



ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT124-3-2-MAE

MOBILE APP ENGINEERING

HAND OUT DATE: 1 JANUARY 2022

HAND IN DATE: 4 MARCH 2022

WEIGHTAGE: 50%

Name : SIA DE LONG

TP Number : TP060810

Intake : APD2F2106CS(IS)

Table of Contents

1.0 Introduction.....	4
1.1 Problem Statement	4
1.2 Application Requirements.....	5
2.0 Application Design	6
2.1 Use Case Diagram.....	6
2.2 Entity Relationship Diagram.....	7
2.3 Data Architecture	8
2.4 Application Mock-ups.....	9
3.0 Implementation	25
3.1 Technique & Framework	25
3.2 Data Passing	25
3.3 Tab Bar Adapter.....	26
3.4 Recycler View	27
3.5 Live Data & View Model.....	28
4.0 Application Testing.....	32
4.1 Espresso Test.....	32
4.2 Junit test.....	34
5.0 User Manual.....	35
6.0 Conclusion	46
References.....	47
Appendix.....	47

List of Figures

Figure 1: Use Case Diagram	6
Figure 2: Entity Relationship Diagram.....	7
Figure 3: Data Architecture Diagram	8
Figure 4: Mock-up of Start-up and Sign in Screen	9
Figure 5: Mock-up of Forgot Password Screen	10
Figure 6: Mock-up of Sign-Up Screen	11
Figure 7: Mock-up of Scan to Borrow Screen	12
Figure 8: Mock-up of Manual Borrow Screen.....	13
Figure 9: Mock-up of E-Resource Screen.....	14
Figure 10: Mock-up of Select Reserve Time Screen.....	15
Figure 11: Mock-up of Select Room Screen.....	16
Figure 12: Mock-up of E-Wallet Screen	17
Figure 13: Mock-up of Profile Screen	18
Figure 14: Mock-up of Personal Information Screen.....	19
Figure 15: Mock-up of Notifications Screen	20
Figure 16: Mock-up of Borrowed Book Screen.....	21
Figure 17: Mock-up of Reserved Room Screen.....	22
Figure 18: Mock-up of Settings Screen	23
Figure 19: Mock-up of Admin Function Screen	24
Figure 20: Data Passing to Activity.....	25
Figure 21: Data Passing to Fragment	26
Figure 22: Tab Bar Adapter	27
Figure 23: Tab Adapter Usage.....	27
Figure 24: Recycler View Adapter	28
Figure 25: Recycler View Usage	28
Figure 26: Book Table.....	28
Figure 27: Book DAO	29
Figure 28: Book Database	29
Figure 29: Book Repository	30
Figure 30: Book View Model	30
Figure 31: Book View Model Factory	31
Figure 32: Live Data Observation.....	31
Figure 33: Login Test	33
Figure 34: Login Test Result	33
Figure 35: Login.....	35
Figure 36: Sign-up	36
Figure 37: Forgot Password	37
Figure 38: Scan to Borrow.....	38
Figure 39: Manual Borrow	39
Figure 40: E-Resource	40
Figure 41: Reserve Date	41
Figure 42: Select Room	42
Figure 43: E-Wallet.....	43
Figure 44: Profile.....	44
Figure 45: Admin Function.....	45

1.0 Introduction

1.1 Problem Statement

Library is a place that has a collection of physical source material including books, documents, magazines, newspaper as well as tools and artworks which are all available for people to borrow for a limited time period (Cambridge Dictionary, 2022). The most significant benefit is that most of the time it can be done without any payment while the source material is very reliable and accurate as a reference for study.

Despite that, there are a lot of report showing that library services are being more and more declining from the society. Even the factor of the pandemic situation right now, the interest of people looking for physical source material is also reducing every year. As example taken from Freckle Report 2021, United State is having 31% decline to the library services, 22% for Australia over 10 years and surprisingly 70% of decline in the United Kingdom since the year of 2000. The report is not only focusing on physical source material but also digital one and found out that the people of preferring digital source material is increasing every year (Andrew, 2021).

As we all know that, digital source material is one of the results from the growth of technology. In the meantime, the growth rate of technology is too fast and unpredictable, from statistic of technology showed that there is 59% which is over half of the system adopt internet while it is also surprising that 62% of the world total population own a mobile phone on January of 2021, everything can be done easily while the device is portable easily (Jacquelyn, 2022). There is many more other technology growth statistics, so it is almost expectable that one day physical source material might not exist anymore.

1.2 Application Requirements

To overcome the problem, the requirements of the application were to be determined to be critical for the problem statement, hence there are more than ten ideas generated based on the affinity analysis study to be implemented on the application. However, only a few chosen requirements will be implemented on this current phase of the application which are shown as below.

- Account Authentication
- Register account
- Borrow book via QR code scanner or manual borrow
- Download e-source materials
- Reserve a library room
- Make compensation payment by building a e-wallet system
- Personalise settings

