CS 3704: Intermediate Software Design

Group 2

TA Queue

Design II: Mock UI & Algorithm Design

Sam Lightfoot, Pasha Ranakusuma, Omar Elgeoushy, Thejus Unnivelan

Check out our GitHub Project here:

https://github.com/Intermediate-Software-Design/ISD-Group-Project

April 15, 2022

Contents

Chapter 1

Mock UI

Please note that the primary function of TA Queue is to serve as a mobile app for general-purpose uses. More specific uses, such as sharing a screen and handling files may be better suited over a desktop. For these reasons, the majority of the UI examples given are as they would appear on a Android mobile device, save for the screen sharing, as that appears as it may on a desktop/laptop device.

1.1 Course UI

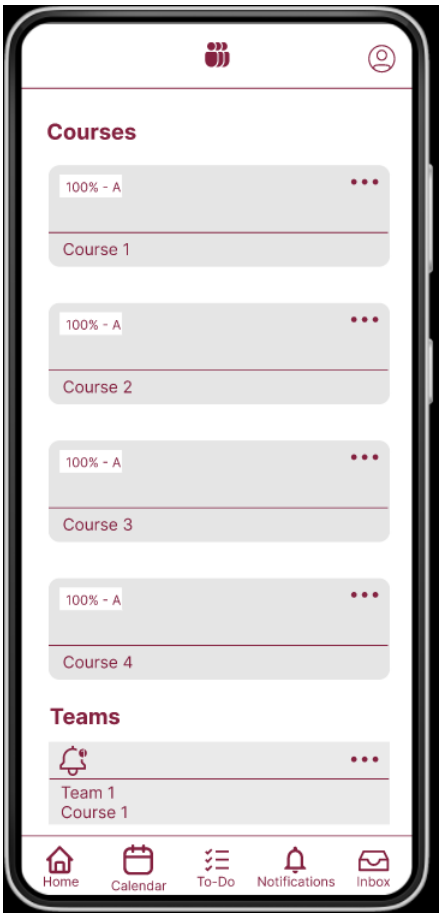

Figure 1.1: Homepage

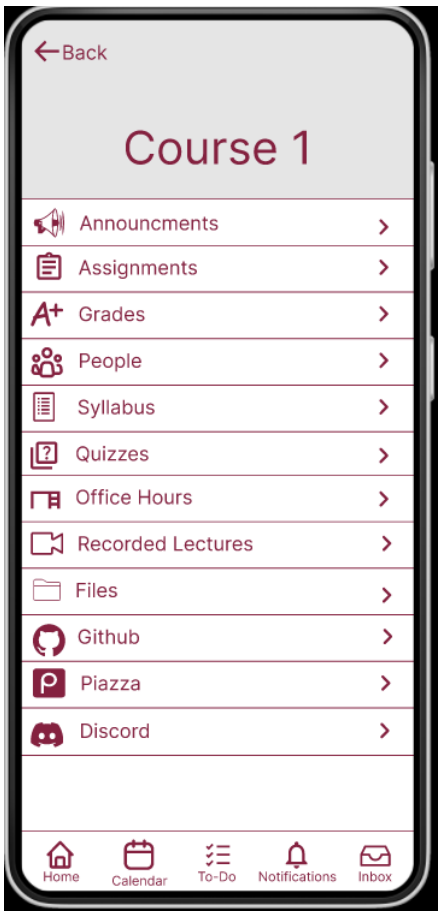

Figure 1.2: Course Overview

1.2 Priority Queue UI

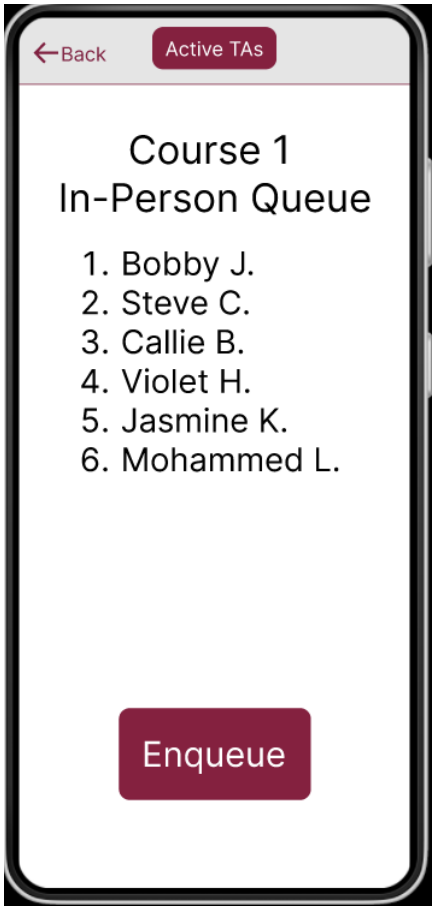

Figure 1.3: Student View (Dequeued) - Print Queue

Figure 1.4: Student View (Dequeued) - Active TAs

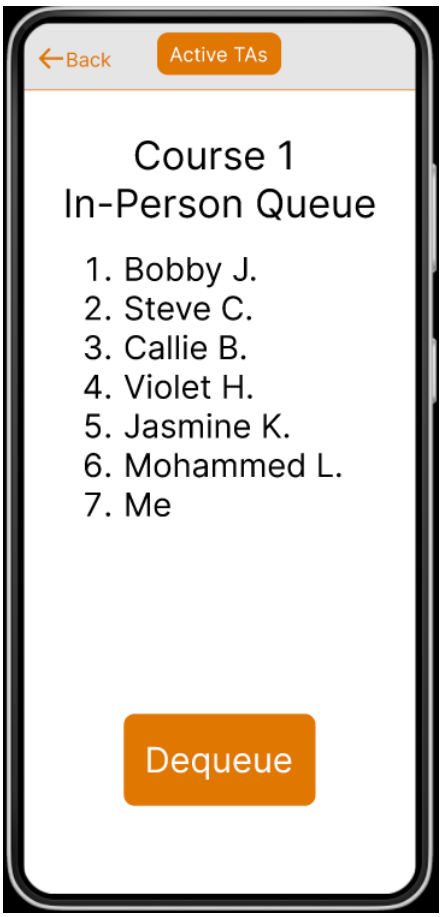

Figure 1.5: Student View (Enqueued) - Print Queue

Figure 1.6: Student View (Enqueued) - Active TAs

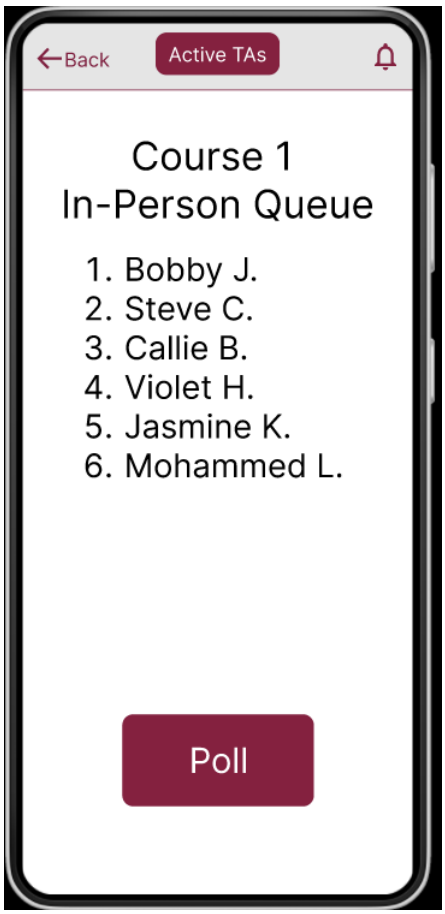

Figure 1.7: TA View - Print Queue

Figure 1.8: TA View (Enqueued) - Active TAs

1.3 Communication UI

