

Frith reference manual

Contents

Database	3
Mobile Application.....	3
1 Incident Report	3
1.1 Application	3
1.2 Backend.....	4
2 Start/End Shift.....	4
2.1 Application	4
2.2 Backend.....	5
3 Flashlight	6
3.1 Application	6
3.2 Development tools.....	7
3.3 Adding a package dependency	7
4 Notepad	7
4.1 Application	7
4.2 Development tools.....	9
5 Emergency Call.....	9
5.1 Application	9
5.2 Development tools.....	9
5.3 Adding a package dependency	10
6 Request Manager.....	10
6.1 Application	10
6.2 Development tools.....	11
6.3 Database	11
6.4 Backend.....	12
7 Request Backup.....	12
7.1 Application	12
7.2 Development tools.....	13
7.3 Database	13
7.4 Backend.....	14
8 SecurityGuardDB.dart	14
9 Information Pool	14
9.1 Application	14

9.2	Backend.....	15
10	Model.....	15
10.1	Application	15
10.2	Backend.....	15
11	View	15
11.1	Application	15
12	Global.....	17
12.1	Application	17

Database

Please view [Appendix D](#), which contains an entity relationship diagram, for a comprehensive view of the database architecture. [Appendix E](#) displays the implementation of the database, as well as all relationships modelled in [Appendix D](#).

Mobile Application

1 Incident Report

1.1 Application

Once on the security guard homepage clicking on the “incident report” icon, the user is taken to the create a new report screen. In the code, this is called “New_report.dart”. The flow chart of this section can be seen in the figure below (figure 1).

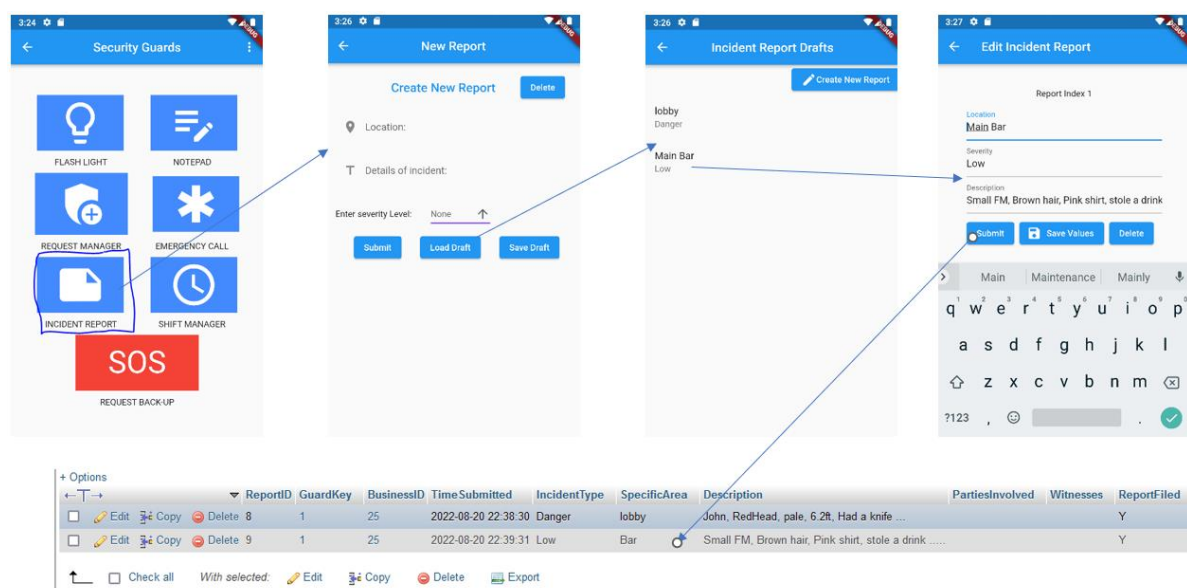


Figure 1

1.1.1 New_report.dart

This file is what builds the view for the user. This uses stateful widgets to give users options to fill out the report. It also has options for saving and loading drafts, viewing drafts, or submitting the report.

Functions included in this file are `_addToList()` and `_submit()`.

`_addToList` is called when a user saves their report as a draft. This simply adds the text fields and dropdown menus to the list of drafts.

`_submit()` is called when a user selects to submit the report. This function maps the input data from the user and then saves it in the database.

1.1.2 Report_Draft_Page.dart

This file is shown when the user selects “Load Drafts”. This uses a scroll list to present all the draft reports that have been saved. This file has no functions other than to display the list. If the user selects a draft, they are taken to the `reportpad_details.dart` file.

1.1.3 *Reportpad_details.dart*

This file will load a view like the New_report.dart file, however, the input fields will be pre-filled with the data from the draft. The user will have the same functions in the New_report.dart file.

1.1.4 *Reportpad.dart*

This file contains the structure of what a report item consists of.

Functions used in this file are, add(), remove(), removeAll() and update(). The add() function adds a new report to the list. remove() removes an item from the list. removeAll() removes all item from the list. update() refreshes the view to show if new entries have been added or removed.