2.0 Application Design

2.1 Use Case Diagram

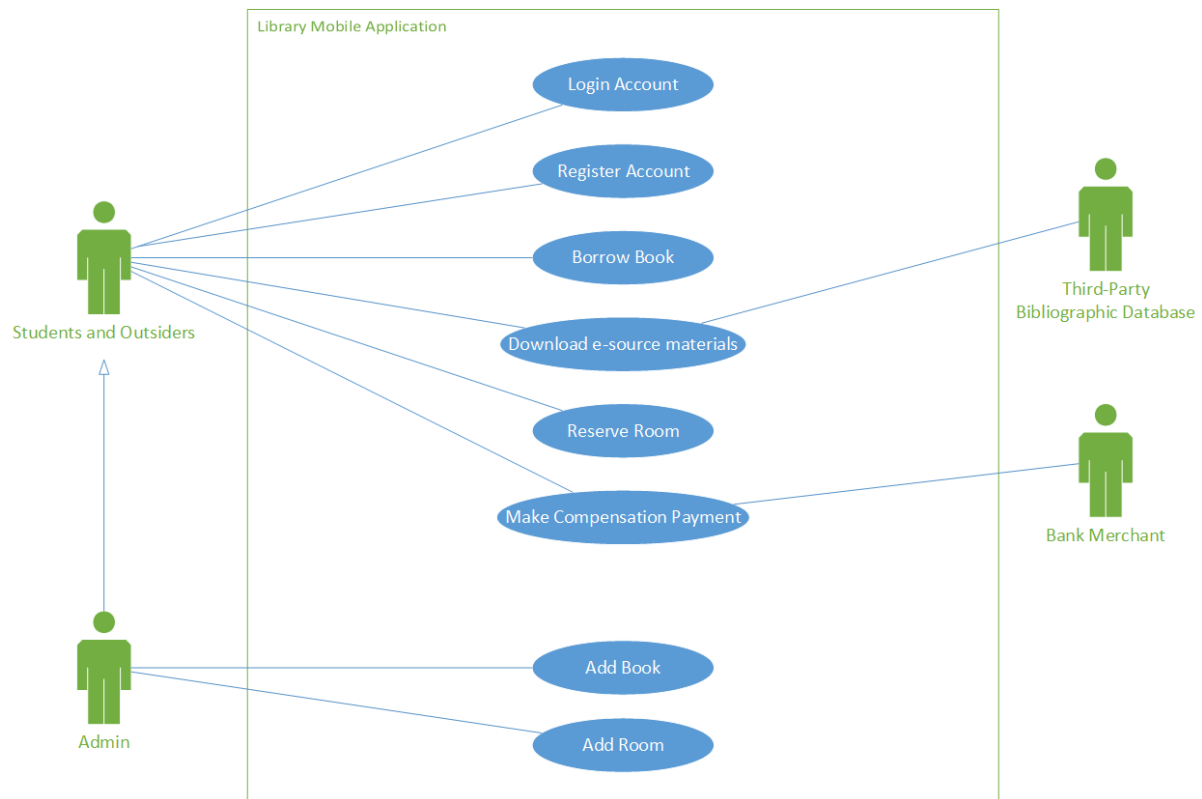


Figure 1: Use Case Diagram

Figure above shows the use case of the application which involved four stakeholders including students and outsiders, admin, third-party bibliographic database and bank merchant. Students and outsiders will be the general user of the application while admin will have both the usual functionalities and admin only functionalities. Some of the use case require secondary stakeholders such as the e-resource and e-wallet system to complete the use case process.

2.2 Entity Relationship Diagram

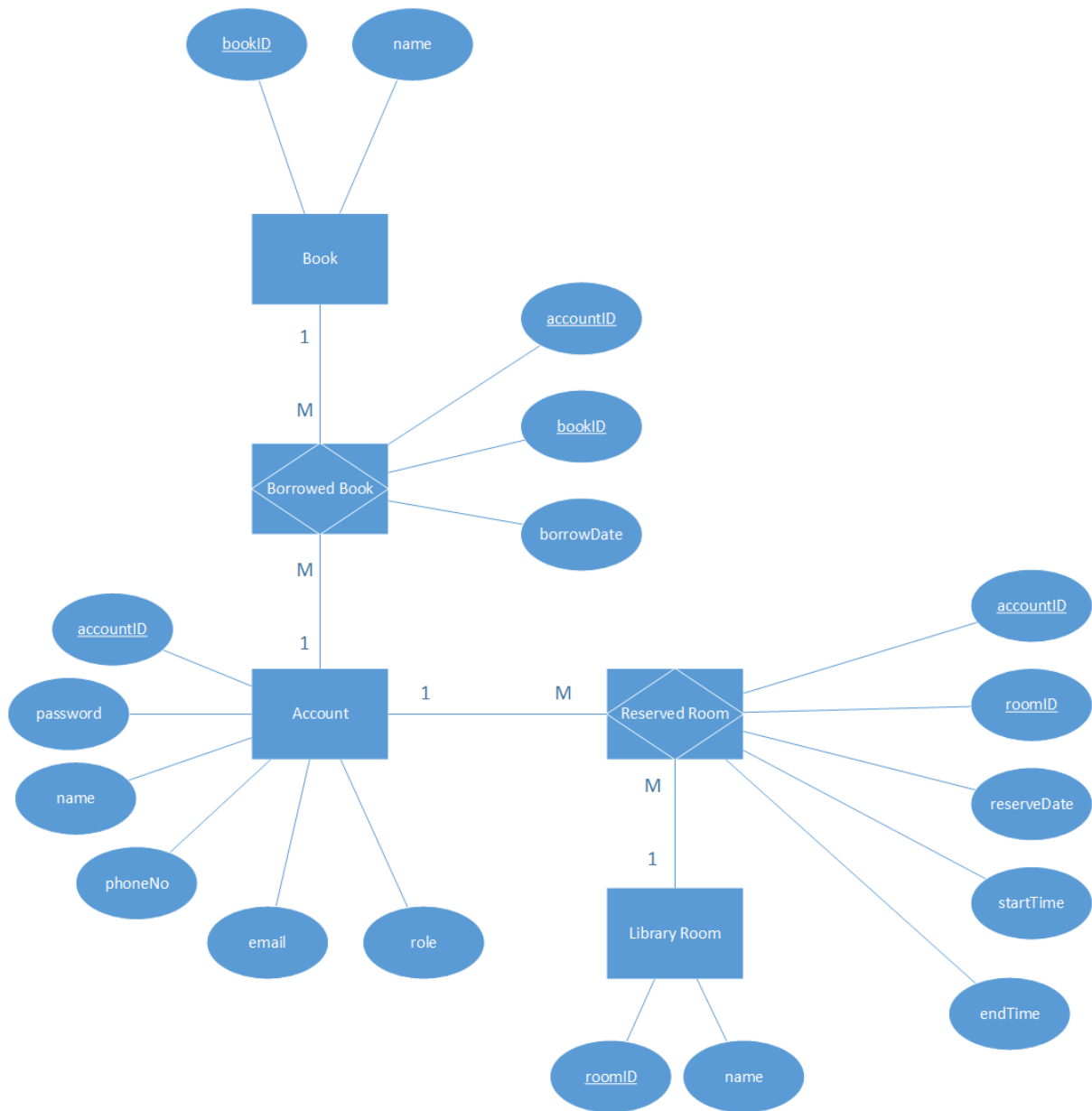


Figure 2: Entity Relationship Diagram

Figure above shows the entity relationship diagram for the application database, there will be a total five table for the application to store the user activities data which are Account, Book, Library Room, Borrowed Book and Reserved Room. Borrowed Book and Reserved Room is an associative entity which created by the many-to-many relationship since an account can borrows many books and reserves many rooms vice versa. Hence, two of the table will have the id from both related tables as foreign and primary while date will also be the primary key to store data.

