# Trojan Check In/Out

Team 10

Ehsan Daya - 3188995525
Austin Huang - 6598238946
Vikram Kher - 4895509565
Michelle Ran - 6998590265
Rithwik Sivakumar - 5707657182
Winston Wei - 7458310532

Table of Contents

# Preface

This document describes the changes to the implementation of Team 10's Trojan Check In/Out application. The document is separated into sections based on the type of deviation from our original design document and requirements document. These sections are as follows: Architectural Change, Detailed Design Change. Also included is a section titled "Requirement Change" where two scenarios that were not accounted for in the original requirements are presented. Within the section is a description of how the project would change to accommodate for the scenarios.

# Introduction

The Trojan Check In/Out Android application is designed to help lessen the risk of Covid-19. The disease spreads very fast and in high-population areas such as the USC campus, the spread of the disease would be even faster. The application does this by allowing students to check in and out of buildings. The check in is done on the application by scanning a manager-provided QR code that will be at a building's entrance. The check-out is done on the application by the student once they have left the building (or by scanning the qr code again). The application also allows managers to control the capacity of buildings and see which students are inside buildings at any given time. This is beneficial because it allows buildings to have a safe number of students at all times. This also helps with contact tracing because the application keeps track of student building check in and check out history. Important to note is the fact when a building is at full manager-set capacity, the application prevents users from checking in - thereby limiting the building capacity. The application also has a full login/register system built in for security purposes (managers need to know which student is checked into a building when they check in). This prevents students from logging in as managers and vice versa. In short, this document describes the deviations from Trojan Check In/Out's previous documentation when compared to the developed application. These deviations come in the form of deletions, additions, and modifications to the original detailed and architectural designs in addition to scenario-based potential changes in the Requirements Change section.

# Architectural Change

We chose not to require authentication for database access to make testing easier.

# Detailed Design Change

## Package Diagram

We chose not to put Activities and Fragments into their own package because we didn't need that level of granularity for importing them. We also chose to use an additional external package, `com.github.kenglxn.qrgen`, to simplify generating a QR code.

## Class Diagram

### Server

Removed `majors: String[]` and `getMajors(): String[]`. A list of majors is only needed to populate Spinners, which a string array in strings.xml would suffice for.

Removed `buildingCache: Map` and `studentCache: Map`. We chose not to implement a cache to avoid coherency problems.

Added constant strings for collection names and a log tag for convenience.

Added `listenForCheckedInStudents` method. We originally used `filterRecords` to get all the students checked into a building, but we felt it was too hacky.

Modified `getCurrentUser(): User` to `getCurrentUser(callback: Callback<User>): void`. We need to fetch the user's data from Firestore, which is an asynchronous operation, and so can't immediately return the User object.

Modified `logout(callback: Callback<Void>): void` to `logout(): void`. Firebase signs the user out synchronously so there's no need to call back.

Modified `registerStudent` and `registerManager` to take a `file: Uri` parameter. We decided to require the user to upload a photo when registering.

Replaced `deleteAccount` with separate `deleteStudent` and `deleteManager` methods. Because the deletion logic for student vs. manager accounts is different and we always know the user's account type when they select the "delete account" option, we can call the method specific to that account type and skip the check logic.

Modified `checkIn` and `checkOut` to take a `Callback<Building>` parameter instead of `Callback<Void>`. It's easy to pass the Building out of the method, so we may as well in case it's needed in the future.

Modified `checkOut` to remove the `buildingId: String` parameter. We assume the user is always checking out of their current building, which is known.
Renamed `searchHistory` to `filterRecords` for consistent terminology.

## User

Added `getUid(): String`. This is the id under which a user's data is stored in Firestore and is needed for some operations.

## Student

Removed non-default constructor. We never directly instantiate Student objects.
Added `uid: String` and `getUid(): String`. This is needed for instantiating Records.
Modified `id: int` to `id: String` and `getId(): int` to `getId(): String`. We never perform numerical operations on a student id.
Modified `setBuilding` to be public. We need to update a local copy of a Student object in their profile to support checking in and out in the same session.

## Manager

Removed non-default constructor. We never directly instantiate Manager objects.
Added `uid: String` and `getUid(): String`. This isn't currently needed but might be.

## Building

No changes.

## Record

Added `studentUid` field and getter to expedite fetching a student's data (stored under uid).
Added `buildingName` and `major` fields and getters to expedite filtering.
Modified constructor to take in `student` and `buildingName` to initialize the new fields as well as avoid having to call the asynchronous `getCurrentUser` method.
Modified `time` field to long type (seconds since epoch) because Firestore's `toObject()` doesn't support LocalDateTime.

### Validator

Added `validateID(ID: String)` to validate that the input is a USC id.
Modified `validateNotEmpty` to take a variable number of String parameters so we can validate multiple strings at once.

### QRCodeHelper

Replaced `encodeToFile` with `generateQRCodeImage` which takes in a building id, output height, and output width, and returns a byte array. By not writing to a local file before uploading to Firebase Storage, we save space on the user's device. We changed the name of the method to reflect this.

