Auto Attendance: A Mobile App for Automatic Attendance Taking and Rapid Contact Tracing

Amy Lin

Princeton High School
Princeton, New Jersey

Abstract

Speedy contact tracing was critical to contain the spread of COVID-19. However, in many schools, contact tracing has been done manually and often took days, especially during the peaks of the epidemic, because of the large number of daily cases and limited staff available to do contact tracing. To solve this problem, I built a mobile app named *Auto Attendance* for automated attendance-taking and rapid contact tracing. In this app, I designed and implemented person identification through QR Code recognition, classroom and roster setup features, and storage and retrieval of attendance records from local and cloud databases. The app was tested, and the contact-tracing results were both accurate and rapid. Data was collected and retrieved from the databases in under a second.

1. Introduction

In the past two years, COVID-19 largely affected our way of living and learning. Contact tracing became a key practice to contain and prevent the spread of the virus. As students returned to in-person learning amid virus spread, schools relied on contact tracing to alert close contacts of infected students and staff.

In my school district, similar to many others, school nurses performed contact tracing daily. When a student was infected, they would report their case to the school. After receiving a list of positive cases, the school nurses pulled out every student's attendance sheets, looked through classroom seating charts to find every close contact of the infected student, and then repeated this task for each student's entire class schedule. During COVID surges, there were large numbers of daily cases and limited staff available to trace close contacts. As one could

imagine, this made contact tracing extremely slow, often taking several days to complete, even though it was the most needed time for timely contact tracing. For the broader society, general contact tracing also faced challenges. Recent work showed that volunteer-run contact tracing programs fell short for CDC timeliness and yield [1].

Rapid contact tracing was a key strategy in slowing down virus spread. Gardner and Kilpatrick showed that by reducing the contact tracing process from 5 days to 2 days, the transmission rate of the virus is reduced by 73% [2]. Technology such as web-reporting tools and mobile apps could help achieve rapid contact tracing. Jian, Cheng, Huang and Liu showed that a self-reporting web-app for COVID infections increased secondary cases detected via contact tracing to 88%, and could reduce the transmission rate of the virus to less than 1 transmission per infected person [3]. Additional modelling and simulation by Almagor and Picascia concluded that if 80% of the population were to use a contact tracing app, the percentage of the population infected would decrease from 45 to 15%. With 40-60% of the population using the contact tracing app, overall infections would decrease to 22-27%, and cases at the peak of the epidemic would be reduced by 70-85% [4].

However, at the time of building this app, there were no known apps for contact tracing for school environments. Thus, I aimed to build a new mobile app incorporating both contact-tracing and automated attendance capabilities, for use in school settings. If instantaneous contact tracing was implemented, schools would be much more effective in controlling the spread of the virus.

2. Problem

Contact tracing in schools was being done manually by school nurses. This had three issues. First, manual contact tracing resulted in delayed identification and notification of close contacts. Second, it had room for human error. Third, its speed depended on the amount of cases and number of school nurses available. Often, infected students and close contacts were notified days after positive cases were reported. During the COVID surge last year, parents became increasingly concerned over this issue.

For instant contact tracing, digital records of student attendance information should be collected. In my school, teachers have been hand-taking attendance by paper and pencil. There were three drawbacks to this: First, manual attendance-taking needed additional time to convert to digital records. Second, it was also prone to human error. Third, manual attendance-taking took away valuable learning time from students. In my experience, teachers took approximately 5-7 minutes each class for manual attendance taking and waiting for late students. That translates to a loss of around 40 minutes of learning in my school day of 8 classes.

In addition, contact tracing and attendance had many blind spots in school environments. This included after-school clubs, common areas, school libraries, and school buses. No contact tracing was conducted in those places, which meant people were not aware if virus spread occurred in those areas. The lack of attendance-taking on school buses was also an issue to elementary school students. Attendance-taking could keep parents informed of the whereabouts of their children.

3. *Auto Attendance* Mobile App Design

3.1 Overview

The mobile app I built is called *Auto Attendance*. It is a multiplatform app that works on both Android and iOS devices. It is designed for students to record their own attendance automatically once they step into a classroom, and in turn teachers and administrators are able to use the attendance data to perform contact tracing instantly, within the same app.