This also has some Hardcoded data for testing.

1.2 Backend

1.2.1 *Incident_report_submit.php*

Incident_report_submit.php uploads the information that is submitted to the database. This has error checks to make sure it is submitted correctly.

2 Start/End Shift

2.1 Application

Once on the security guard homepage clicking on the "SHIFT MANAGER" icon, the user is taken to the "Shift Clock" Page with all the businesses that the user is assigned to. In the code, this is under "businessModel.dart". After clicking on the business that the user wants to start the shift with, the user is taken to the "Details" page which the user can click on the dropdown list to choose where the user wants to start the shift with by clicking on to the "Start Shift" button. In the code, this is under "businessDetail.dart". Ending the shift is almost the same process as starting a shift, the only difference is to click on the "End Shift" button instead of the "Start Shift" button. The flow chart of this section can be seen in the figure below (figure 2).

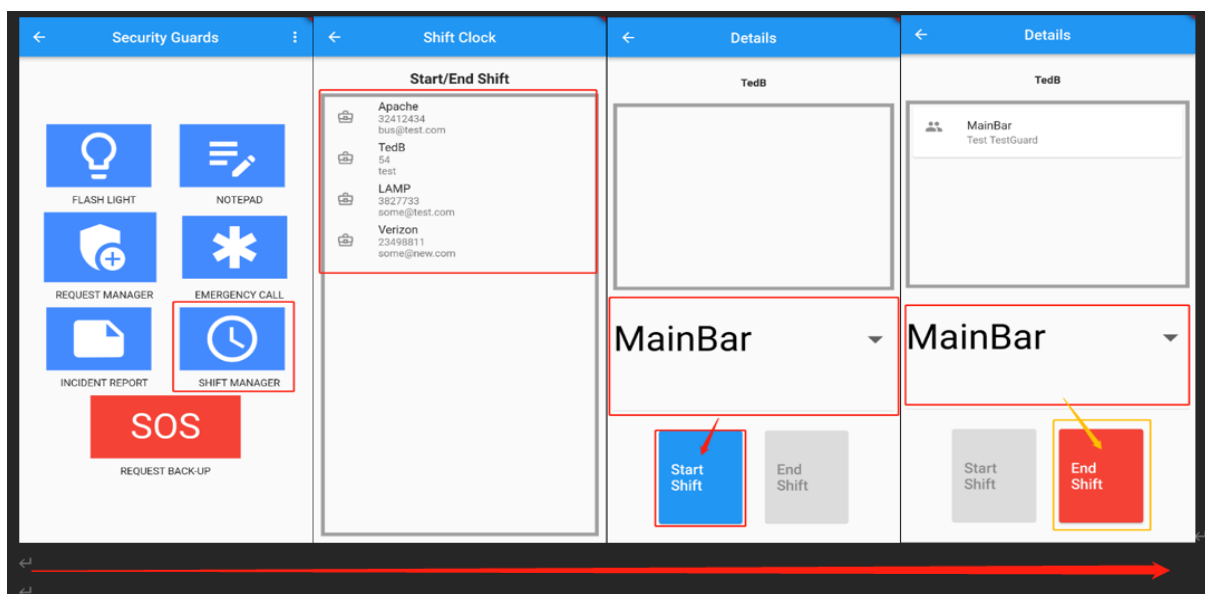


Figure 2

2.1.1 *newShiftClockHomepage.dart*

Whenever the user clicks the “SHIFT MANAGER” button, the application would direct to a page named ‘NewShiftClockPage’, the page structure is stored in “newShiftClockHomepage.dart”, it contains a container to wrap a list view, and the list view would display a list named ‘businesses’, and the quantity of list items is based on the length of the list.

2.1.2 *businessModel.dart*

This is a model class that defines the data structure of the business, also it uses a URL to get all the business details from ‘accountbusinessowner’ table and store them in a list named ‘business’.

2.1.3 *businessDetail.dart*

This is the page when the user clicks any business in “newShiftClockHomepage.dart”, the application would get the index from the list in ‘newShiftClockHomepage.dart’ and use it to define which business is clicked. First of all, the application would use the business to link to “connection/business_gard_show.php”; it would use the businessID to get related shifts from the ‘shiftdetails’ table, it will have AreaID and GuardKey from the result, then it would use the AreaID to get area details from ‘guardablearea’ table and use the GuardKey to get the guards details from the ‘accountsecurityguard’ table, finally, it would combine both data into an array and return a list, that will be used to display in the top container list.

Then it would link to another PHP which is “connection/getArea.php”, it would use the businessID to get all the area detail from “guardablearea” table.

Changing items in the dropdown menu will trigger a function name `changeEnd()`, which will define whether is the start or end button that needs to be able.

The Start button would return a function name ‘startShiftImpl’, it will define the guard already on shift and end the other shift first if the guard is on shift, or directly create a new shift.

The End button would return a function name “endShiftImpl”, it will just link to the “connection/updateShiftDetail.php”, and delete the selected data row from the “shiftdetails” table.

