50.001 Introduction to Information Systems & Programming

Team Number 3-5: SeatSmart

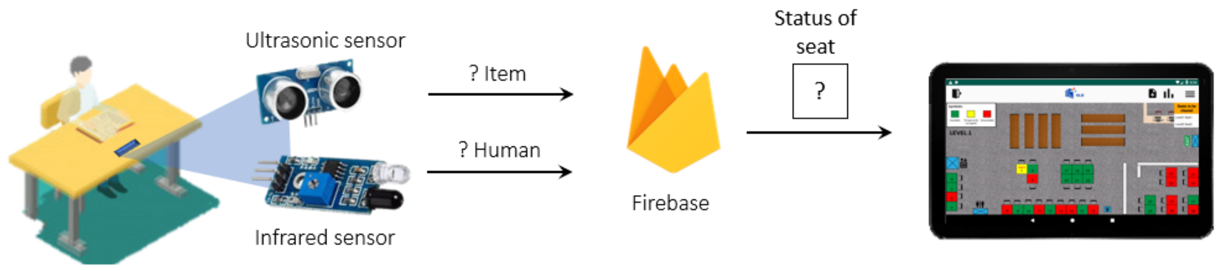
Xie Han Keong 1003876, Jisa Mariam Zachariah 1003638, Yeh Swee Khim 1003885, Loh De Rong 1003557, Jia Yunze 1003612, Lin Cheng 1003326

Background and problem

SeatSmart attempts to solve three key library associated problems:

- Library goers wasting time having to find seats physically
- Library users hogging seats with their items
- Librarians having a difficult time monitoring all the seats efficiently, and unsure if they should clear the items to make space for other users

Solution



<u>Scenario 1</u>:

The user's presence will cause the seat status to be 'Unavailable', regardless whether any item is placed on the table.

<u>Scenario 2</u>:

After a grace period of 20 minutes, if the items are still left unattended, the librarian will be alerted via a sound notification and an update on the right panel.

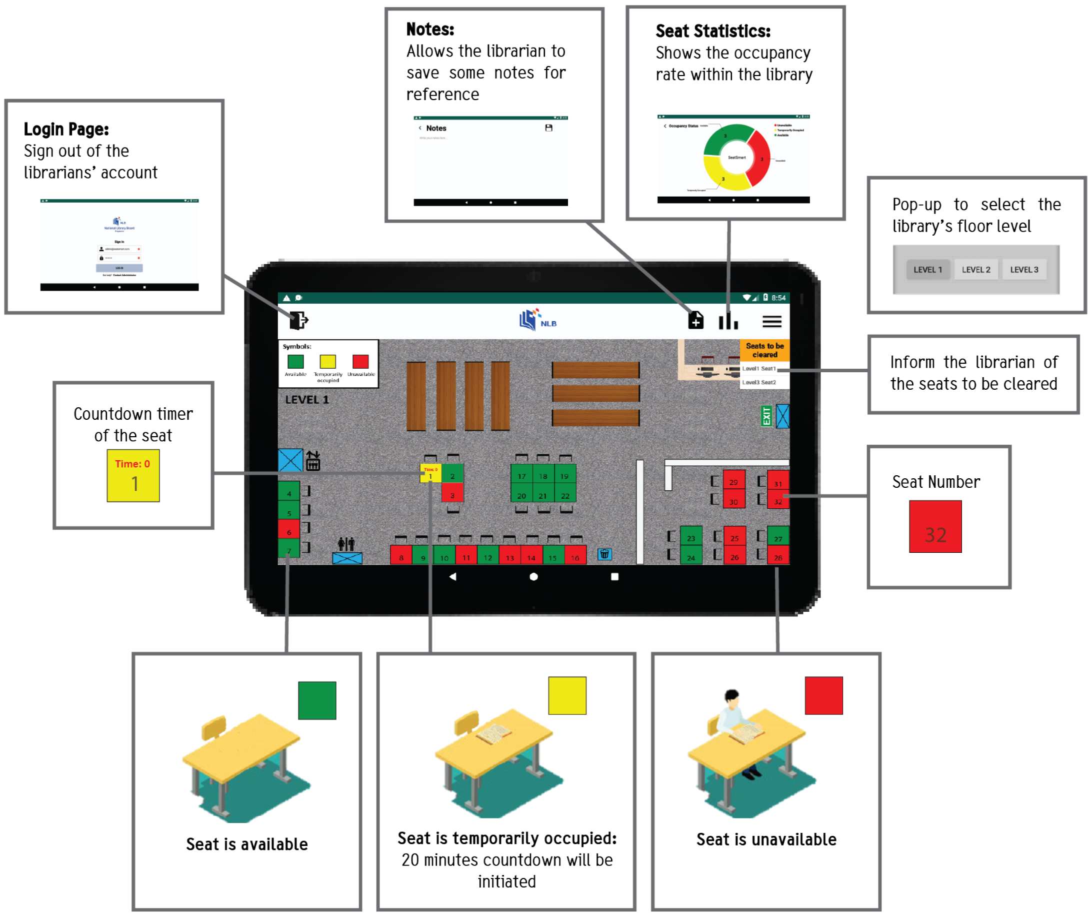
<u>Scenario 3</u>:

The seat status will become 'Available' if the librarian has cleared the item at that particular seat or if the user has left the seat with all his belongings

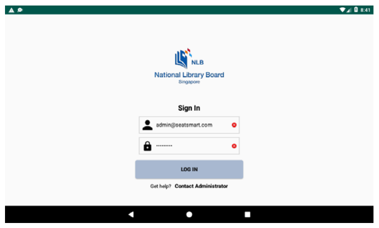
<u>Data Retrieval</u>:

To retrieve data from the sensors into the Firebase, we created a python script that reads the serial output from the arduino IDE which will then be uploaded onto the Firebase. This is a stop-gap measure as we were unable to get the Arduino Wifi Module to work.

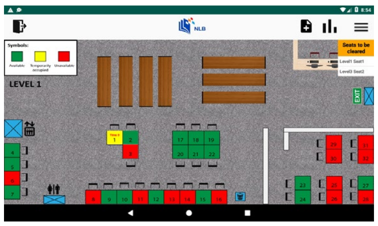
App layout


Login Page:
Sign out of the librarians' account
Notes:
Allows the librarian to save some notes for reference
Seat Statistics:
Shows the occupancy rate within the library
Pop-up to select the library's floor level
LEVEL 1
LEVEL 2
LEVEL 3
Inform the librarian of the seats to be cleared
Countdown timer of the seat
Time: 0
1
Seat Number
32
Seats to be cleared
LEVEL 1
EXIT
Seat is available
Seat is temporarily occupied:
20 minutes countdown will be initiated
Seat is unavailable