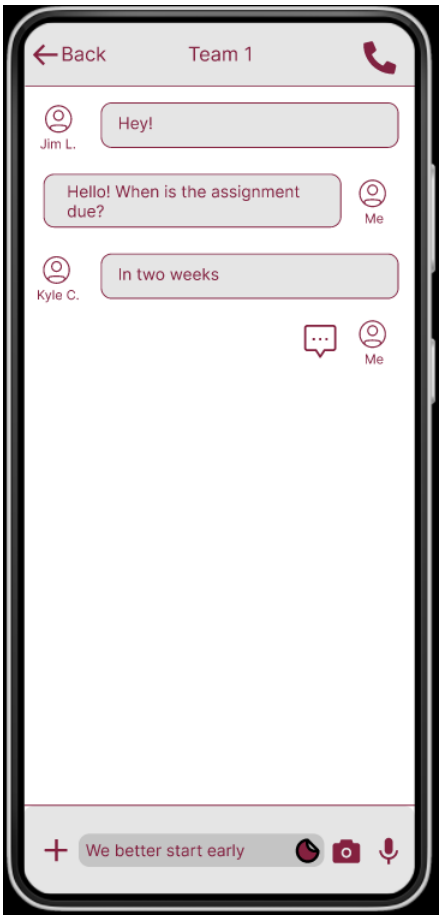

Figure 1.9: Group chat among team members

Figure 1.10: Neutral Call Screen

Figure 1.11: Neutral Call Screen (Muted) (Deafened)

```
// Use of zero-parameter Lambda Expression
case "0":
    empty.run();
    break;

// Use of single-parameter Lambda Expression
case "1":
    String message = writeMessage.run("Tada!  This is the result of
    another Lambda Expression");
    message = exclaim.run(message);
    System.out.println(message);
    break;

// Use of two-parameter Lambda Expression
case "2":
    int a = 99;
    int b = 101;
    int product = multiplication.run(a, b);
    System.out.println("The product of " + a + " and " + b + " is "
    + product);
```

Figure 1.12: Shared Screen Over Call

```
// Use of zero-parameter Lambda Expression
case "0":
    empty.run();
    break;

// Use of single-parameter Lambda Expression
case "1":
    String message = writeMessage.run("Tada!  This is the result of
    another Lambda Expression");
    message = exclaim.run(message);
    System.out.println(message);
    break;

// Use of two-parameter Lambda Expression
case "2":
    int a = 99;
    int b = 101;
    int product = multiplication.run(a, b);
    System.out.println("The product of " + a + " and " + b + " is "
    + product);
```

Figure 1.13: Shared Screen Over Call (Edit Screen Menu Open)