2.2 Backend

2.2.1 *startShift.php*

startShift.php, this file is to upload the information submitted to the database. This has error checks to make sure it is submitted correctly.

2.2.2 *shift_display.php*

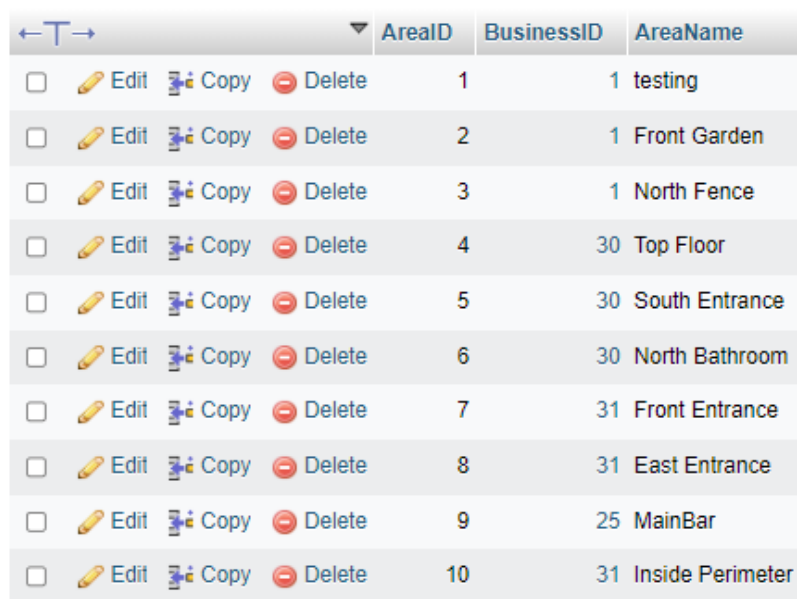
shift_display.php, this file is to read the information submitted to the database. This has error checks to make sure it is submitted correctly.

iii. updateShiftDetail.php

updateShiftDetail.php, this file is to delete the information submitted to the database. This has error checks to make sure it is submitted correctly.

Database

The following diagrams show the Area (figure 3) and shift detail (figure 4) of the existing data named shiftdetails.



	AreaID	BusinessID	AreaName
<input type="checkbox"/> Edit Copy Delete	1	1	testing
<input type="checkbox"/> Edit Copy Delete	2	1	Front Garden
<input type="checkbox"/> Edit Copy Delete	3	1	North Fence
<input type="checkbox"/> Edit Copy Delete	4	30	Top Floor
<input type="checkbox"/> Edit Copy Delete	5	30	South Entrance
<input type="checkbox"/> Edit Copy Delete	6	30	North Bathroom
<input type="checkbox"/> Edit Copy Delete	7	31	Front Entrance
<input type="checkbox"/> Edit Copy Delete	8	31	East Entrance
<input type="checkbox"/> Edit Copy Delete	9	25	MainBar
<input type="checkbox"/> Edit Copy Delete	10	31	Inside Perimeter

Figure 3



	ShiftID	GuardKey	BusinessID	AreaID	TimeStarted
<input type="checkbox"/> Edit Copy Delete	2	113	1	1	2022-09-26 12:28:52
<input type="checkbox"/> Edit Copy Delete	15	1	1	2	0000-00-00 00:00:00

Figure 4

3 Flashlight

3.1 Application

Once on the security guard homepage clicking on the “flashlight” icon, the flashlight that comes with the Android mobile can be turned on and off by the user, and all these functions of the flashlight are implemented in security_guards.dart. The flowchart for this section is shown in the following figure (Figure 5).

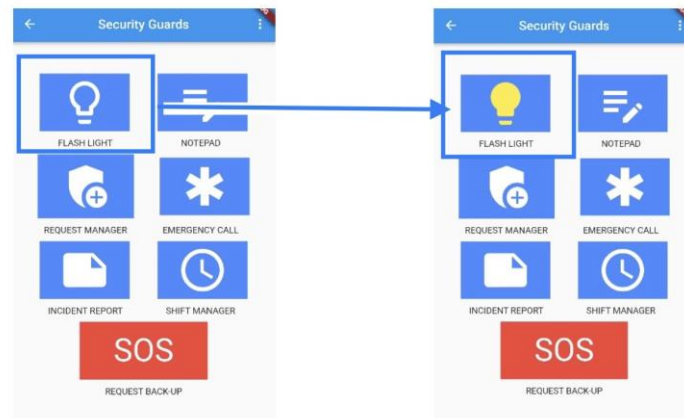


Figure 5

The flashlight-related functions contained in this file are `_CheckIfLightExist()`, `_turnOnLight()` and `_turnOffLight()`.

`_CheckIfLightExist()` is to check if the flashlight on the mobile device may be working properly because the flashlight as hardware can be affected by uncontrollable external factors, so if it can, the user can perform the next step.

`_turnOnLight()` is called when the user wants to turn on the flashlight.

`_turnOffLight()` is called when the user wants to turn off the flashlight.