2.3 Data Architecture

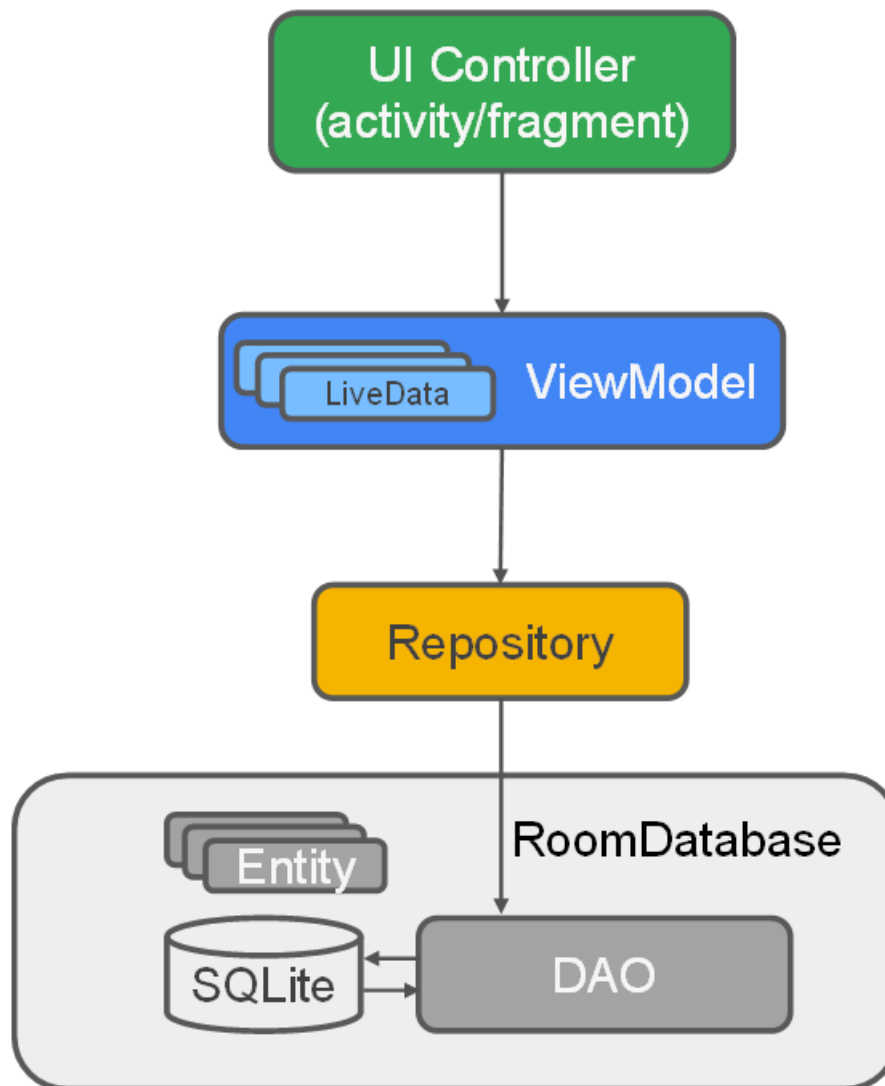


Figure 3: Data Architecture Diagram

Figure above shows the data architecture layers of the mobile application, users will only interact with the UI controller which will display data and forward on UI events. The data it used is from a ViewModel holding a LiveData needed for the UI where LiveData will notify the data changes everytime using observation function. Repository is act as a clean API for UI to communicate with the database. Since, the mobile application will implement RoomDatabase, it will manage the local data storage using SQLite data source by having entity act as tables and DAO act as queries for that table.

2.4 Application Mock-ups

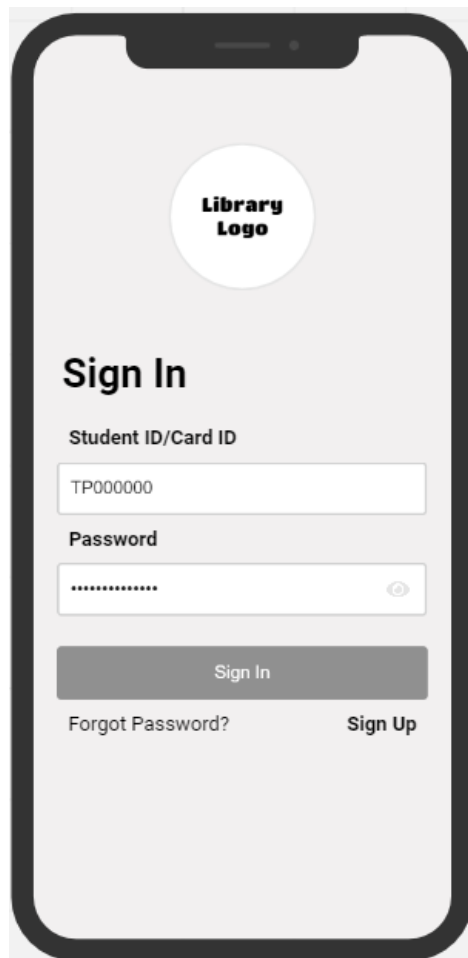


Figure 4: Mock-up of Start-up and Sign in Screen

This is the start-up screen of the application which is used for the account login function, user can input their id and password to the slot and pressing sign in button to check the account existence. If the account information is correct then it will move to the homepage, else a toast message will pop out, while empty slot validation will be done before accessing to the database to minimise unnecessary memory usage. User can also go to forgot password fragment or sign-up activity in this page by clicking the textView under the button.

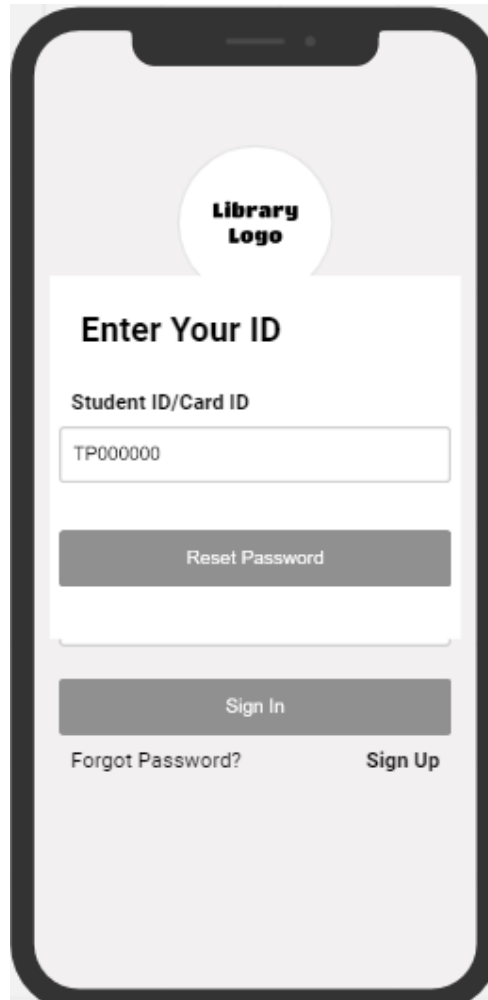


Figure 5: Mock-up of Forgot Password Screen

This is the fragment that pop out after user click on the forgot password text on the login page. It will request the user to enter the account ID to retrieve the email information of that account. After the user click on reset password button, an email will be sent to the account email address which can then use to solve the password issue, while empty slot validation will be done before accessing to the database to minimise unnecessary memory usage.