### Other

Added MyApplication to initialize the Server's static variables when the application starts.
Added StartPage to present the user with login and register options.
Added StudentBasicActivity which is used to display the info of a student when their photo is selected. A separate Activity was simpler to implement than setting a flag in StudentActivity to hide options meant for logged-in users (e.g. change password).
Added StudentHistory to show the student's visit history in a separate activity.
Added ManagerActivity to host a tab layout for the profile, building list, and filter.
Renamed StudentProfileActivity to StudentActivity for brevity.
Replaced ManagerProfileActivity with ManagerProfileFragment; BuildingListActivity with BuildingListFragment; BuildingActivity with BuildingDetailsFragment; SearchActivity with FilterFragment; and SearchResultsActivity with FilterResultsFragment in order to contain them in a tab layout.
Modified StudentActivity to implement `View.OnClickListener` to trigger scan functionality instead of having a `scan(view: View)` method.

# Requirement Change

## Scenario 1

This change will call for a few changes to be made to this design. Given that GPS tracking is now necessary, we would have to add in a component that allowed for GPS tracking

and setting a geofence at the very least. We would have to attach a geofence to each building with data that outlines the footprint of each building. We would have to use location services that constantly update and set timers triggered by geofences being crossed. As such multiple parts of the existing design would need to be changed as outlined below.

New Classes andComponents would likely include a new Timer Class, GPS Tracker class, Geofence Class, and a Location class. The timer class would keep track of how long a student has been outside of or inside of a geofence based on their check in status. The GPS Tracker class would constantly update the Location class with details on the student's current location. The Geofence class would simply be attached to buildings and check the data from the Location class to ensure the geofence has or hasn't been crossed.

These new classes naturally call for an update to the existing classes and connectors. The current Student class must now contain a Location variable such that their location can be referenced at all times. Meanwhile, the current Building class must contain the Geofence and Timer classes in order to monitor and time student activity.

The overall design architecture of the application will not differ much from what is currently being implemented. However, the Functional and Non-Functional Requirements must be updated to meet these new changes. These changes are outlined below in detail.

Functional Requirements:

1. The system shall use the GPS Tracker to update the Student's Location every minute.
    1.1. The system will then evaluate the check in status of the student
        1.1.1. If the student is currently checked in, their Location will be cross referenced against all the Geofences of the building they are currently checked in at.
            1.1.1.1. If the student's location is not in the building's geofence, the building will start its countdown timer
            1.1.1.2. The system shall then check the student's check in status and location every minute following the start of its timer
                1.1.1.2.1. The countdown timer will stop and if the student's check in status becomes negative(checked out) for the building
                1.1.1.2.2. The countdown timer will stop and reset if the student's location is detected back within the geofence of the building.
                1.1.1.2.3. Once 300 seconds have elapsed. If the student's location is not within the building's geofence, and their checked in status is positive(currently checked in to that building), they will receive an alert once a minute to check out
        1.1.2. If the student is not currently checked in, the Location will be cross referenced against all the Geofences in the system's building database to ensure the student is not in a building.
            1.1.2.1. If the student is in a building, the building will start using its timer and counting down from 300 seconds.
            1.1.2.2. The system shall check the student's check in status and location every minute following the start of the countdown timer

1.1.2.2.1. The countdown will stop if the student's check in status becomes positive(currently checked in) for the building.
1.1.2.2.2. The countdown timer for the building will also stop and reset if the student leaves the building's geofence at anytime within the 300 seconds
1.1.2.2.3. Once 300 seconds have elapsed, if the student's check in status is not positive, and the student is still detected within the building's geofence, the student will be alerted to check in once a minute until the check in positive becomes positive.

2. The system shall alert the user if the app is running and wireless connection is lost for more than a minute.

Non-Functional Requirements:

1. The app shall maintain GPS functionality in the background.
2. The application shall require the use of location services
3. The application shall require a consistent wireless connection

# Scenario 2

The Trojan Check In/Out application could be swiftly adapted to meet this change in requirements. It would require some slight changes to the design of our application, however, the overall architecture of the system would remain intact. Namely, we would need to include a new Kick Out method in the Server class, which would take in a student id and check the corresponding student out from their current building. This method would also generate a record corresponding to this check out event.

We would also have to update the UI of our application to meet the new requirements. The main update would occur for manager accounts, specifically in the building list tab of the application. Currently, once a manager selects a building from the building list they are presented with a list of currently checked in students to the building along with their photos. We now include a "kick out" button next to each student's entry. If a manager presses this button, the server's new Kick Out method is called and the student would be removed from the building. Additionally, once they have been removed, we would also push a notification to the student's phone stating that they have been removed from the building.

We provide a quick summary of the new functional requirements below.

Functional Requirements

1. The system shall provide managers with the ability to remove individual students from buildings if they are not wearing masks.
    1.1. If a student is found to be in violation of USC's indoor mask policy, the system shall allow the manager to navigate to the buildings tab of the application.

1.1.1. The system shall present to the Manager a list of their buildings, with each list item consisting of the building name, maximum capacity, and current capacity.

1.1.1.1. If a Manager selects a building with a defined capacity, the system shall present a list of the names and photos of the students currently checked into the building.

1.1.1.2. The system shall now also include a “Kick Out” button next to each students entry in the list

1.1.1.2.1. If the manager clicks the “Kick Out” button, the system shall check out the student from the building.

1.1.1.2.1.1. The system shall decrement the current capacity of the Student’s current building and then set the student’s current building to null.

1.1.1.2.1.2. The system shall then inform the student of their removal from the building via a notification popup message within the application

1.1.1.2.1.3. The system shall generate a record certifying the students removal from the building.

Non-Functional Requirements:

There are no additional non-functional requirements needed to implement this new requirement.