3.2 Development tools

DevTools version 2.16.0

3.3 Adding a package dependency

`torch_light: ^0.4.0`

4 Notepad

4.1 Application

Once on the security guard homepage clicking on the “notepad” icon, the user will be taken to the notepad screen. In the code, this is called "notepad.dart".

Then if the user wants to edit the details of the notepad, the user will be taken to the notepad details screen, which is called "notepad_details.dart". The flowchart of this section is shown in the figure below (Figure 6).

After that, if the user wants to create a new notepad, the user will be taken to the screen where the new notepad is created, called "new_notepad.dart". The flowchart for this section is shown in the following figure (Figure 7).

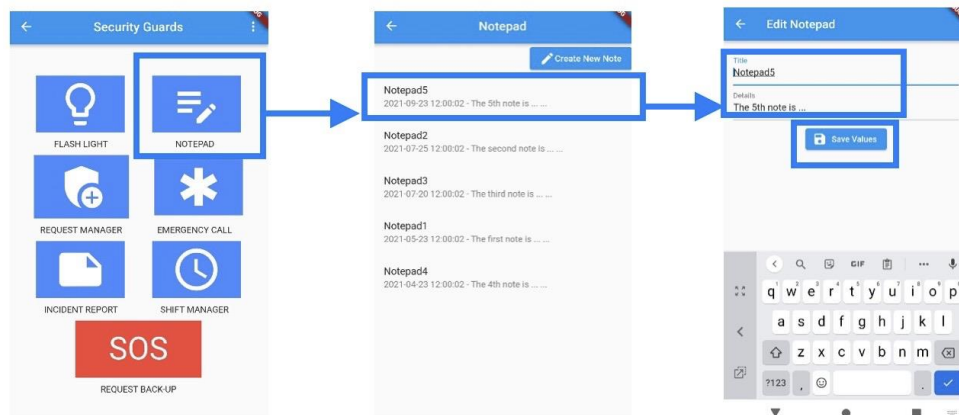


Figure 6

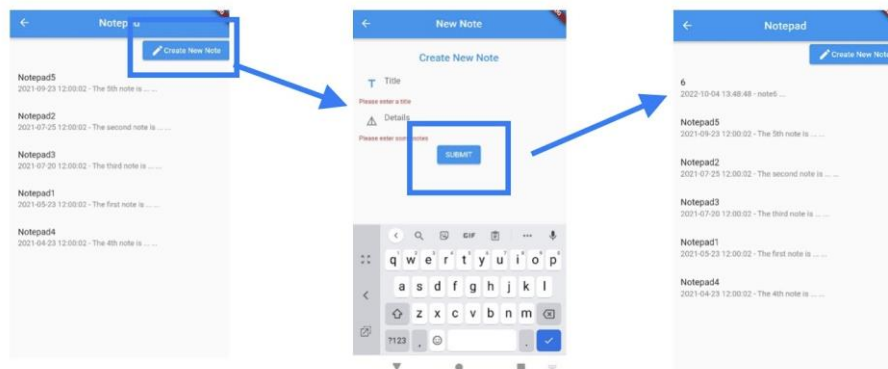


Figure 7

4.1.1 notepad.dart

This file is the file that builds the view for the user. This uses stateful widgets to provide the user with the option to fill in the report. It is composed of creating a new note and loading an existing note.

The functions used in this file are `add()`, `sort()`, `removeAll()` and `update()`. The `add()` function adds a new report to the list. `sort()` is to sort the items by time. `removeAll()` removes all items from the list. `update()` refreshes the view to show if new items have been added or removed.

4.1.2 new_note.dart

This file is the file that builds the view for the user and has the function to create a new note. The function `_buildtitleofNoteTextField()` is used in this file, `_bulddetailsofNoteTextField()`, `_addToList()` and `_onSubmit()`.

The `_buildtitleofNoteTextField()` is called to get the title of the note entered by the user. `_bulddetailsofNoteTextField()` is called to get the details of the note entered by the user. The

`_addToList()` call allows the user to save the note. The call to `_onSubmit()` is to get all the information entered by the user.

4.1.3 `notepad_details.dart`

This file is the function to view and edit the existing notes. The function used in this file is `update()`. `update()` is called to update a note after committing the changes.

4.1.4 `notepad_homepage.dart`

This file is the view that manages notepad. The function used in this file is `sort()`. `sort()` is called in order to sort the notes in chronological order, e.g. newer notes with newer dates are listed first.

4.2 Development tools

DevTools version 2.16.0

5 Emergency Call

5.1 Application

Once on the security guard homepage clicking on the “Emergency Call” icon, the user will be taken to the screen to make calls. It is important to note that the view for making calls will differ using different phones. In the code, this is called "emergencyphone.dart". The flowchart for this section is shown in the following figure (Figure 8).