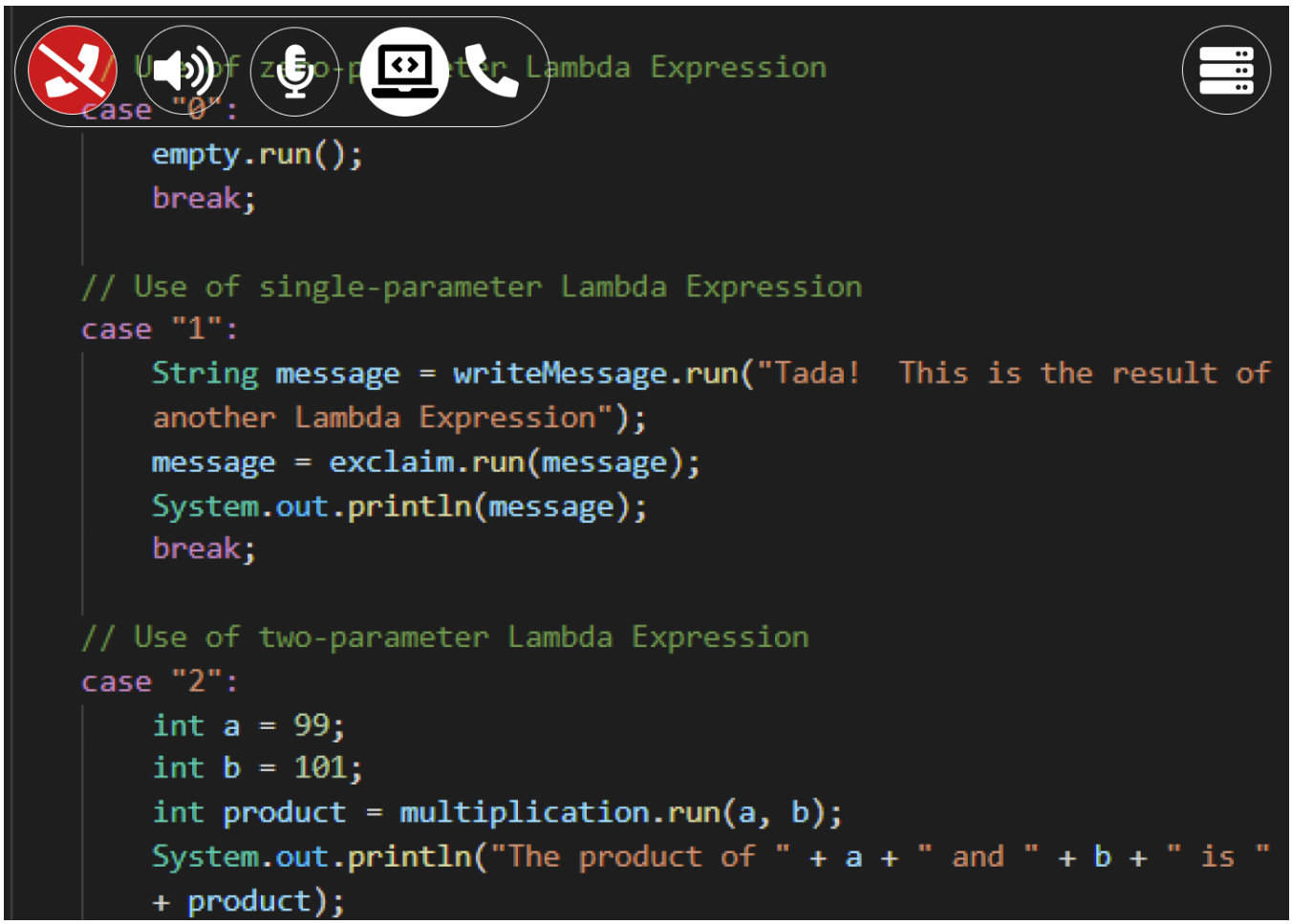

Figure 1.14: Shared Screen Over Call (Edit Screen Menu Open)

Chapter 2

Algorithm Design

2.1 Course - addStudent(Student stu) : boolean

//**this** is-a Course

```
if (stu is valid && stu paid tuition && this.prereq has been met && stu.major matches required)
    if class.size != FULL :
        classRoster.add(stu)
    else
        return class full error (False)
else
    return cannot add student error (False)
return success
```

2.2 Student - addCourse(Course c) : boolean

//(precond: addStudent was successful)
//**this** is-a Student

```
if (this.classSchedule is NOT full && c.semester is current semester):
    this.classSchedule.add(Course)
homePage.add(c.icon)
```

2.3 CourseFaculty - gradeAssignment(Assignment a, Student stu, Course c) : boolean

```
int score = 0
if (stu is enrolled && stu is in c.roster):
    compare a.submissions(stu) to a.answerKey
    for (int i = 0; i ¡ assignment.numQuestions; i++):
        if (submission.answer[i] == a.answerKey[i]):
            score += question
    a.submissions(stu).grade = score
    double grade = 0
    for (Assignment x : stu.assignments):
        grade += (x.grade/x.total)*x.weight
    stu.setGrade(c, grade)
```

2.4 PriorityQueue - enQueue(Student stu) : boolean

```
//this is-a PriorityQueue
    if (stu is in this.course && !(banList.contains(stu))):
        this.size++
        this.arr[size] = new QueueNode(stu, this.enqueueList.getNum(stu))
        heapify()
        queue.printStatement = queue.toString()
        display queue.printStatement
```

2.5 PriorityQueue - heapify() : void

```
//this is-a PriorirtyQueue
    for (int i = this.size() / 2; i >= 0; i–):
        siftDown(i)
```

2.6 PriorityQueue - siftDown(int pos) : void

