SRS

Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

The purpose of this document is to specify the requirements for the **Study Buddy** scheduling application. The application will be implemented as a command-line interface (CLI) in Java. It is designed to help students in a class connect with classmates, share their availability, and schedule study sessions.

1.2 Scope

The Study Buddy application will allow students to:

- Create and manage a profile with courses they are enrolled in.
- Add or remove availability slots for study sessions.
- Search for classmates enrolled in the same courses.
- Receive suggested matches based on course enrollment and availability.
- Propose and confirm study sessions with classmates.

The system will be a standalone CLI program that stores data locally.

1.3 Definitions, Acronyms, Abbreviations

- **CLI**: Command-Line Interface
- **Availability**: Time slots a student is available to study
- **Match**: A suggested classmate for a study session
- **Session**: A scheduled meeting between students to study

1.4 References

- Project assignment description (classroom reference)

1.5 Overview

The remainder of this document describes the product's functions, user interactions, constraints, requirements, and simple use cases.

---

2. Overall Description

2.1 Product Perspective

The Study Buddy app is a new, self-contained application. It does not integrate with external systems. It will be built in Java and run in a terminal.

2.2 Product Functions

At a high level, the system will:

1. Let students create profiles with basic information and courses.
2. Allow students to add and remove availability.
3. Suggest study partners based on shared courses and matching availability.
4. Enable students to schedule and confirm study sessions.

2.3 User Classes and Characteristics

- **Student**: Any member of the class. Expected to be comfortable using simple terminal commands.

2.4 Operating Environment

- Java 17 or later
- Windows, macOS, or Linux terminal

2.5 Design and Implementation Constraints

- Must be a CLI program
- Written in Java
- Local data storage (no network or external services required)

2.6 Assumptions and Dependencies

- Students will enter accurate course and availability information.
- All users are from the same class.

---

3. Specific Requirements

3.1 Functional Requirements

- **FR-1**: The system shall allow students to create a profile with their name and enrolled courses.
- **FR-2**: The system shall allow students to add and remove availability slots.
- **FR-3**: The system shall allow students to search for classmates by course.
- **FR-4**: The system shall suggest matches based on overlapping courses and availability.
- **FR-5**: The system shall allow students to propose a study session to one or more classmates.
- **FR-6**: The system shall allow invited classmates to confirm the study session.
- **FR-7**: The system shall display scheduled study sessions to students.

3.2 Non-Functional Requirements

- **NFR-1**: The application shall run in a terminal environment without requiring internet access.
- **NFR-2**: The system shall store data locally in a simple format (e.g., text or JSON files).
- **NFR-3**: The system shall respond to typical user commands within 1 second.
- **NFR-4**: The system shall use clear and simple text-based menus and prompts.

3.3 User Interfaces

The system will provide text-based commands and menus.

4. Use Cases

UC-1: Create Profile

- Actor: Student
- Steps: Student enters name and courses → Profile is saved

UC-2: Add Availability

- Actor: Student
- Steps: Student enters a time slot → Availability is saved

UC-3: Suggest Matches

- Actor: Student
- Steps: Student requests matches → System lists classmates in same course with overlapping availability

UC-4: Schedule Session

- Actor: Student

- Steps: Student selects a classmate and proposes a time → Classmate confirms → Session is saved

---

5. Data Requirements

- Student profiles: name, list of courses
- Availability: student ID, time slots
- Study sessions: participants, course, date/time, status (proposed/confirmed)