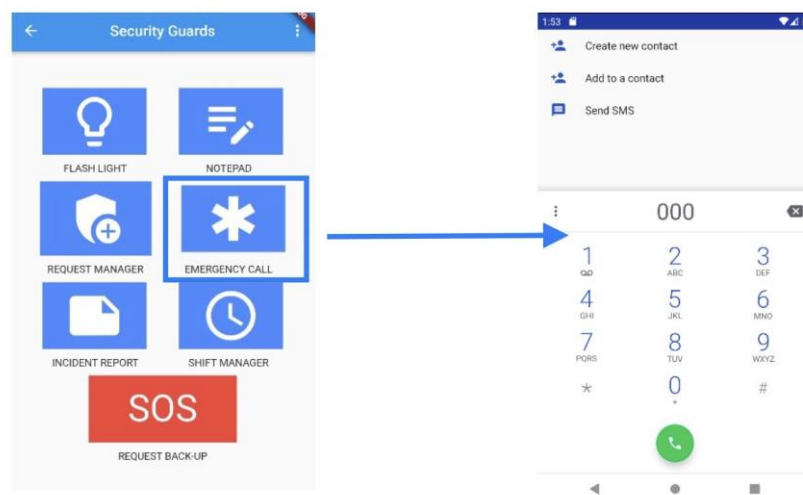


Figure 8

5.1.1 `emergencyphone.dart`

The file is for connecting to the phonebook and making outgoing calls. In this file, the string is set to "000" as the default number. Also, the flutter package called `url_launcher: ^6.1.2` is called to get the call function.

5.2 Development tools

DevTools version 2.16.0

5.3 Adding a package dependency

url_launcher: ^6.1.2

6 Request Manager

6.1 Application

Once the "Request Manager" icon is clicked on the Security Guard home page, the user will be taken to the Request Manager screen. In the code, this is referred to as "backup_list.dart".

Then, if the user wants to view the details of a particular entry, the user will be taken to the Notepad details screen, which is referred to as "backup_homepage.dart". The user can choose to respond (Figure 9) or reject it (Figure 10).

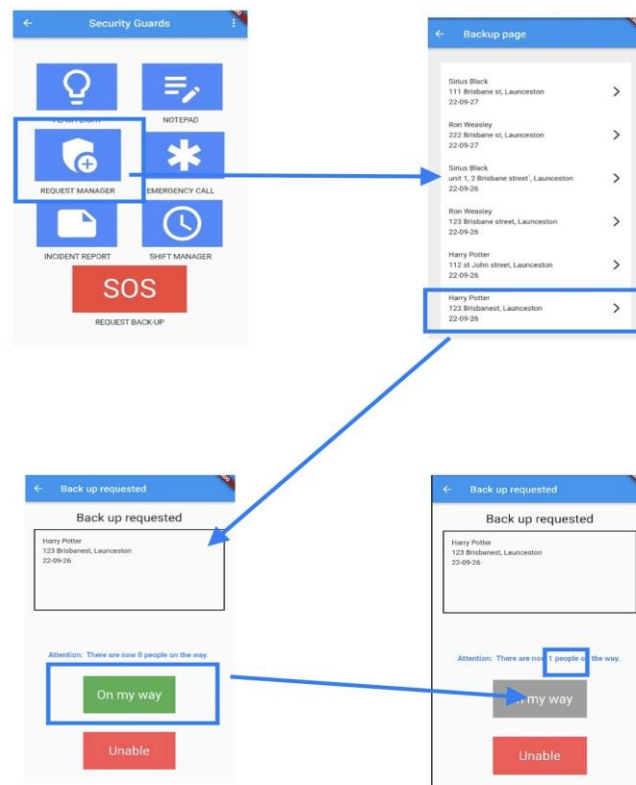


Figure 9

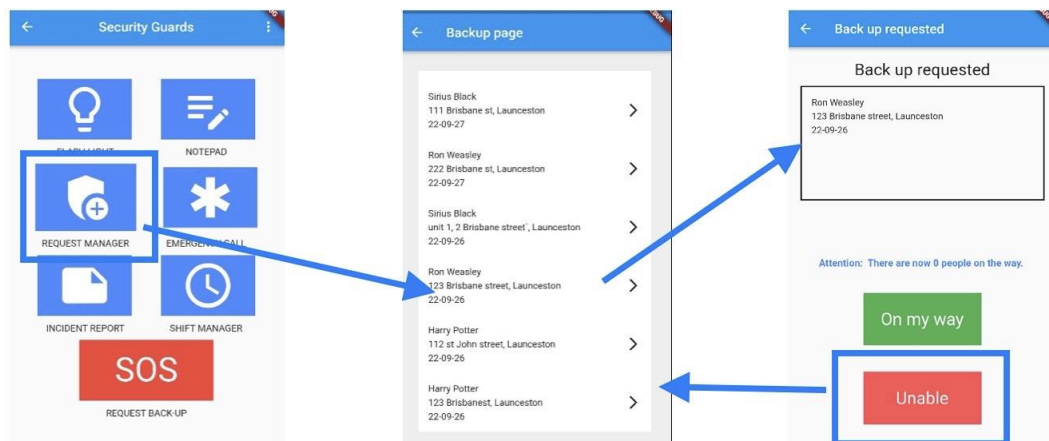


Figure 10

6.1.1 backup.dart

This file is the file that builds the view for the user. This uses stateful widgets to provide the user with the option to fill in the report. The functions used in this file are add(), removeAll() and update(). The add() function adds a new report to the list. removeAll() removes all items from the list. update() refreshes the view to show if new items have been added or removed.