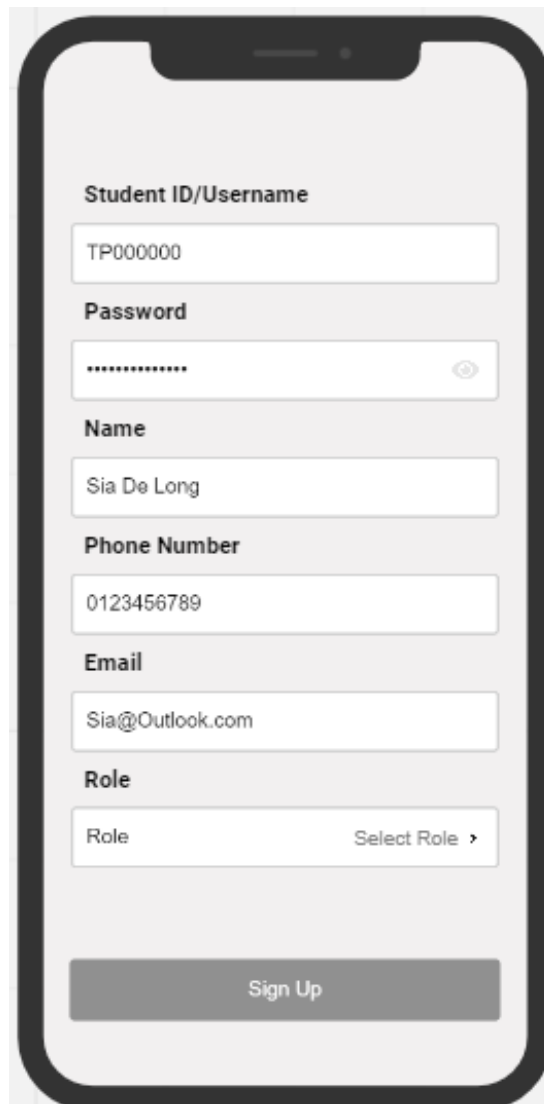
A mock-up of a mobile application sign-up screen. The screen is light gray with a black border. It contains several input fields with labels above them: 'Student ID/Username' with the value 'TP000000', 'Password' with a masked value '*****' and an eye icon, 'Name' with the value 'Sia De Long', 'Phone Number' with the value '0123456789', 'Email' with the value 'Sia@Outlook.com', and 'Role' with a dropdown menu showing 'Role' and 'Select Role >'. At the bottom is a large gray button labeled 'Sign Up'.

Figure 6: Mock-up of Sign-Up Screen

This is the sign-up activity page which will be led to after user click sign up text on the login page. It will request the user to give all the required account information, while the role is only students or outsiders available to select. After the user complete filling and press the sign up button, the account will then be added to the database.

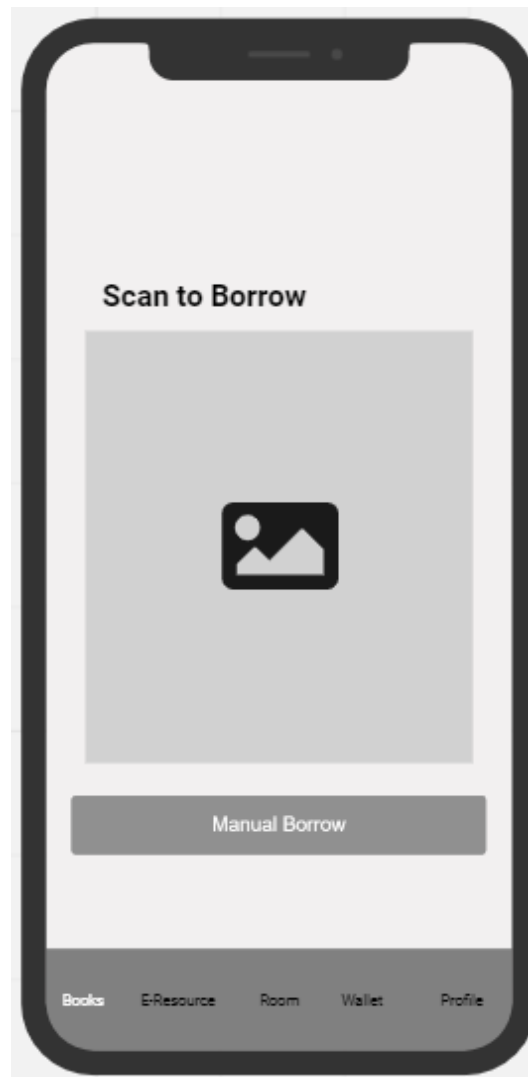


Figure 7: Mock-up of Scan to Borrow Screen

This is the homepage activity screen which will be displayed after a successful login, the first tab will be the default to show the fragment of scan to borrow. It will be using the camera of the device to detect and scan the book QR code, then the QR will be used to a decoder to proceed the borrow book process. After the borrow process complete, a message should pop out to the user.

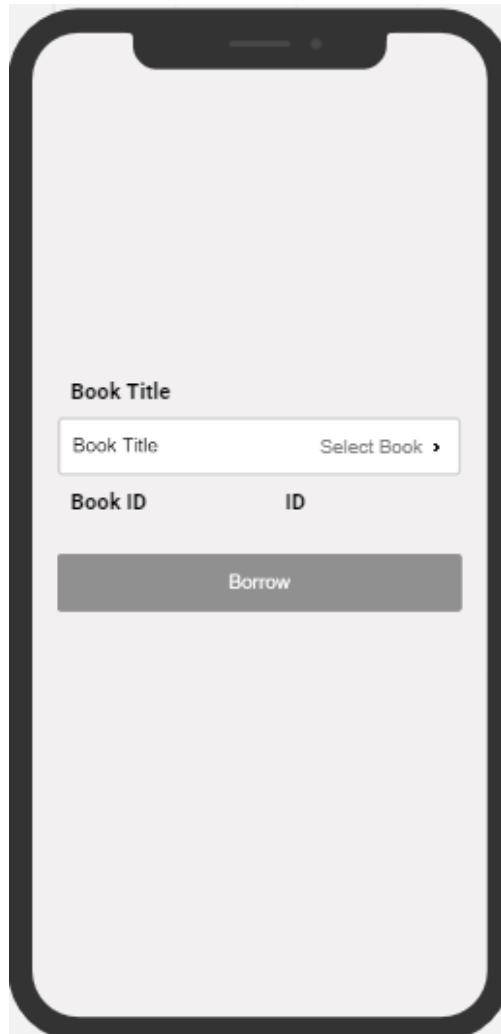


Figure 8: Mock-up of Manual Borrow Screen

This is the manual borrow activity which is from the button of scan to borrow screen. This page allows users to select an available book title from the database and display the ID for it, only book that haven't be borrow will be displayed on the selection. After selecting, the user can click the borrow button to complete the borrow process.