---

6. Constraints

- Must be implemented in Java
- CLI-based only
- Local file-based storage

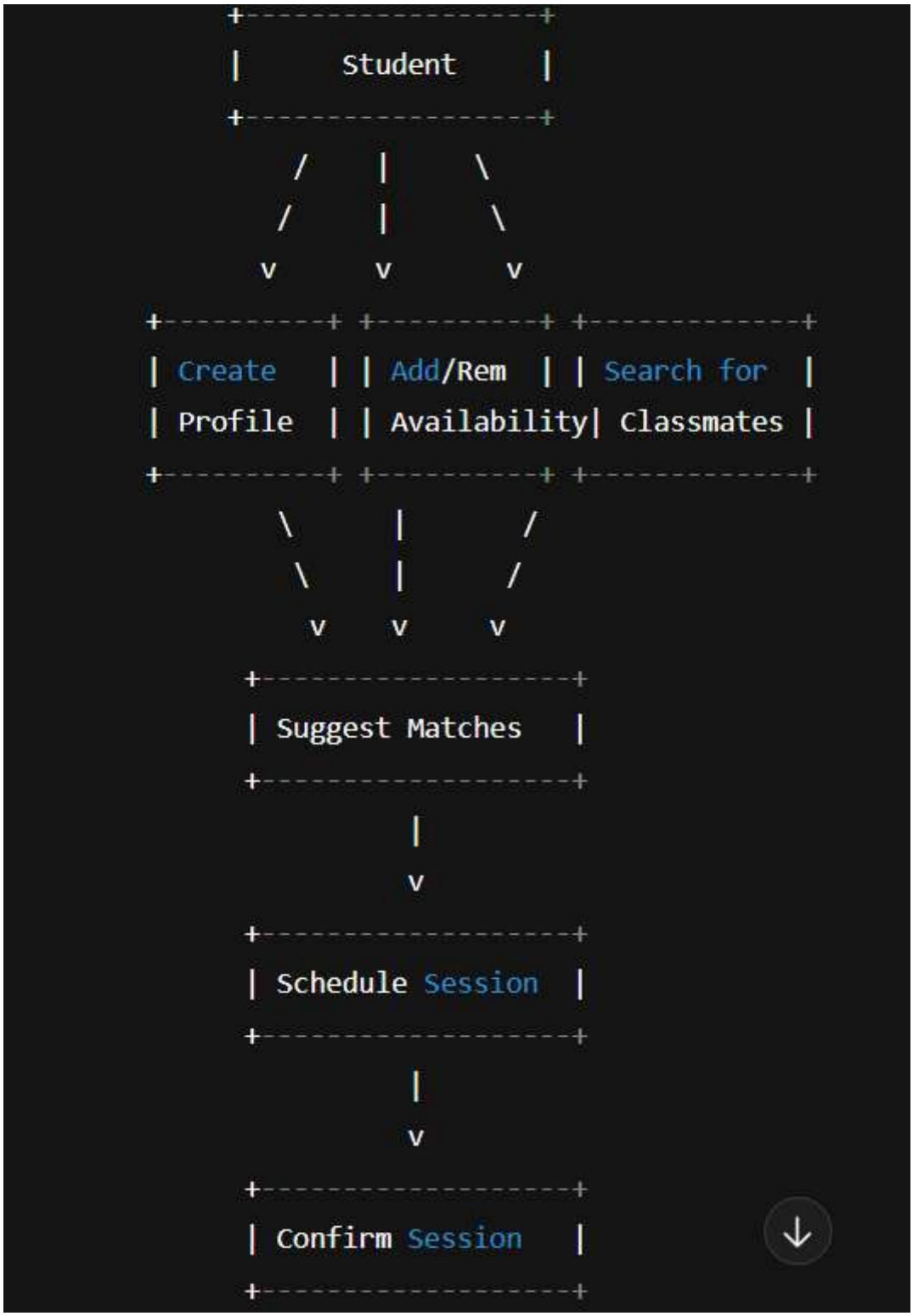
---

7. Future Enhancements (Out of Scope)

- Support for multiple classes or institutions
- Integration with calendar apps
- Notifications via email or mobile