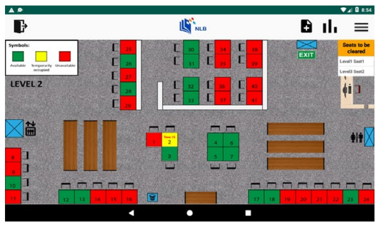
Overall functionalities in *SeatSmart*



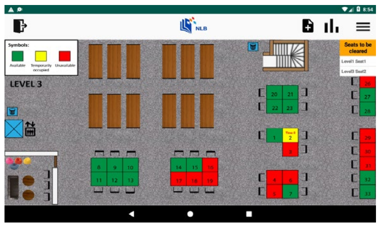
Login Page



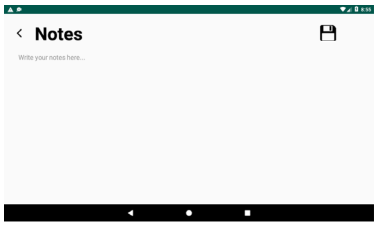
Level 1



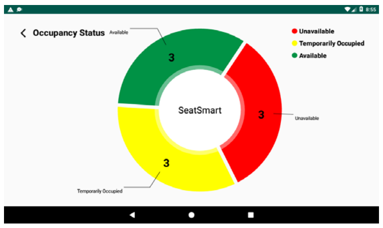
Level 2



Level 3



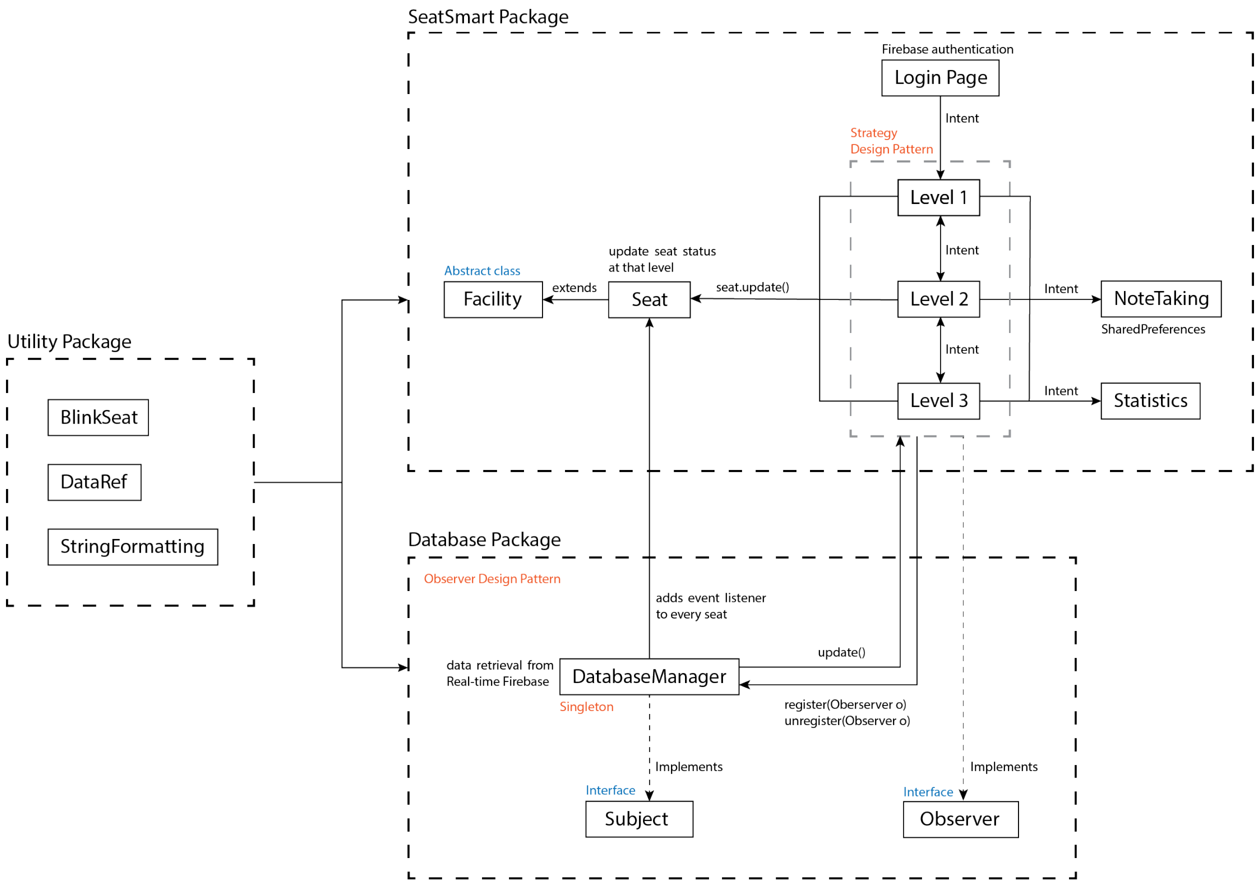
Notes



Seat Statistics

System Design and Implementation

Relationship between packages and classes



Firebase Authentication

SeatSmart utilizes email and password based authentication in Firebase Authentication SDK to create, manage and verify librarians' credentials upon login.

Real-time Firebase

Real-time data of the seat status are collected using the sensors and updated on Firebase. As we query the Firebase API to retrieve data, it is important that that UI remains responsive. The asynchronous nature of Firebase allows for the download to be done on a separate thread from the UI. Hence, the librarian is still able to interact with the app, such as going to another level, while the download continues to happen in the background.

Inheritance, Abstraction and Polymorphism

`Facility` is an abstract superclass of `Seat`. Hence, the programmer is able to perform constructor chaining easily using the `super()` method, and is also forced to provide a concrete implementation to perform the necessary updates:

- Update the seat status with the correct colour
- Update the adapter view

This is related to the concept of polymorphism because the overridden method in the subclass will be invoked at runtime, regardless of the reference data type used in the source code at compile time. In the future, other subclasses such as Printer and Meeting Room will also inherit from Facility in the same manner, and the programmer will be forced to provide their own specific concrete implementation of `update()` which will certainly help in the overall tracking process. As a way of abstraction, we want `Facility` to act as a superclass that defines a **generalization** form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. These object-oriented principles and generic programming are also heavily used as we implement the Observer Design Pattern, which will be discussed further below.

Code Modularization

The system implements a modular design with regard to the packages and classes used. Each group of classes has their own package and is not allowed to access other classes unless necessary. In particular, we achieve **encapsulation** by declaring our variables as private, and provide appropriate getter and setter methods to modify and access selected variables.

Using the **Single Responsibility Principle**, each class has a sole purpose and will call other classes to do smaller tasks. For example, the `BlinkSeat` class contains a side functionality that helps to blink the seat in the app when the librarian clicks onto a seat within the list of seats to be cleared. The `DataRef` class consists of attributes that helps to store and keep track of data. Lastly, the `StringFormatting` class helps to reformat strings where required.