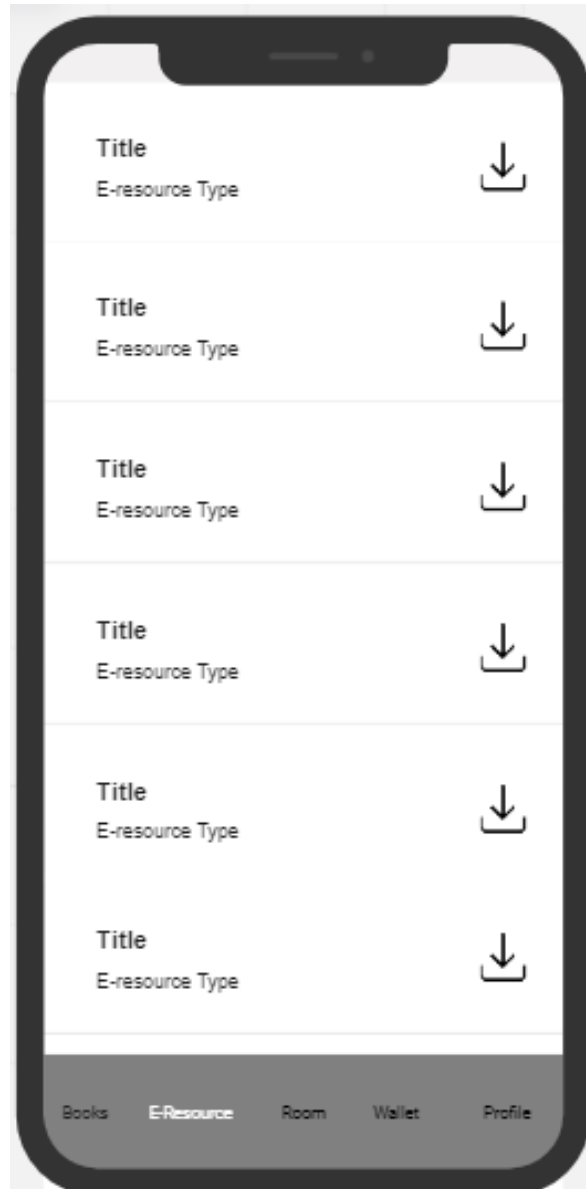


Figure 9: Mock-up of E-Resource Screen

This is the e-resource tab from the homepage screen, all the e-resource connected from the third-party bibliographic database will be displayed as a list to be scroll down. User can freely download the e-resource they want by pressing the download button beside it, then a pdf file will be downloaded to the device storage.

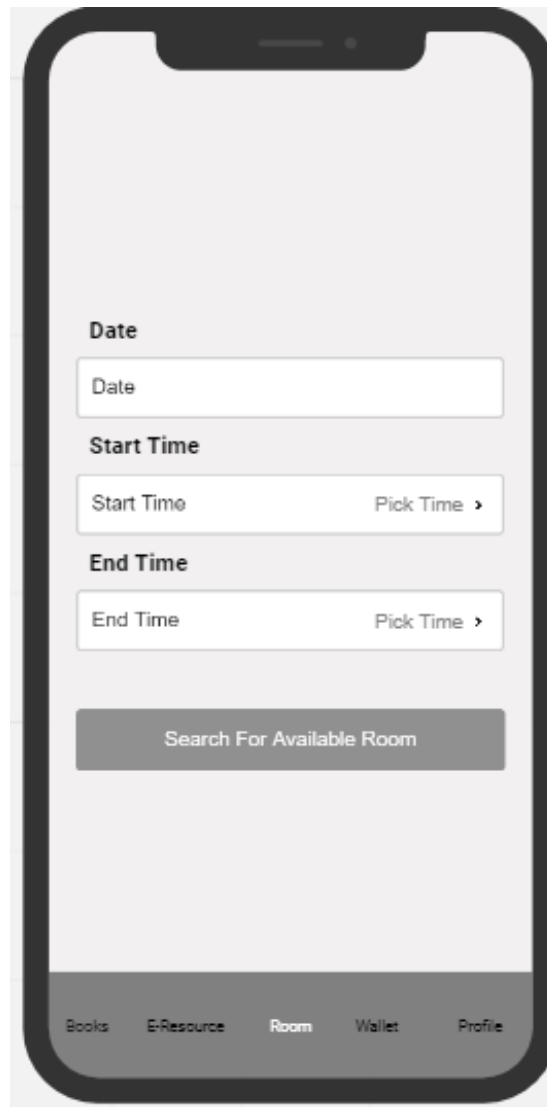


Figure 10: Mock-up of Select Reserve Time Screen

This is the room tab from the homepage screen, it will request the user to give the date, start time and end time for the room reservation in this page. User can press for search for available room button after complete filling the reservation time to select available room to reserve.

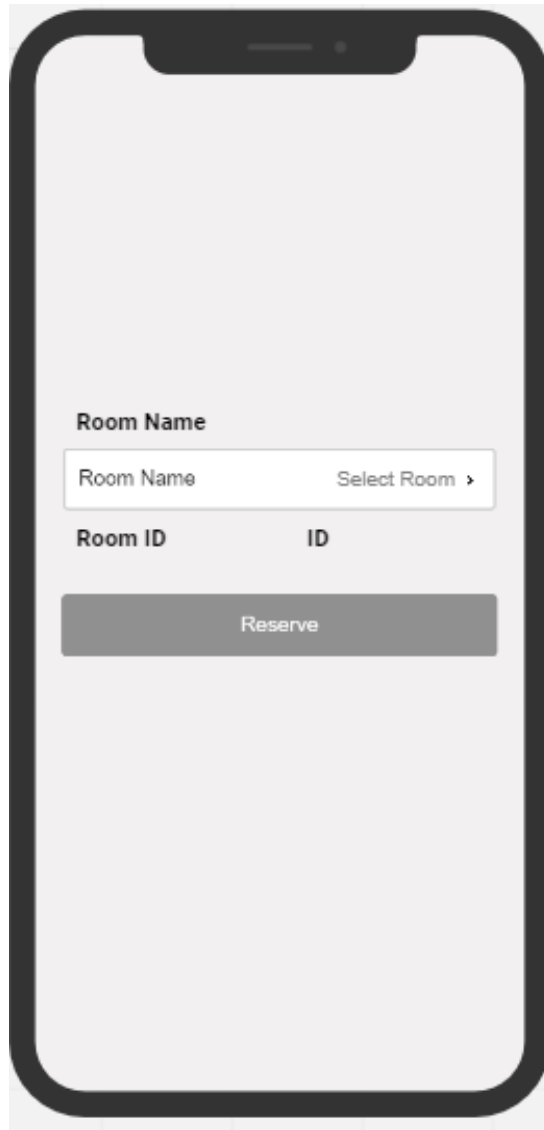


Figure 11: Mock-up of Select Room Screen

This is the select room activity from the room tab button. User can select the room name that they want to reserve and it will display the room id, only room that available at that time range will be displayed on the list to be selected. After select a room, user can press on the reserve button to add a reserve record to the database.