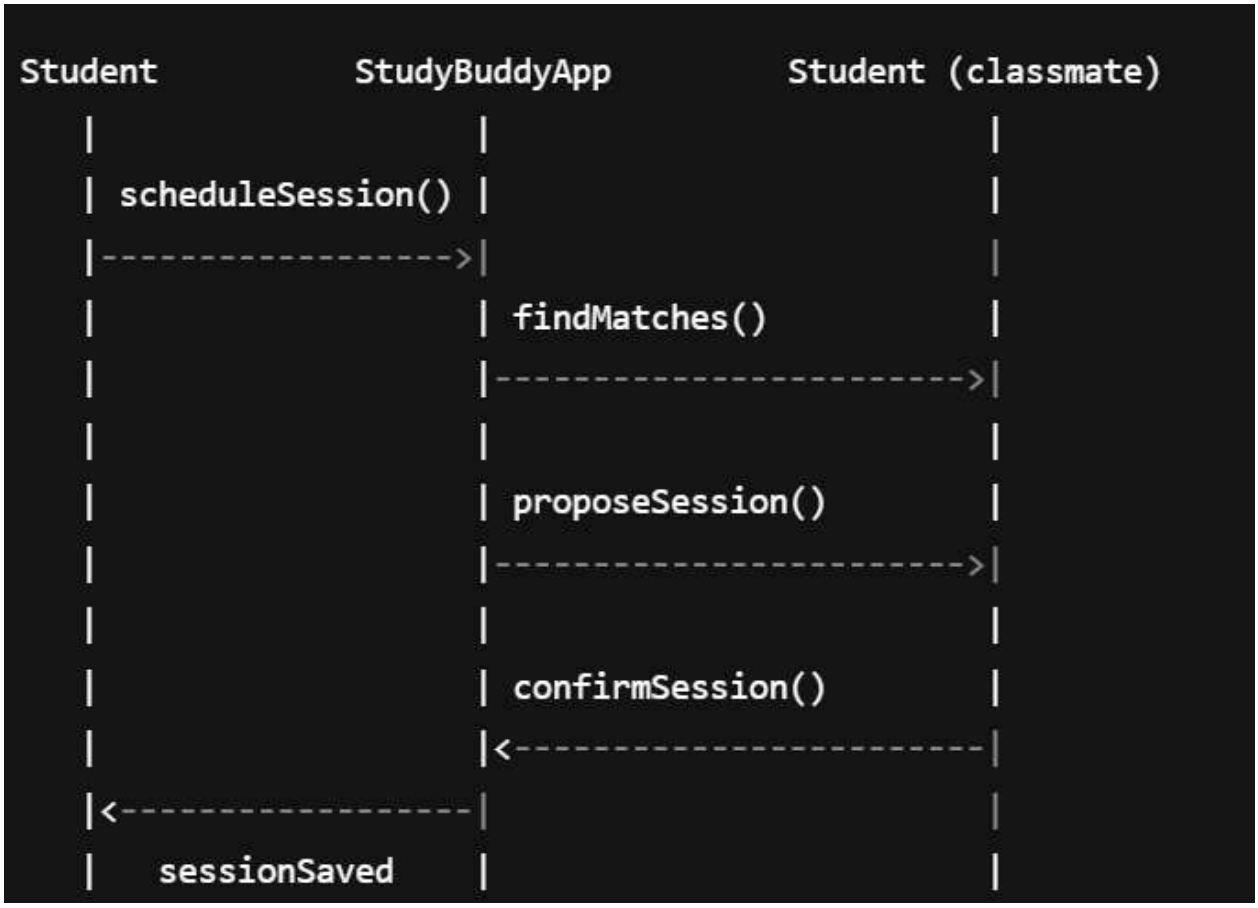
Design Docs (pseudocode, UML diagrams)

UML Use Case Diagram:



UML Class Diagram:

UML Sequence Diagram:



High-Level Pseudocode:

MAIN MENU:
  Display options:
    1. Create Profile
    2. View Profile
    3. Add Availability
    4. Remove Availability
    5. Search for Classmates
    6. Suggest Matches
    7. Schedule Session
    8. Confirm Session
    9. Exit

