ALGORITHM SchoolManagementSystem

- DATA STRUCTURES:
- students: Dictionary {student_id → Student}
- Student: Record {id: str, name: str, class: str, marks: Dictionary{subject→float}}

- 1. ADD_STUDENT(student_id, name, class_name):
- IF student_id ∉ students:

  students[student_id] ← new Student(student_id, name, class_name, empty_marks)

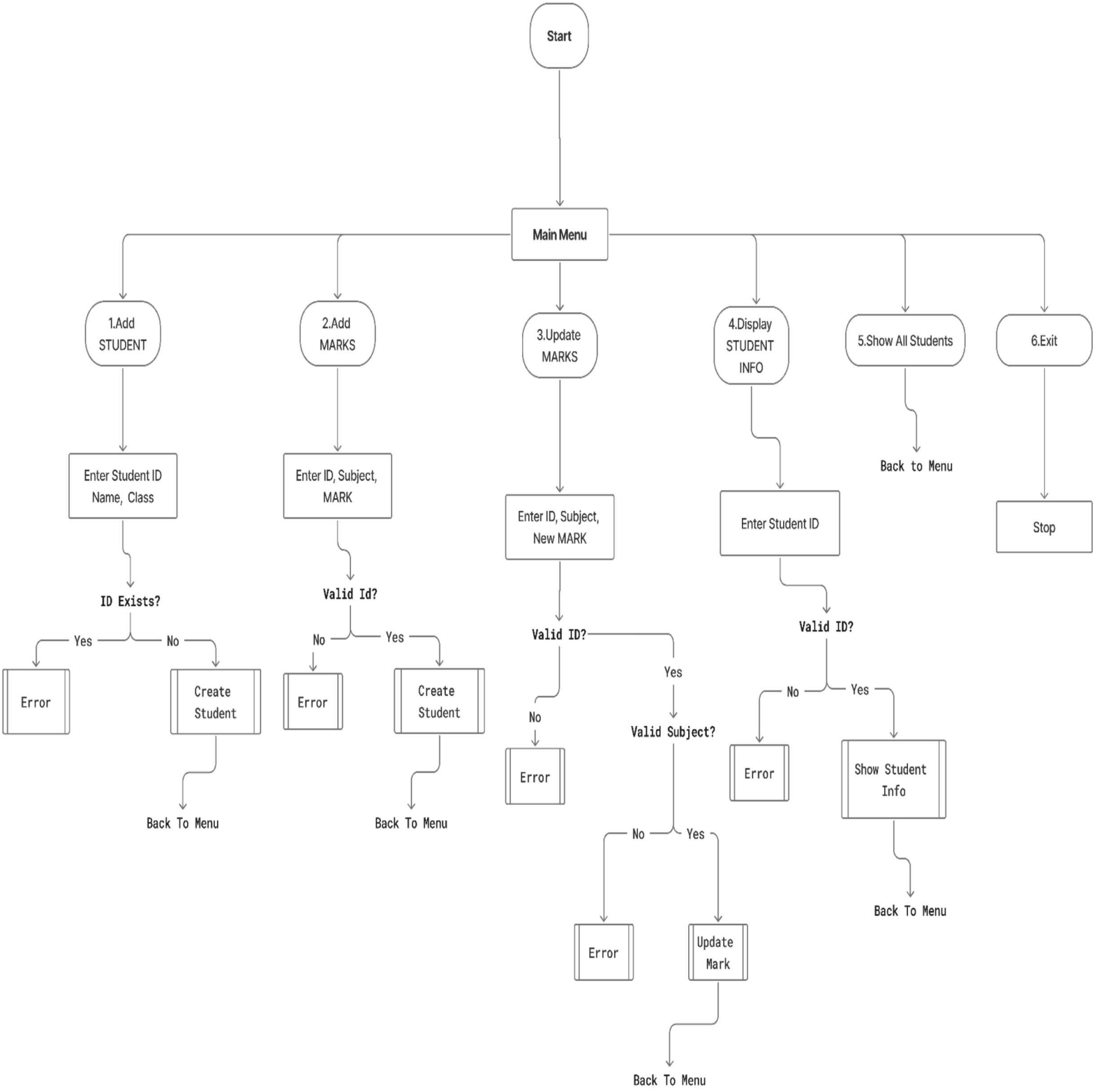
- ELSE:

  PRINT "Student ID already exists"

