// Studied today

RETURN {

currentStreak: streakData.currentStreak,

longestStreak: streakData.longestStreak,

status: 'active',

message: 'Great job! Streak maintained.',

isAtRisk: FALSE

}

ELSE IF daysSinceLastStudy = 1 THEN

// Haven't studied today yet

RETURN {

currentStreak: streakData.currentStreak,

longestStreak: streakData.longestStreak,

status: 'at\_risk',

message: 'Study today to keep your streak!',

isAtRisk: TRUE

}

ELSE

// Streak broken

RETURN {

currentStreak: 0,

longestStreak: streakData.longestStreak,

status: 'broken',

message: 'Streak broken. Start a new one!',

isAtRisk: FALSE

}

END IF

END FUNCTION

// Helper function to calculate days between dates

PRIVATE FUNCTION calculateDaysBetween(date1, date2)

SET date1Midnight = setTimeToMidnight(date1)

SET date2Midnight = setTimeToMidnight(date2)

SET millisecondsDiff = date2Midnight - date1Midnight

SET daysDiff = millisecondsDiff / (1000 \* 60 \* 60 \* 24)

RETURN Math.floor(daysDiff)

END FUNCTION

// Get appropriate emoji for milestone

PRIVATE FUNCTION getMilestoneEmoji(days)

IF days = 7 THEN RETURN '🌟'

ELSE IF days = 14 THEN RETURN '⭐'

ELSE IF days = 30 THEN RETURN '🏆'

ELSE IF days = 60 THEN RETURN '🎯'

ELSE IF days = 100 THEN RETURN '💎'

ELSE IF days = 365 THEN RETURN '👑'

ELSE RETURN '🔥'

END FUNCTION

END CLASS