FUNCTION createProfile():
  prompt "Enter name:"
  prompt "Enter courses (comma-separated):"
  create Student with name and courses
  save to local file

```
FUNCTION addAvailability(student):
  prompt "Enter day:"
  prompt "Enter start time:"
  prompt "Enter end time:"
  create TimeSlot and add to student.availability
  save

FUNCTION removeAvailability(student):
  list student.availability
  prompt "Select slot to remove"
  remove slot
  save

FUNCTION searchByCourse(course):
  results = all students where course in student.courses
  display results

FUNCTION suggestMatches(student):
  for each other student in system:
    if shared course AND overlapping availability:
       add to match list
  display match list

FUNCTION scheduleSession(student, match, timeslot, course):
  session = new StudySession([student, match], course, timeslot, "proposed")
  add to system.sessions
  save

FUNCTION confirmSession(session, match):
  session.status = "confirmed"
  save
  notify participants

LOOP until exit
```

Test Plan & Results

```
class StudyBuddyTest {

   @Test
   void testCreateProfileAndPrintProfile(){
       Scanner fakeScanner = new
Scanner("Jonah\nColestock\n1\nCPSC\n3720\n1\nCPSC\n4300\n1\nCPSC\n2310\n2\n1\n
October 3\n0830\n1230\n2\n");
       StudyBuddy.createProfile(fakeScanner);
       ByteArrayOutputStream outContent = new ByteArrayOutputStream();
       PrintStream originalOut = System.out;
       System.setOut(new PrintStream(outContent));
       Scanner fakeScanner2 = new Scanner("Jonah\nColestock\n");
       StudyBuddy.printProfile(fakeScanner2);
       System.setOut(originalOut);
       assertFalse(StudyBuddy.students.isEmpty());
       String output = outContent.toString();
       System.out.println("Reached here\n");
       assertTrue(output.contains("Jonah Colestock"));
       assertTrue(output.contains("CPSC 3720"));
       assertTrue(output.contains("October 3, 830 to 1230"));
   }

   @Test
   void testSearchByCourse(){
       Scanner fakeScanner = new
Scanner("Jonah\nColestock\n1\nCPSC\n3720\n1\nCPSC\n4300\n1\nCPSC\n2310\n2\n1\n
October 3\n0830\n1230\n2\n");
       StudyBuddy.createProfile(fakeScanner);
       Scanner fakeScanner2 = new
Scanner("Jack\nHilliard\n1\nCPSC\n4300\n1\nCPSC\n2310\n2\n1\nOctober
3\n0830\n1230\n2\n");
       StudyBuddy.createProfile(fakeScanner2);
       ArrayList<Student> us = StudyBuddy.searchByCourse("CPSC 2310");
       assertFalse(us.isEmpty());
       for(Student s : us){
           assertTrue(s.getName().equals("Jonah Colestock") ||
s.getName().equals("Jack Hilliard"));
       }
   }


}
```

```
class StudentTest {

   @Test
   void addCourseTest() {
```

```
        Scanner fakeScanner = new Scanner("CPSC\n4300\n");
        Student me = new Student("Jonah");
        me.AddCourse(fakeScanner);
        assertFalse(me.getCourses().isEmpty());
        String myClass = me.getCourses().get(0);
        assertEquals("CPSC 4300", myClass);
    }

    @Test
    void addAvailabilityTest(){
        Scanner fakeScanner = new Scanner("October 3\n1230\n1630\n");
        Student me = new Student("Jonah");
        me.AddAvailability(fakeScanner);
        assertFalse(me.getAvailability().isEmpty());
        TimeSlot time = new TimeSlot("October 3", 1230, 1630);
        assertTrue(me.getAvailability().get(0).equals(time));
    }

    @Test
    void removeAvailabilityTest(){
        Scanner fakeScanner = new Scanner("October 3\n1230\n1630\n");
        Student me = new Student("Jonah");
        me.AddAvailability(fakeScanner);
        Scanner fakeScanner2 = new Scanner("October 3\n1230\n1630\n");
        me.RemoveAvailability(fakeScanner2);
        assertTrue(me.getAvailability().isEmpty());
    }
}
```

After several failures and hours of debugging, the test cases all passed.

Additional testing occurred in the command-line interface, particularly for making the program initially compile and run.