The way this app is used is described as follows. Teachers install the app on their mobile phone. Alternatively, a dedicated mobile device with this app installed is placed in each classroom. Teachers launch the app and set up the classroom name and period for their class. Each student is given a personal QR Code. Each time they enter a new class, students scan the QR Code into the app, recording their attendance automatically. Administrators install the app on their devices for contact tracing purposes. To retrieve a list of close contacts to a positive COVID case, and all of their respective classes, they launch the app and enter the name of the

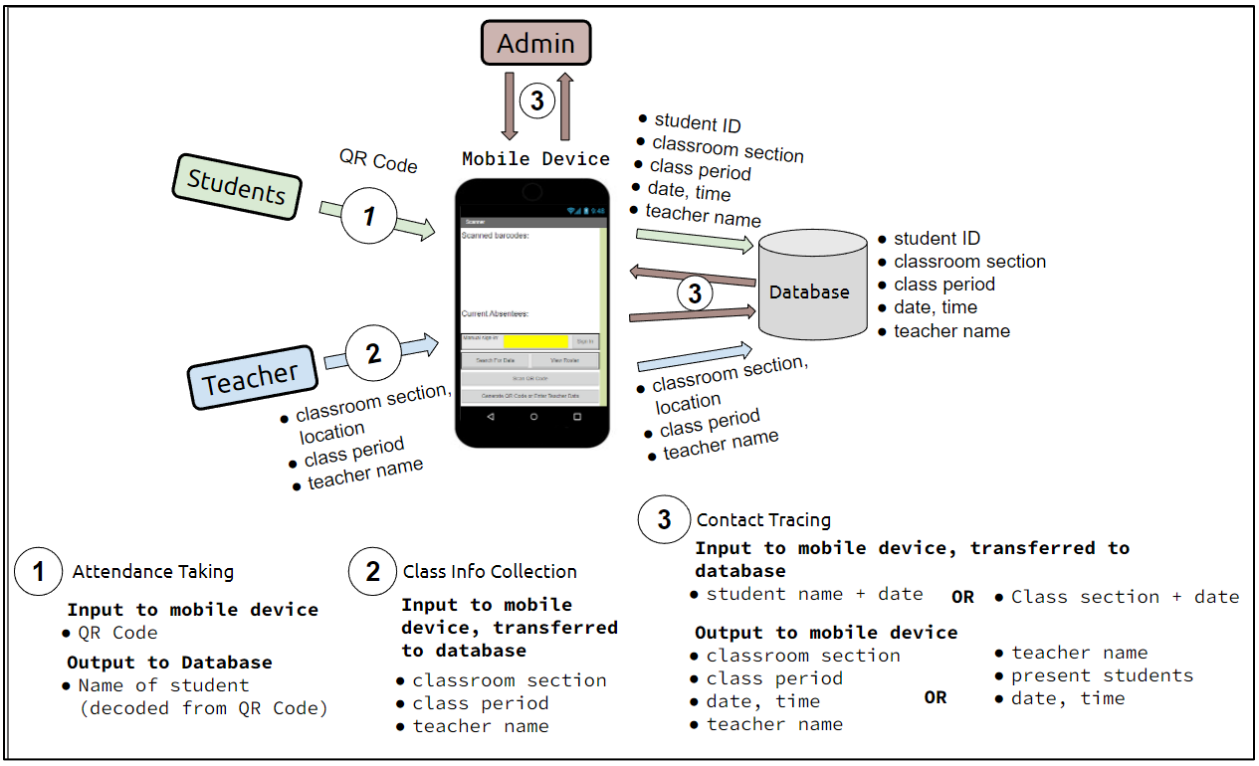

Figure 1. Logic Flow and Main Modules of Auto Attendance App

infected student and the date of infection into the app. The app will display all close contacts instantly.