6.1.2 backup_list.dart

This file is the file that builds the backup list for the user. The functions used in this file are add() and sort(). The add() function adds a new report to the list. The sort() is called to put each backup message in chronological order.

6.1.3 backup_homepage.dart

The file contains the setState() function setState(), which is called to prevent multiple presses when the user clicks the "on my way" button.

6.2 Development tools

DevTools version 2.16.0

6.3 Database

The following diagrams show the table (Figure 11), table structure (Figure 12) and unique key (Figure 13) of the existing data named backupdetails respectively.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	Firstname	varchar(36)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/> 2	Lastname	varchar(36)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 3	Location	varchar(72)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 4	Time	varchar(36)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 5	test	varchar(200)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 6	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More

☐ Check all With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#) [Spatial](#) [Fulltext](#)

Figure 11




























←T→		▼	Firstname	Lastname	Location	Time	test	id	
<input type="checkbox"/>				Harry	Potter	123 Brisbanest, Launceston	22-09-26	NULL	1
<input type="checkbox"/>				Harry	Potter	112 st John street, Launceston	22-09-26	NULL	2
<input type="checkbox"/>				Ron	Weasley	123 Brisbane street, Launceston	22-09-26	NULL	3
<input type="checkbox"/>				Sirius	Black	unit 1, 2 Brisbane street, Launceston	22-09-26	NULL	4
<input type="checkbox"/>				Ron	Weasley	222 Brisbane st, Launceston	22-09-27	NULL	12
<input type="checkbox"/>				Sirius	Black	111 Brisbane st, Launceston	22-09-27	NULL	13
<input type="checkbox"/>				Sirius	Black	1 Brisbane st, Launceston	22-09-27	NULL	14
<input type="checkbox"/>				Ron	Weasley	2 Brisbane st, Launceston	22-09-27	NULL	15
<input type="checkbox"/>					NULL	currunt location	22-09-27	NULL	16

Figure 12

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	id	39	A	No	

Figure 13

6.4 Backend

6.4.1 *backup_list.php*

backup_list.php , this file is to get information about the existing database. This has error checks to make sure it is submitted correctly.

7 Request Backup

7.1 Application

Once the " Request Backup " icon is clicked on the Security Guard home page, the user will be taken to the Request Backup screen. In the code, this is referred to as "*backup_successful.dart*".Alternatively, after clicking the "SOS" button, you can select "yes" in the pop-up window to confirm the request (Figure 14). Alternatively, click "no" to prevent accidental touch (Figure 15). When the request is sent successfully, a pop-up will appear to alert other users that a new request has been sent (Figure 16).

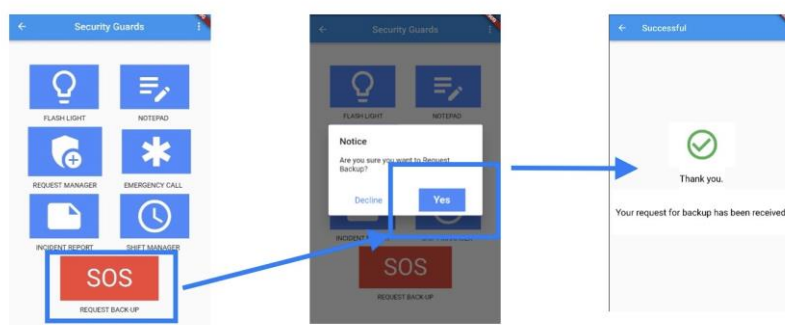


Figure 14

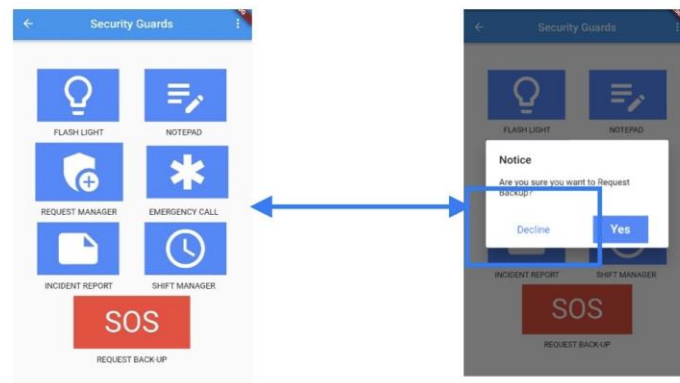


Figure 15

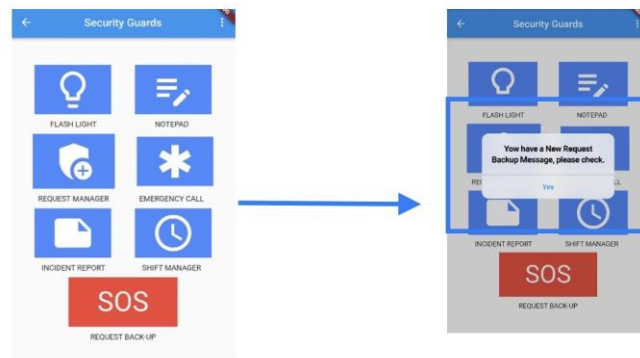


Figure 16

7.1.1 `backup_successful.dart`

This page currently does not call any function, because this page is just direct visual feedback to the user. The user has successfully jumped to this page, which means the user has sent the request successfully.