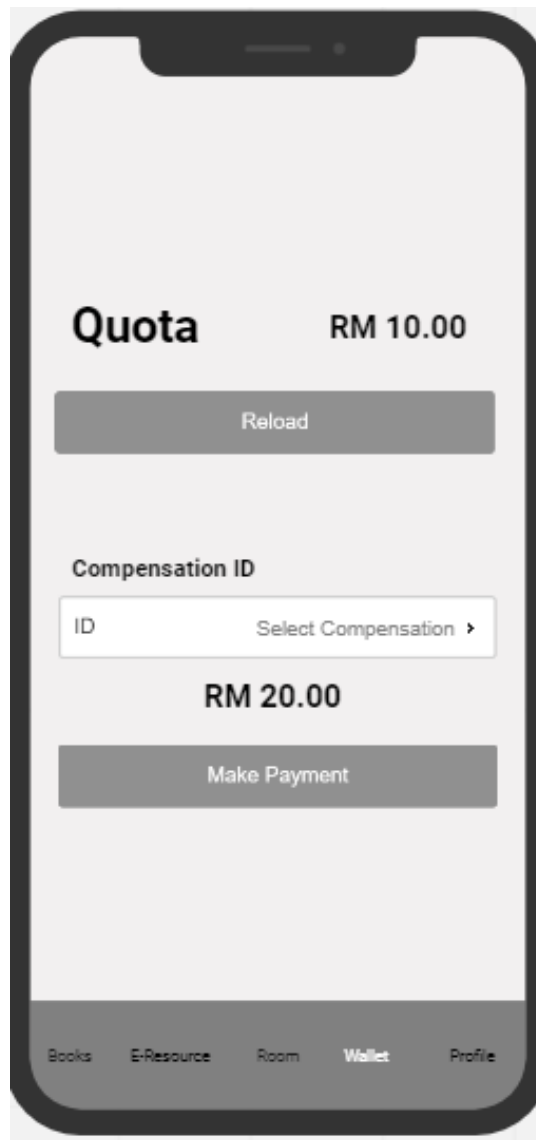


Figure 12: Mock-up of E-Wallet Screen

This is the wallet tab from the homepage screen, it will show the balance of the account wallet and able to let user to reload their balance by connecting to their bank merchant. Besides that, this page is also allowed user to make payment to their compensation by selecting the compensation ID and use the balance of the account.

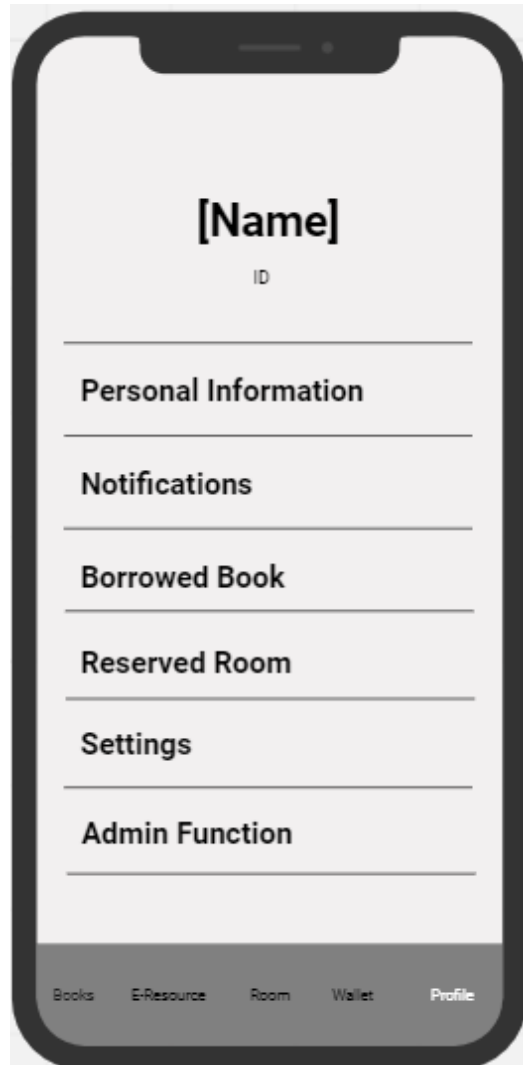


Figure 13: Mock-up of Profile Screen

This is the profile tab from the homepage screen, it will show the name and ID for the user and a few of the buttons which related to this profile. The admin function button will only be displayed if the current account is having admin role.

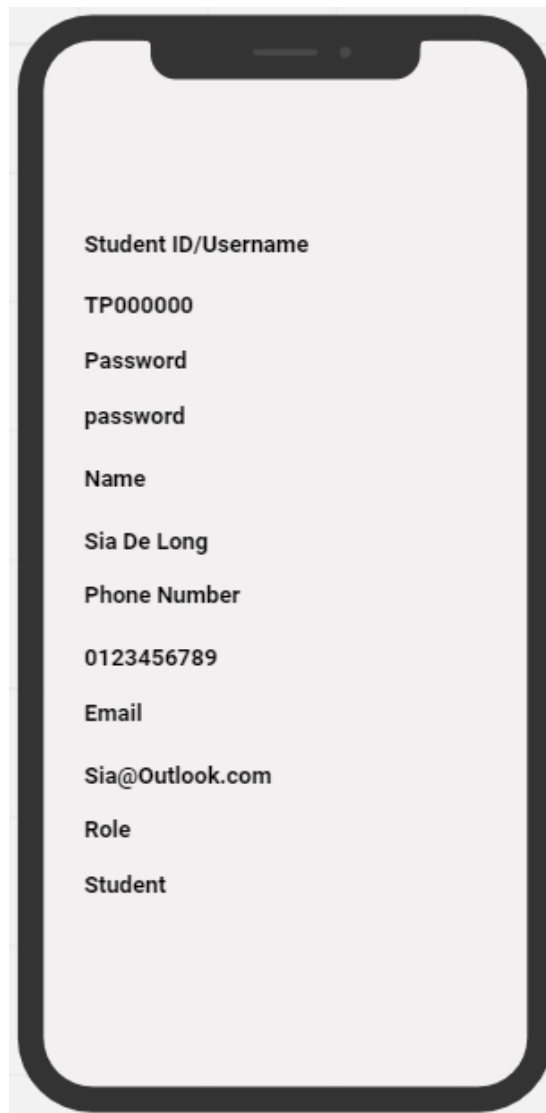


Figure 14: Mock-up of Personal Information Screen

This is the personal information activity from profile tab, it will show the account personal information for the user.