Figure 1 shows the logic flow and main modules of this app: Student Attendance Recording, Class Information Collection, and Contact Tracing. The app interfaces three types of users, students, teachers and administrators. When students scan their QR Code, their ID is acquired and stored in a database. When teachers set up the classroom name and class period, the time and location information of the class is collected and stored in a database as well. When administrators perform contact tracing, the app uses student ID and date of attendance to retrieve close contacts who attended the same classes with the infected person on the specified date.

3.2 Design Process

Auto Attendance and *Contact Tracing* were originally designed as two different apps. *Auto Attendance* was built as an attendance-taking app to save time wasted on manual attendance-taking. *Contact Tracing* was initially built as a separate app. I recorded a user's location using the GPS sensor on the mobile device. However, there were several limitations to this method. First, the GPS sensor was inaccurate by a few meters. Second, this app would not be feasible in school environments, since the entire student population needed to run the app all the time for it to work effectively. Third, students might not give permission for the app to constantly record their location.

After noticing these issues, I realized that the attendance data collected with the *Auto Attendance* app could be used in rapid contact tracing. The digital records stored in the databases allowed for a quick search of student attendance records. This process retrieved the data needed for contact tracing. Because of this, contact tracing capabilities were added to the attendance app. *Auto Attendance* became an app that performs both automatic attendance-taking and rapid contact tracing.

To build this app in a timely manner, I used the *MIT App Inventor* platform to code the app. *MIT App Inventor* uses the drag-and-drop coding language *Yail* to create multiplatform apps. Creating apps in *MIT App Inventor* drastically decreased the development time, allowed created apps to run on both Android and iOS devices, and allowed access to many useful sensor components without need of implementing them.

In constructing this mobile app, I implemented the three main features and five screens interfacing the three types of users: students, teachers and administrators. The main features include an Attendance Taking component, a Class Information Collection component, and a Contact Tracing component. Among the five screens, three screens collect data and two handle data retrieval:

- Screen 1: Attendance Screen
- Screen 2: Class Information Screen
- Screen 3: Roster Setup Screen
- Screen 4: Student Data Search Screen

- Screen 5: Contact Tracing Screen

These screens will be described more in depth in the following sections.

3.3 Attendance-Taking

Three app screens in *Auto Attendance* contribute to its attendance-taking feature: Attendance Screen, Class Information Screen, and Roster Setup Screen.

3.3.1 Attendance Screen (*Screen 1*)

In *MIT App Inventor*, the *Attendance Screen* (Screen1) is the first screen the user will see upon launching the app. On this screen, there are options for attendance-taking, viewing present and absent students, and buttons leading to other app screens. Students are the primary users interacting with this screen.

The Attendance Taking component performs person identification. Attendance is done by having a student scan their personal QR Code or entering their name into the app. I selected these two methods because they are accurate and without privacy concerns. Other approaches for person identification, such as face recognition and speech recognition, are less accurate and use private biometric information. To enable the QR Code scanning feature, the Barcode Scanner component within *MIT App Inventor* is used, allowing the app to decode QR Codes into a string of text. A Textbox and Button component is also provided in case students need to enter their name.

While the app records the user's name, it simultaneously records the classroom section, class period, and teacher name, as well as the date and time of sign-in. The information of the classroom section, class period, and teacher name is pulled from the *Class Information* screen, which will be explained in detail in the next section. To retrieve the current date and time, the Clock component from *MIT App Inventor* is used and its instant is set to the code block "call Clock1.now". Time is recorded in "MM/dd/yyyy hh:mm:ss" format. The CloudDB component of *MIT App Inventor* is used to store the attendance data into a cloud database, in tag-value pairs. Two tag-value pairs were stored in the database. The first tag is set to the student name and its

corresponding value is set to a list containing the classroom name, period, date and time, and teacher name. This pairing of data allows retrieval of a student's attendance history from the student's name and date of attendance. The second tag is set to the teacher's name. Its corresponding value is set to a list containing the class information and the student's name. The second pairing of data allows retrieval of a class's attendee records from the teacher's name and class date. The data retrieval method is explained below in *Student Data Search Screen* and *Contact Tracing Screen*.