7.2 Development tools

DevTools version 2.16.0

7.3 Database

The following diagrams show the table (Figure 17), table structure (Figure 18) and unique key (Figure 15) of the existing data named backupdetails respectively.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	Firstname	varchar(36)	utf8mb4_general_ci		No	None		Change Drop More
<input type="checkbox"/>	2	Lastname	varchar(36)	utf8mb4_general_ci		Yes	NULL		Change Drop More
<input type="checkbox"/>	3	Location	varchar(72)	utf8mb4_general_ci		Yes	NULL		Change Drop More
<input type="checkbox"/>	4	Time	varchar(36)	utf8mb4_general_ci		Yes	NULL		Change Drop More
<input type="checkbox"/>	5	test	varchar(200)	utf8mb4_general_ci		Yes	NULL		Change Drop More
<input type="checkbox"/>	6	id	int(11)			No	None	AUTO_INCREMENT	Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

Figure 17

		▼ Firstname	Lastname	Location	Time	test	id
<input type="checkbox"/>	Edit Copy Delete	Harry	Potter	123 Brisbanest, Launceston	22-09-26	NULL	1
<input type="checkbox"/>	Edit Copy Delete	Harry	Potter	112 st John street, Launceston	22-09-26	NULL	2
<input type="checkbox"/>	Edit Copy Delete	Ron	Weasley	123 Brisbane street, Launceston	22-09-26	NULL	3
<input type="checkbox"/>	Edit Copy Delete	Sirius	Black	unit 1, 2 Brisbane street, Launceston	22-09-26	NULL	4
<input type="checkbox"/>	Edit Copy Delete	Ron	Weasley	222 Brisbane st, Launceston	22-09-27	NULL	12
<input type="checkbox"/>	Edit Copy Delete	Sirius	Black	111 Brisbane st, Launceston	22-09-27	NULL	13
<input type="checkbox"/>	Edit Copy Delete	Sirius	Black	1 Brisbane st, Launceston	22-09-27	NULL	14
<input type="checkbox"/>	Edit Copy Delete	Ron	Weasley	2 Brisbane st, Launceston	22-09-27	NULL	15
<input type="checkbox"/>	Edit Copy Delete		NULL	currunt location	22-09-27	NULL	16

Figure 14

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	id	39	A	No	

Figure 18

7.4 Backend

7.4.1 backup_list.php

backup.php , this file is to upload the information submitted to the database. This has error checks to make sure it is submitted correctly.

8 SecurityGuardDB.dart

This file is called when the user tries to log into the security guard side of the application. It checks with the database for the correct email and password.

9 Information Pool

9.1 Application

This section is found by logging into the business owner side of the application and selecting the event log.

[*9.1.1 create_event_view.dart*](#)

This file contains the widgets for building the view for the user. It includes input fields and a drop-down menu. This has one function called `_submitToDatabase()`. This function takes the information for the input fields and saves it to the database. It also contains some error checks.

[*9.1.2 event_details_view.dart*](#)

This file contains widgets to display selected event from the information pool. Once an event is selected it will display all the details

[*9.1.3 event_information_pool.dart*](#)

This file contains the structure of what an event item consists of.

Functions used in this file are, `add()` and `update()`. The `add()` function adds a new event to the list. `update()` refreshes the view to show if new entries have been added or removed.

[*9.1.4 information_homepage.dart*](#)

This file is responsible for building the list view for events. It displays all the information for each event in the database and displays them in order of date.

9.2 Backend

[*9.2.1 event_details.php*](#)

This file is responsible for accessing the associated database table and returning an array of events. This file accesses two tables, `severitylevel` to get the corresponding severity colour, and `eventdetails`, to get the event details.

[*9.2.2 create_new_event.php*](#)

This file is responsible for creating a new event by inserting it into the `eventdetails` database table.

10 Model

10.1 Application

This section holds the structure for business owners, security guard accounts and API.

[*10.1.1 business_owner.dart*](#)

This file contains the structure of what a business owner item consists of.

Functions used in this file are `getLoggedInBusinessValue()`. This function retrieves the value for the logged in business owner.

[*10.1.2 securityGuard.dart*](#)

This file contains the structure of what a security guard object consists of.

Functions used in this file are `getGuardValue()`. This function retrieves the value for the logged in security guard.

10.2 Backend

11 View

11.1 Application

This contains the code for login and homepage displays.