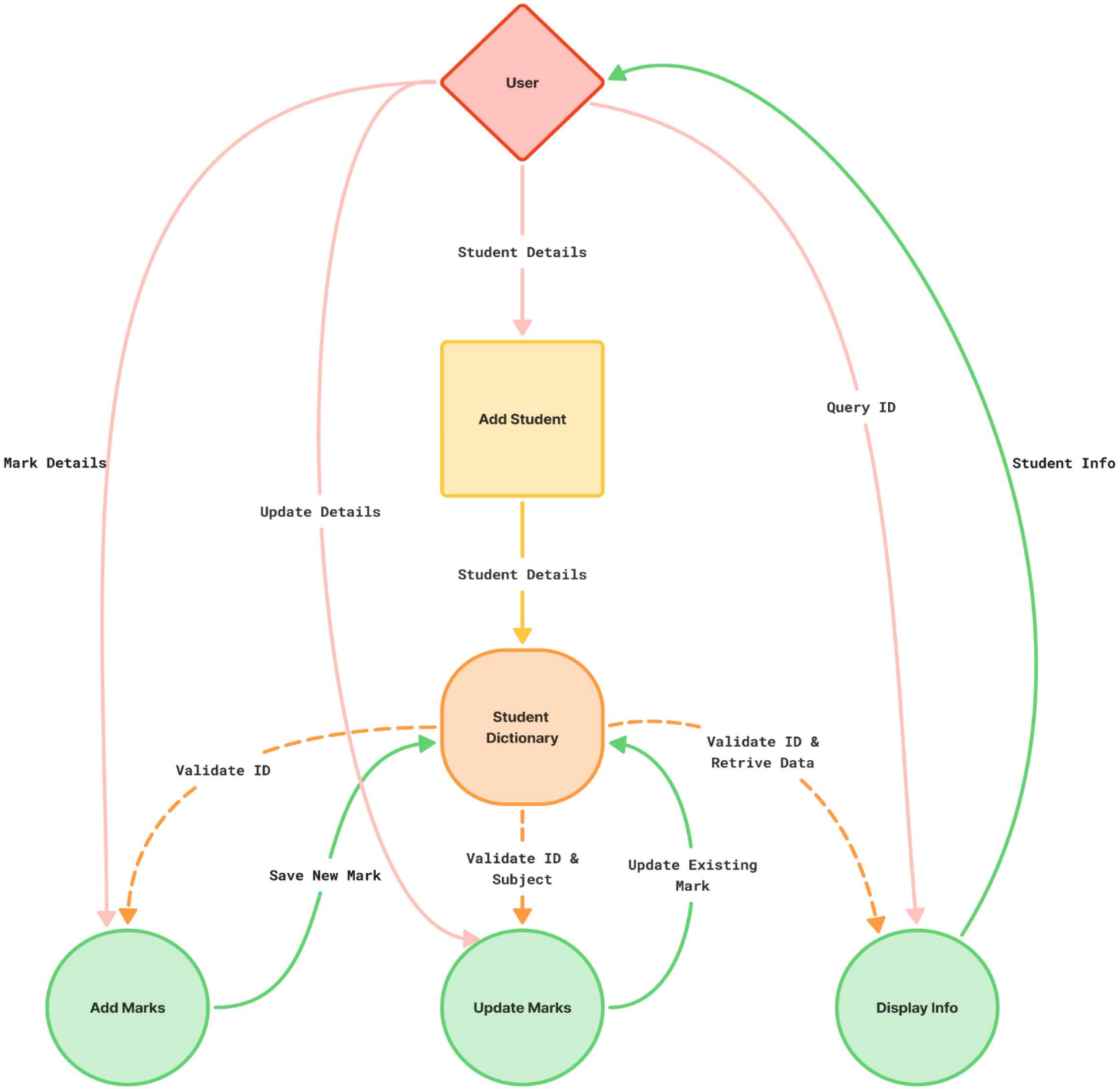
- 2. ADD_MARKS(student_id, subject, mark): // @validate_student_id
- VALIDATE student_id ∈ students
- students[student_id].marks[subject] ← mark
-
- 3. UPDATE_MARKS(student_id, subject, mark): // @validate_student_id + @validate_subject
- VALIDATE student_id ∈ students
- VALIDATE subject ∈ students[student_id].marks
- students[student_id].marks[subject] ← mark
-
- 4. DISPLAY_STUDENT(student_id): // @validate_student_id
- VALIDATE student_id ∈ students
- PRINT students[student_id].display_info()

- 5. DISPLAY_ALL():
- FOR each student IN students.values():
- student.display_info()
-
- DECORATOR VALIDATE_STUDENT_ID(func):
- wrapper(student_id, args):
- IF student_id ∉ self.students:
- PRINT "Student not found"
- RETURN null
- RETURN func(student_id, args)
-
- MAIN CLI LOOP:
- WHILE true:
- DISPLAY menu {1=add_student, 2=add_marks, 3=update_marks, 4=display_student, 5=display_all, 6=exit}
- READ choice
- SWITCH choice:
- CASE 1: CALL add_student(input())
- CASE 2: CALL add_marks(input())
- CASE 3: CALL update_marks(input())
- CASE 4: CALL display_student(input())
- CASE 5: CALL display_all()
- CASE 6: BREAK

FLOWCHART


Start
Main Menu
1.Add STUDENT
2.Add MARKS
3.Update MARKS
4.Display STUDENT INFO
5.Show All Students
6.Exit
Enter Student ID Name, Class
Enter ID, Subject, MARK
Enter ID, Subject, New MARK
Enter Student ID
Back to Menu
Stop
ID Exists?
Valid Id?
Valid ID?
Valid ID?
Yes
No
Error
Create Student
Back To Menu
Valid Subject?
Update Mark
Show Student Info

DFD


User
Student Details
Add Student
Mark Details
Update Details
Query ID
Student Info
Student Details
Student Dictionary
Validate ID
Validate ID &
Retrive Data
Save New Mark
Validate ID &
Subject
Update Existing
Mark
Add Marks
Update Marks
Display Info

Traceability Matrix

Req ID	Requirement	Test Scenario	Test Type	Expected Result	Final Result
R1	Add a new student	Add student with a unique student ID	Functional	Student is added successfully	Pass
R2	Prevent duplicate student IDs	Add student using an existing student ID	Functional	System throws DuplicateStudentError	Pass
R3	Add marks to a student	Add marks for a valid student and subject	Functional	Marks are stored correctly	Pass
R4	Validate student existence while adding marks	Add marks using an invalid student ID	Functional	StudentNotFoundError is raised	Pass
R5	Update student marks	Update marks for an existing subject of a valid student	Functional	Marks are updated successfully	Pass
R6	Validate subject existence during update	Update marks for a subject that does not exist	Functional	SubjectNotFoundError is raised	Pass
R7	Validate student existence during update	Update marks using an invalid student ID	Functional	StudentNotFoundError is raised	Pass
R8	Display student information	Display details of a valid student	Functional	Student information is displayed	Pass
R9	Handle invalid student display request	Display student info using an invalid student ID	Functional	StudentNotFoundError is raised	Pass