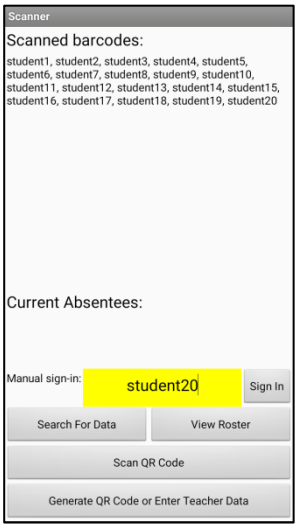

Figure 2. Attendance Screen (Screen1) App Screen

To determine present students, a function is implemented to display all names entered or scanned by QR Code. A second function is implemented to determine absent students, by collecting student names from the class roster, but not in the list of present student names. The class roster will be explained in depth in *Roster App Screen*. To display present and absent students, the Label component in *MIT App Inventor* is used to show an immutable string of text. The pseudocode for the Attendance Screen is described below:

```
when scan_button.Click:
        call BarcodeScanner1.DoScan()

when BarcodeScanner1.Afterscan:
        call CloudDB1.GetValue(tag=BarcodeScanner1.result)
        present_students_label.text.append(BarcodeScanner1.result)

when sign_in_button.Click:
        call CloudDB1.GetValue(tag=sign_in_textbox.text)
        present_students_label.text.append(sign_in_textbox.text)
```

```
when CloudDB1.GotValue:
        CloudDB1.AppendValueToList(tag=studentName,
        itemToAdd=[Clock1.Now, teacherName, classroomName,
        classPeriod]
        CloudDB1.AppendValueToList(tag=teacherName,
        itemToAdd=[studentName, classroomName, classPeriod]
        call retrieve_absentees()
```

3.3.2 Class Information Screen (*Screen 2)*

The Class Information screen includes features for entering class information, such as classroom name and section, class period, and teacher name, as well as a feature for generating student's personal QR Codes. Teachers are the primary user of this screen.

A set of Textbox and Button components is implemented for users to enter the classroom name and section, class period, and teacher name, each entered in a separate textbox. The class information is saved into a local database in a tag-value pair using the TinyDB component from *MIT App Inventor*. The values of classroom name and section, class period and teacher name are saved under the tags "classroom name", "period", and "teacher name", respectively. The local database is used solely for saving and using variables between screens. The class information entered on this screen is used by the Attendance Screen for attendance-taking.

To implement the in-app generation of personal QR Codes, the WebViewer component from *MIT App Inventor* is used to access a website, https://goqr.me/, for generating QR Codes within the app. Textbox and Button components were created to collect the user's name. The app provides a base URL, "https://api.qrserver.com/v1/create-qr-code/?" , appended with the QR Code size and the student name, to the WebViewer component, which in turn displays the resulting QR Code on the app screen. The pseudocode for the Class Information Screen is as follows:

```
api_base_url = "https://api.qrserver.com/v1/create-qr-code/?"
size = "size=150x150&data="
name_data = entered_studentName_textbox.text

when enter_teacher_button.Click:
        TinyDB1.StoreValue(tag="teacher name", value=teacherName)

when enter_classroom_button.Click:
        TinyDB1.StoreValue(tag="classroom name", value=classroomName)

when enter_period_button.Click:
        TinyDB1.StoreValue(tag="period", value=classPeriod)
```

```
when generate_qr_button.Click:
        replace_all(text=studentName, segment=“ ”, replacement=“+”)
        WebViewer1.GoToUrl(api_base_url + size + name_data)
        WebViewer1.display(webpage)
```

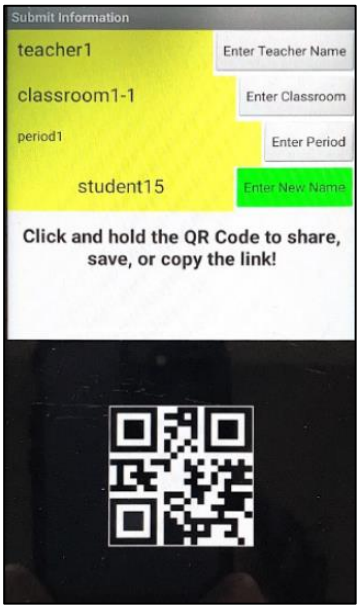

Figure 3. Class Information Screen

3.3.3 Roster Setup Screen (*Screen 3*)

The purpose of the Roster Setup screen is to set up a class roster. The primary users of this screen are teachers, who enter, edit and remove students from the roster. The roster list is stored in the same cloud database used by the Attendance Taking screen. The class roster is used to determine and display absent students on the Attendance Taking screen.