Whereas the key activities in the `seatsmart` package will make use of these smaller classes to perform its intended interaction with the user interface, while classes in the `database` package will be solely in-charge of retrieving data from the Firebase. On this note, it also becomes apparent that the `utility` package was created to hide the internal details of the individual classes from the public interface.

Android Activity Lifecycle

This paradigm is useful in helping us declare the way our activity should behave at specific stages of each activity.

As will be discussed further under the Observer Design Pattern, each of our level activity is an observer which will register itself on `onStart()`and then deregister itself `onStop()`. This ensures that upon data change, only the level activity containing that particular seat will be notified and updated as it should be the only dependent.

If the librarian clicks on the seat that resides on another level on the ‘Seats to be cleared’ listview, she should be directly to that activity, followed by a seat that blinks on-screen to signal its location to the librarian. Hence, in the new activity, it must be able to receive the intent with that particular seat `id`, specifically in the `onResume()` callback since blinking can only be observed in the state in which the app interacts with the user.

Data Persistence: Shared preferences

In the Notes page, the librarian can save notes for references or to pass on messages to librarians within the next few shifts. To deliberately store the app’s data, we used `SharedPreferences` to enable data persistence. Upon clicking the save button, the notes entered will be saved.

Design patterns used

Creational Pattern: Singleton

We use a lazy instantiation to create the database. The variable is static and will only create one instance of itself. It also has a private constructor, so nothing outside the class can instantiate it. `getModel()` is the only static factory method that provides a global point of access to the singleton object and returns the instance to the caller.

Behavioral Pattern: Observer

Each of the activities in all three levels are the concrete subclasses of the interface `Observer`, which registers itself on `onStart()`and deregisters itself `onStop()`. The `DatabaseManager` is a concrete subclass of the interface `Subject`, which maintains a list of its dependents (i.e. the three level activities). With this architecture, we can model the independent functionality with the "subject" abstraction as well asl the dependent functionality with an "observer" hierarchy.

In the `DatabaseManager` class, each seat at each level creates an instance of an anonymous class implementing the `ValueEventListener` interface. Hence, upon data change, all its dependents are notified and updated automatically at run-time.

Behavioral Pattern: Strategy

The `listView` object delegates the role of retrieving the data to the `ListAdapter` object.

Discussion and Lessons Learnt

User Interface/ User Experience Design (UI/UX)

After getting the basic functionalities of the application to work, we focused more on the UI/UX by enhancing existing implementations and/or introducing additional features, such as having a clickable listview of all the seats that need to be cleared, sound notification, note-taking and statistics pages. User friendliness also comes in the smallest of ways, such as having dialogue progress bars during login or Toast messages, to provide some visual information to the user about his actions. This smoothens the transition and improves the user experience.

Continuous Testing

We learnt that it is important to consistently perform continuous testing, especially every time we create or integrate a new feature as it may break another related part of the app. To this end, debugging using `Logcat` is extremely useful. This is because the window displays messages in real-time, which helped us tremendously during data retrieval from Real-Time Firebase, and keeps a history, which helped us trace back to the source of error.

Conclusion

With the implementation of SeatSmart, we hope that it has provided a better seat management system to ensure that everyone has an equal opportunity to study conducively in the library as well as to ease the work of the librarians. We also learnt a lot about the fundamentals of Android app developments and managed to put into practice the design patterns taught in class.

Effort and Contributions of each of our team members

Xie Han Keong	Sensors & sending data to Firebase
Jisa Mariam Zachariah	Background & Layout, Clearlist, Poster
Yeh Swee Khim	Background & Layout, Seat display, Poster, Sensors & sending data to Firebase, Countdown timer, Housing of sensors, Logo & Design
Loh De Rong	Firebase, Firebase Authentication, Notes, Graph, Sensors & sending data to Firebase, Code clean up, Code Modularization, Implementation of design patterns, Sharedpreferences
Jia Yunze	Background & Layout, Level pop up, Sound notification, Housing of sensors
Lin Cheng	Firebase, Clearlist, Blink effect, Housing of sensors

References

1. https://firebase.google.com/docs/auth/android/password-auth
2. https://firebase.google.com/docs/database/admin/retrieve-data
3. https://github.com/PhilJay/MPAndroidChart
4. https://developer.android.com/guide/components/activities/activity-lifecycle
5. https://www.javatpoint.com/android-listview-example
6. Lesson 2: Shared Preferences by otakuprof