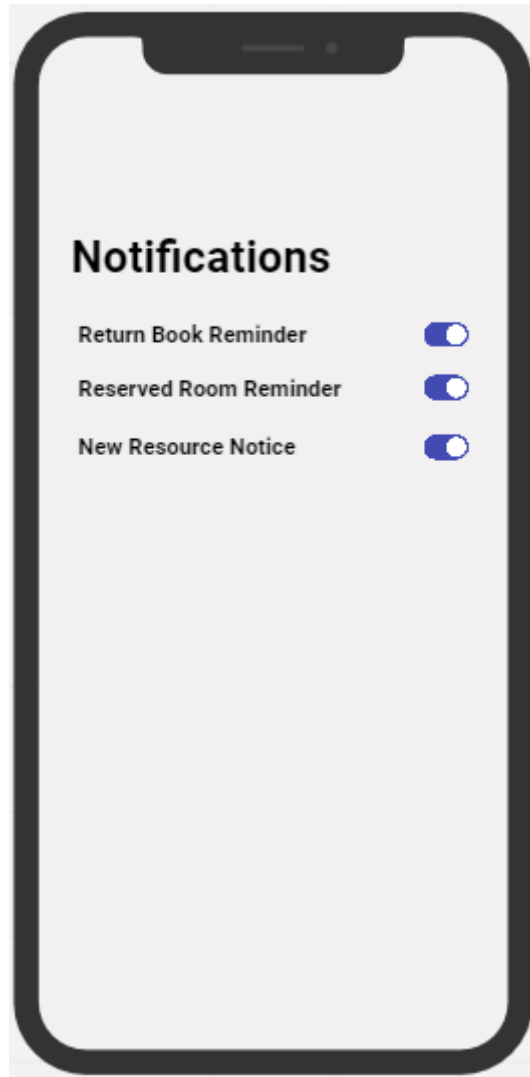


Figure 15: Mock-up of Notifications Screen

This is the notifications activity from profile tab, it will show the notifications of the application, users can choose whether to enable or disable for the notifications function by own preferences.

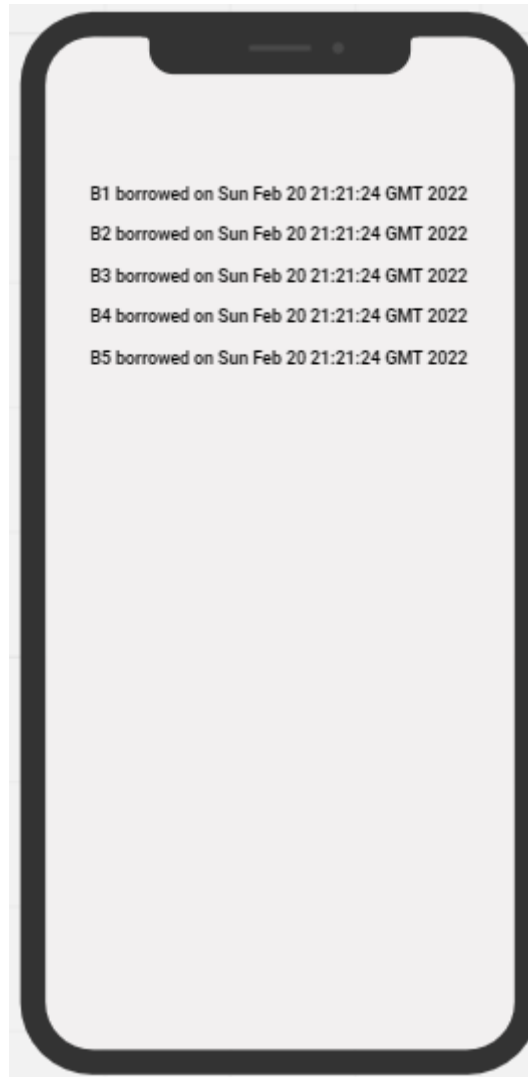


Figure 16: Mock-up of Borrowed Book Screen

This is the borrowed book activity from profile tab, it will show the borrowed book history of the current account, but if the account is admin role, then it will show every account borrow history.



Figure 17: Mock-up of Reserved Room Screen

This is the reserved room activity from profile tab, it will show the reserved room history of the current account, but if the account is admin role, then it will show every account reserve history.

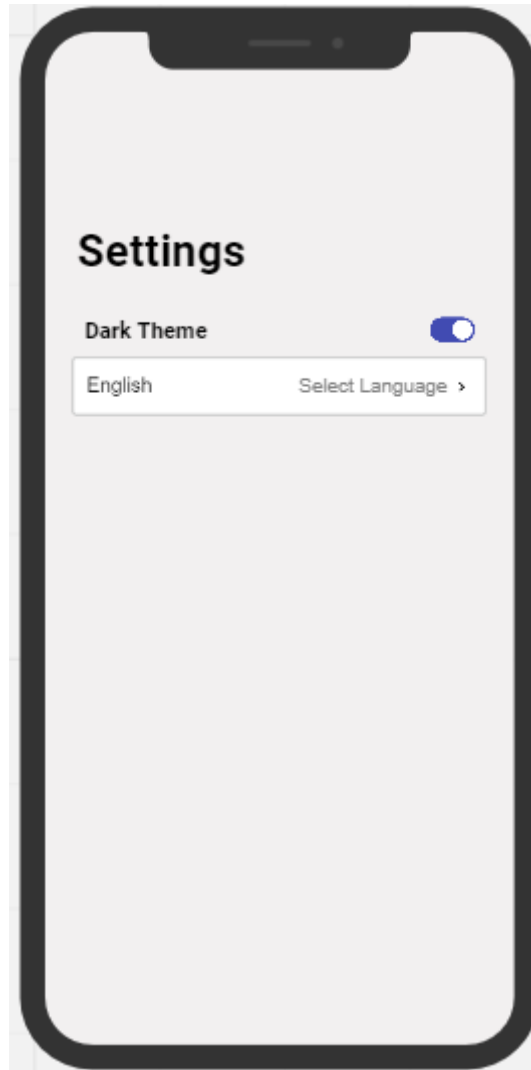
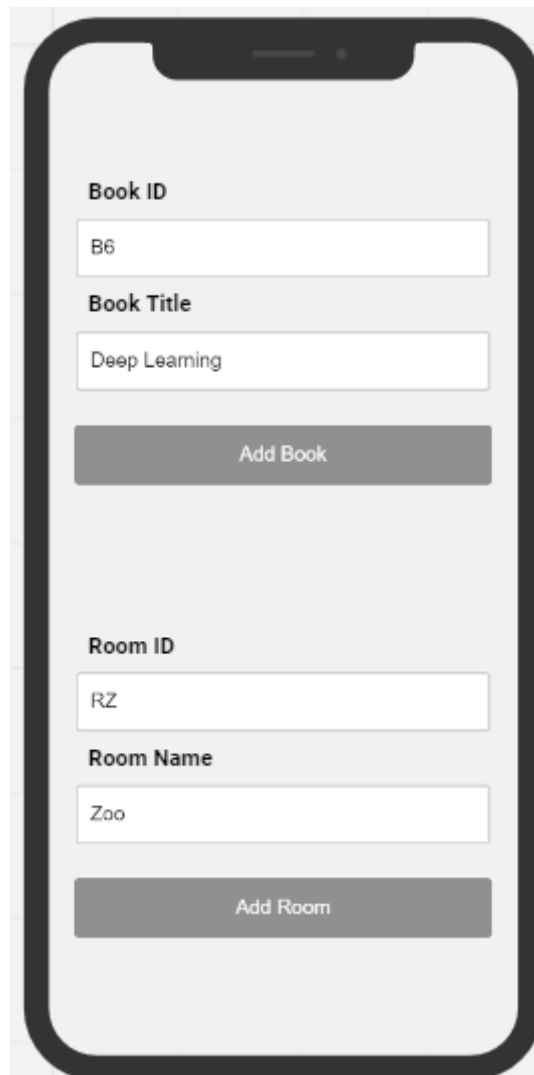