To implement the Roster Setup screen, the Textbox and Button components are used for users to enter names into the app. When the user records a name onto the roster, the name is appended to a roster list of student names. The roster data is saved as a value, with a corresponding tag of a list containing the class information. When the user removes a name from the roster, the name is removed from the roster list stored in the database. The Attendance Taking screen pulls this information from the database to show absent students. The pseudocode for the Roster Setup screen is listed below:

```
teacherName = TinyDB1.GetValue(tag=”teacher name”)
classroomName = TinyDB1.GetValue(tag=”classroom name”)
classPeriod = TinyDB1.GetValue(tag=”period”)

when roster_add_button.Click:
        CloudDB1.AppendValueToList(tag=[teacherName, classroomName, classPeriod],
        itemToAdd=roster_add_textbox.text
        roster_label.text = rosterList

when roster_remove_button.Click:
        CloudDB1.StoreValueToList(tag=[teacherName, classroomName,
        classPeriod],   itemToAdd=rosterList.remove(roster_remove_textbox.text))
```

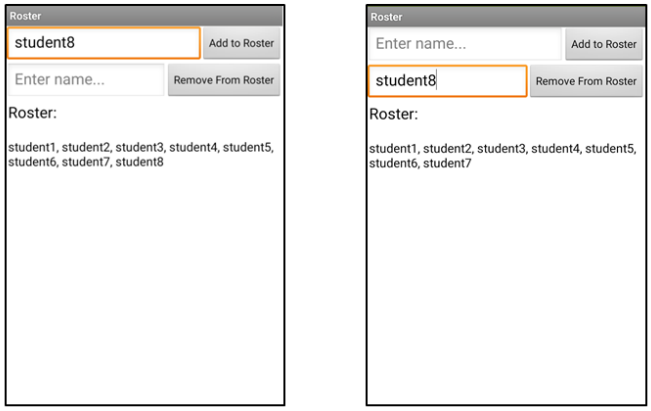

Figure 4. Roster Setup Screen (Left: adding a student; right: removing a student.)

3.4 Contact Tracing

Contact tracing is performed through the *Student Data Search* and *Contact Tracing* screens. The primary users for these screens are administrators. The *Student Data Search* screen allows users to search for students’ attendance history, while the *Contact Tracing* screen allows search of the attendees of a teacher’s classes.

3.4.1 Student Data Search Screen (*Screen 4*)

The Student Data Search screen allows students’ class attendance history to be retrieved, including the classroom names, class periods, date and time of sign-in, and teacher name, from student name and date of attendance. Again, users can enter a student’s name and attendance date through the Textbox and Button. Data retrieval is performed through the CloudDB component, and the Label component is used to display the student’s attendance history. The

student name entered into the Textbox component is searched in the cloud database as a tag, and the value is retrieved as a list of attendance history and class information. If an item in the attendance history list contains the date entered, the item is displayed on the screen. This algorithm is able to display multiple sign-ins from the same student. The pseudocode for the Student Data Search screen is shown below:

```
month = month_textbox.text
date = date_textbox.text
year = year_textbox.text
studentName = studentName_textbox.text

when search_button.Click:
        CloudDB1.GetValue(tag=studentName)

when CloudDB1.GotValue:
        for item in CloudDB1.value:
                if item[0] contains (month + ”/” + date + ”/” + year):
                        student_data_label.text.append(item)
```