[*11.1.1 business_login.dart*](#)

This file contains widgets for the business owner login. This takes the input for the user for email and user and starts the check to make sure it is correct. It has the `_login()` function. This checks the database for the correct login.

[*11.1.2 business_owners.dart*](#)

This file contains the business owner homepage display. It contains widgets for displaying all the options available.

[*11.1.3 home_page.dart*](#)

This is the initial landing page. It has widgets for displaying the options for business owner login and security guard login.

[*11.1.4 security_guard.dart*](#)

This file contains the security guard homepage display. It contains widgets for displaying all of the options available.

[*11.1.5 security_login.dart*](#)

This file contains widgets for the security guard login. This takes the input for the user for email and user and starts the check to ensure it is correct. It has the `_login()` function. This checks the database for the correct login.

[*11.1.6 signup_business.dart*](#)

This file contains the display and functions for the business owner sign-up page. It has widgets for displaying the inputs that the user needs to fill out. It has a `_submit()` function which will take the input and save it to the database.

[*11.1.7 signup_security.dart*](#)

This file contains the display and functions for the security guard sign-up page. It has widgets for displaying the inputs that the user needs to fill out. It has a `_submit()` function which will take the input and save it to the database.

11.2 Backend

[*11.2.1 business_owner_login.php*](#)

This is an API file that is responsible for checking that the provided business owner login details exist in the database table and are correct. On a successful request, it will provide a response to `business_login.dart`, otherwise it will return an error.

[*11.2.2 security_guard_login.php*](#)

This is an API file that is responsible for checking that the provided security guard login details exist in the database table and are correct. On a successful request, it will provide a response to `security_login.dart`, otherwise it will return an error.

[*11.2.3 business_owner_create.php*](#)

This is an API file that is responsible for adding the provided details into the corresponding database table and creating the new business owner account.

[*11.2.4 security_guard_create.php*](#)

This is an API file that is responsible for adding the provided details into the corresponding database table and creating the new security guard account.

12 Global

12.1 Application

This section contains files used globally and the main.

12.1.1 *global_ip.dart*

This file is used to set the ip address throughout the project for connecting to the local database.

12.1.2 *main.dart*

This is the main file. It is the top-level function of the Dart. This is what first runs and generates the rest of the project.

13 Website

13.1 *secfirm*

This folder contains all associated security firm dashboard files. This includes CSS, JS, and PHP files.

13.1.1 *css*

13.1.1.1 *homepage.css*

This CSS file contains the styling for the homepage.

13.1.1.2 *style.css*

This CSS file contains the style for the entire website.

13.1.2 *js*

13.1.2.1 *index.js*

This file manages logout.

13.1.2.2 *login.js*

This file manages the security firm account login process.

13.1.2.3 *register.js*

This file manages the security firm account registration process.

13.1.3 *business_info.php*

This file populates the business information modal in *security_firm_dashboad.php*. It uses the table element id to grab the corresponding row information from the *accountbusinessowner* table. It also grabs the associated list of guardable locations from *guardablearea* table.

13.1.4 *db_conn.php*

This file contains the database connection information.

13.1.5 *generate_key.php*

This is an API file that inserts the newly generated guard key into the *accountsecurityfirm* table.

13.1.6 *incident_report_info.php*

This file populates incident report modal in *security_firm_dashboard.php*. It receives data from *incidentreport*, *accountsecurityguard*, and *accountbusinessowner* tables.

13.1.7 *index.php*

This file contains the landing page of the website.

[*13.1.8 key_info.php*](#)

This file populates the modal in security_firm_dashboad.php and displays the assigned guard.

[*13.1.9 login_process.php*](#)

This file creates a session on successful login.

[*13.1.10 login.php*](#)

This file contains the security firm account login structure.

[*13.1.11 logout.php*](#)

This file holds the logout functionality for the security firm user.

[*13.1.12 register_success.php*](#)

This file contains the landing page when a new security firm account is registered.

[*13.1.13 register.php*](#)

This file contains a form that enables new security firm accounts to be made. It passes the form details to 13.1.2.3 register.js.

[*13.1.14 security_firm_dashboard.php*](#)

This file contains the security firm dashboard. It utilises many of files to populate modals. It contains many tables, which also contain API calls to the database.

[*13.1.15 security_info.php*](#)

This file is responsible for showing the details for each security guard.

Once the post is received, it will check the database for same GuardKey

[**14 Development Tools Used**](#)

Flutter: Version 3.0.0, for all Mobile Application Development.

Android Studio: Version 2021.1, to deploy the Android SDK android-x86 Version 8.1.0.

Visual Studio Code: Version 1.71.2, for all development.

XAMPP: Version v3.3.0, for local hosting server + MySQL database.

PHP: Version 8.1.5, for all API.

[**15 Information Regarding System Modification**](#)

Since a MVC design was not closely adhered to, it will be difficult to modify the system without closely understanding each file.

The next step is connecting it to a live server. To do this, everywhere an API call is made, the corresponding URL provided will need to be changed to the new API location on the server. The global IP address will no longer be necessary as the domain should already be defined.