```
//this is-a PriorityQueue
    while (!isLeaf(pos)):
        int j = leftChild(pos)
        if (j > this.size()):
            break
        if ((j < last) && (this.arr[j].compareTo(this.arr[j + 1] > 0)):
            j++
        if (this.arr[pos].compareTo(this.arr[j]) <= 0):
            return
        swap(this.arr, pos, j)
        pos = j
```

2.7 PriorityQueue - hogAlert(Student stu) : boolean

```
//this is-a PriorityQueue
    this.alertTA(stu.name + " is Hogging TA, please use caution")
    //Creating new Time when student can enqueue again
    Time unbanned = new Time(NOW + 3600)
    this.banList.add(stu, unbanned)
    return true
```

2.8 User - call(ArrayList<User> list) : Call

```
//this is-a User
    Call call = new Call(this)
    for (User u in list):
        if (u is in this.contacts):
            call.addUser(u)
    call.notifyUsers()
    return call
```

2.9 Call - screenShare(User u) : boolean

//**this** is-a Call

```
if (this.users.contains(u)):
    ScreenStream stream = u.screen
    if (stream == null):
        return false
    this.displayScreen(stream)
    return true
```

2.10 User - forumPost(String body, Folder files, Course c) : boolean

//**this** is-a User

```
if (c.students.contains(this) || c.faculty.contains(this)):
    Post post = new Post(this, body, files)
    return c.posts.add(post)
```

2.11 User - clockIn(Course c) : boolean

//**this** is-a User

```
if (c.faculty.contains(this)):
    Log log = new Log(this, NOW)
    this.logs.add(log)
    return true
return false
```

2.12 User - clockOut(Course c) : boolean

//**this** is-a User

```
if (c.faculty.contains(this)):
    Log log = this.logs.get(this.logs.size())
    log.end()
    return true
return false
```