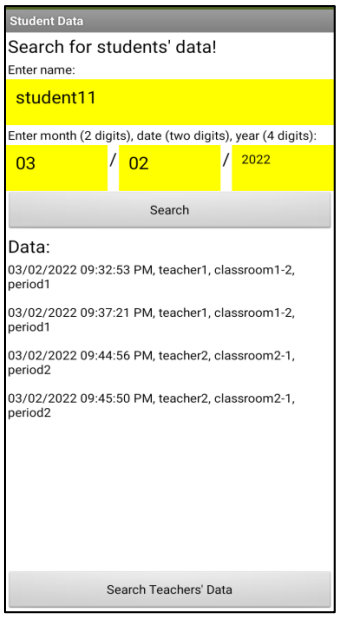

Figure 5. Student Data Search Screen

3.4.2 Contact Tracing Screen (*Screen 5*)

The Contact Tracing screen allows users, primarily administrators, to search the attendance of a teacher’s classes given a teacher’s name. This screen is similar to the Student Data Search screen. The Textbox and Button components are used to enter the teacher name, class date, and classroom section into the app. The entered teacher name becomes a tag and its

corresponding value is searched in the cloud database, and displayed on the app screen if the value matches with the entered date and class section. This algorithm is able to display multiple student sign-ins. The pseudocode of the Contact Tracing screen is displayed below:

```
month = month_textbox.text
date = date_textbox.text
year = year_textbox.text
teacherName = teacherName_textbox.text
classroom = classroom_textbox.text

when search_button.Click:
        CloudDB1.GetValue(tag=teacherName)

when CloudDB1.GotValue:
        for item in CloudDB1.value:
                if item[1] contains (month + ”/” + date + ”/” + year) and if item[2] contains classroom:
                        teacher_data_label.text.append(item)
```

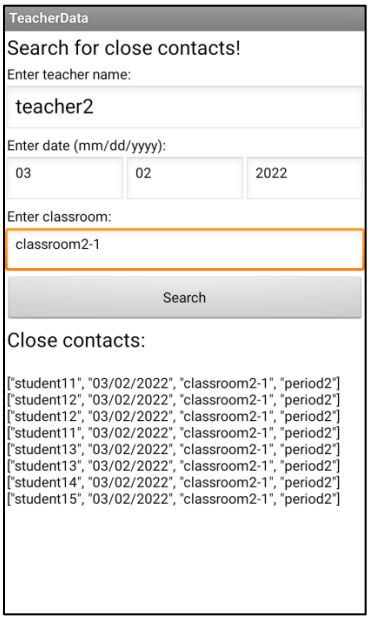

Figure 6: Contact Tracing Screen (a list of close contacts is displayed on the screen)

4. Tests and Results

Tests were performed to verify if the *Auto Attendance* mobile app correctly collected attendance data and retrieved the attendees and the corresponding attendance records in a timely manner. A total of 40 students were used in the tests, named as “student1” through “student40”. Each student was issued a unique QR Code and assigned randomly to different classrooms and class periods. Teachers, named “teacher1” through “teacher4”, were associated to each

classroom and class period. Each classroom had different class sections, for example, “classroom1-1” and “classroom1-2” represented section 1 and section 2 of classroom 1. Close contacts were defined as students sitting in the same class section in the same class period as the infected student.

In the tests, all the teachers set up their class information on the Class Information Screen and Roster Setup Screen. Students signed in to their classes by scanning their QR Code. The app correctly recognized the student’s names by decoding the QR Code’s data. To cover corner cases, 5 students were randomly selected to be absent over one class period, and 10 students were randomly selected to sign in multiple times to the same class. 10 students were randomly selected as COVID cases. Their names and date of infection were entered into the Student Data Search Screen. The app correctly retrieved and displayed the attendance data and class information. On the Contact Tracing Screen, the class date, teacher name, and classroom section were entered, and close contacts of an infected student were correctly displayed.

Table 1 shows 3 examples of the test cases, 2 students and 1 teacher. They are representative of all the 50 test cases performed. The students “student15” and “student19” were used as positive cases in the tests. The app found that “student15” signed into “classroom1-2”, “period1” 4 times, and “classroom2-1”, “period2” 1 time. The app found that “student19” signed into “classroom1-3”, “period1” 2 times. In the case of “classroom2-1”, “period2”, taught by “teacher2”, the app found “student11” through “student14” as close contacts of “student15”. In all tests, the app correctly retrieved the student’s attendance records and close contacts in under a second.

The app is published in the MIT App Inventor Gallery [8], searchable under the name “Auto Attendance”.

5. Conclusions and Future Work

This app was built and tested to be a useful tool for automatic attendance-taking and rapid contact tracing. The tests showed that this app accurately retrieved attendance records and close contacts in under a second. Automated attendance-taking and contact tracing reduces the workload on teachers and nurses. Using this app, potential error made in manual contact tracing

is eliminated. Students and teachers can save a substantial amount of time every day for learning. A future work is to extend this app for use on school transportation to track student riders.

Through this project experience, I discovered many publicly available technologies, such as *MIT App Inventor*, which can solve various problems in the community. It was a learning experience that encouraged me to delve further into developing apps to help the community.

Table 1. Accuracy of Data Retrieved from Auto Attendance Mobile App

Test Case Information	**Data Retrieved**	**Accuracy**
Name entered: student15 **Date entered:** 03/02/2022	03/02/2022 09:33:13 PM, teacher1, classroom1-2, period1 03/02/2022 09:33:17 PM, teacher1, classroom1-2, period1 03/02/2022 09:33:18 PM, teacher1, classroom1-2, period1 03/02/2022 09:37:36 PM, teacher1, classroom1-2, period1 03/02/2022 09:45:59 PM, teacher2, classroom2-1, period2	correct
Name entered: student19 **Date entered:** 03/02/2022	03/02/2022 09:40:14 PM, teacher1, classroom1-3, period1 03/02/2022 09:42:58 PM, teacher1, classroom1-3, period1	correct
Name entered: teacher2 **Date entered:** 03/02/2022 **Classroom entered:** classroom2-1	[“student11”, “03/02/2022”, “classroom2-1”, “period2”] [“student12”, “03/02/2022”, “classroom2-1”, “period2”] [“student12”, “03/02/2022”, “classroom2-1”, “period2”] [“student11”, “03/02/2022”, “classroom2-1”, “period2”] [“student13”, “03/02/2022”, “classroom2-1”, “period2”] [“student13”, “03/02/2022”, “classroom2-1”, “period2”] [“student14”, “03/02/2022”, “classroom2-1”, “period2”] [“student15”, “03/02/2022”, “classroom2-1”, “period2”]	correct

Bibliography

[1] Shelby T, Schenck C, Weeks B, Goodwin J, Hennein R, Zhou X, Spiegelman D, Grau LE, Niccolai L, Bond M, Davis JL, "Lessons Learned From COVID-19 Contact Tracing During a Public Health Emergency: A Prospective Implementation Study", *Front. Public Health* 9:721952 (2021).

[2] Gardner, BJ and Kilpatrick, AM, "Contact tracing efficiency, transmission heterogeneity, and accelerating COVID-19 epidemics", PLoS Computational Biology, 17(6), Jun 17, 2021

[3] Shu-Wan Jian, Hao-Yuan Cheng, Xiang-Ting Huang, Ding-Ping Liu, Contact tracing with digital assistance in Taiwan's COVID-19 outbreak response, International Journal of Infectious Diseases, Vol. 101, 2020, Pages 348-352.

[4] Almagor, J, Picascia, S, Exploring the effectiveness of a COVID-19 contact tracing app using an agent-based model, *Sci Rep* **10,** 22235 (2020).

[5] Kanu FA, Smith EE, Offut-Powell T, Hong R, Delaware Case Investigation and Contact Tracing Teams, Dinh TH, Pevzner E, "Declines in SARS-CoV-2 Transmission, Hospitalizations, and Mortality After Implementation of Mitigation Measures—Delaware, March-June 2020", MMWR Morb Mortal Wkly Rep 2020;69:1691–1694.

[6] MIT App Inventor, https://appinventor.mit.edu/

[7] Go QR, QR Code Generator, https://goqr.me/

[8] https://gallery.appinventor.mit.edu/?galleryid=2fa33621-7437-4fe4-b9fe-16fb5de5b0f6