Figure 18: Mock-up of Settings Screen

This is the settings activity from profile tab, users can change the theme of the application to dark theme and also change the language of the application by their own preferences.



The image shows a mobile app interface for an admin function. It features two distinct sections for adding data. The first section, titled 'Book ID', has a text input field containing 'B6' and a label 'Book Title' above another text input field containing 'Deep Learning'. Below these fields is a grey button labeled 'Add Book'. The second section, titled 'Room ID', has a text input field containing 'RZ' and a label 'Room Name' above another text input field containing 'Zoo'. Below these fields is a grey button labeled 'Add Room'. The entire interface is displayed within a black smartphone frame.

Figure 19: Mock-up of Admin Function Screen

This is the admin function activity from profile tab, admin can add new book or new room to be borrowed or reserved respectively to the database.

3.0 Implementation

3.1 Technique & Framework

The application will be using a local database API, since the library app is heavily relying on server to read data from database and display to the client side, the API can be really useful in this case to manage database related process, so that most of the features can be accomplished successfully. Besides that, a simple Wallet API is needed to enable the wallet functionality in the system. Moreover, a QR code scanner API is also needed to achieve the scan to borrow concept in the app. Last but not least, a push notification API is used for sending reminder of book return date to the users.

3.2 Data Passing

Since the account information is needed to be recorded for most of the main use cases, it should be able to be passed to each activity or fragment without any issue. The first data passing will be occurred after a success account login, the account information will be transformed to an array list to allow it to use the putExtra function. For the data passing to activity UI, putExtra of Intent will be used while, for fragment UI the data will be passed using the parameter of the constructor. Hence, both of the situation will be shown as source code below.

```
val accountList: ArrayList<String> = arrayListOf(account.accountID,  
    account.name,  
    account.phoneNo,  
    account.email,  
    account.role)  
  
intent = Intent( packageContext: this, Homepage::class.java)  
intent.putExtra( name: "account", accountList)
```

```
account = intent.getStringArrayListExtra( name: "account")!!.toList()
```

Figure 20: Data Passing to Activity

```

val fragmentBook: FragmentBook = FragmentBook(account)

class FragmentBook(account: List<String>) : Fragment() {

    private val account: List<String> = account

```

Figure 21: Data Passing to Fragment

3.3 Tab Bar Adapter

Since the homepage of the application will be organised by tabs, an adapter for it is needed to switch between tabs. Hence, a class is created to guide which fragment to execute when the tab is selected which is shown as below.

```

@Suppress("DEPRECATION")
internal class TabBarAdapter(var context: Context, fm: FragmentManager, var
totalTabs: Int, account: List<String>) :
    FragmentPagerAdapter(fm) {

    private val account: List<String> = account
    val fragmentBook: FragmentBook = FragmentBook(account)
    val fragmentEResource: FragmentEResource = FragmentEResource()
    val fragmentRoom: FragmentRoom = FragmentRoom(account)
    val fragmentWallet: FragmentWallet = FragmentWallet()
    val fragmentProfile: FragmentProfile = FragmentProfile(account)

    override fun getCount(): Int {
        return totalTabs
    }

    override fun getItem(position: Int): Fragment {
        return when (position) {
            0 -> {
                return fragmentBook
            }
            1 -> {
                //fragmentEResource.account = account
                return fragmentEResource
            }
            2 -> {
                return fragmentRoom
            }
            3 -> {
                //fragmentWallet.account = account
                return fragmentWallet
            }
        }
    }
}

```

```

    }
    4 -> {

        return fragmentProfile
    }
    else -> getItem(position)
}
}
}

```

Figure 22: Tab Bar Adapter

After the adapter class is completed, it can then be declared as an adapter for the tab layout which is happened on the homepage activity and is able to set the tab selection listener to act differently every time a tab is selected. Hence, the source code of it is shown as below.

```

var tabLayout = findViewById<TabLayout>(R.id.tabLayout)
var viewPager = findViewById<ViewPager>(R.id.viewPager)

viewPager.adapter = TabBarAdapter( context: this, supportFragmentManager, tabLayout.tabCount, account)
viewPager.addOnPageChangeListener(TabLayout.TabLayoutOnPageChangeListener(tabLayout))
tabLayout.addOnTabSelectedListener(object : TabLayout.OnTabSelectedListener {
    override fun onTabSelected(tab: TabLayout.Tab) {
        viewPager.currentItem = tab.position
    }
    override fun onTabUnselected(tab: TabLayout.Tab) {}
    override fun onTabReselected(tab: TabLayout.Tab) {}
})

```

Figure 23: Tab Adapter Usage

3.4 Recycler View

To create a dynamic e-resource list, an adapter which extended from a view holder should be built to properly assign value to the UI component every time an item added to the list which is shown as source code below.

```

class EResourceRecyclerViewAdapter (private var eresourcelist:
List<EResource> ): RecyclerView.Adapter<EResourceViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
EResourceViewHolder {
        val itemView =
LayoutInflater.from(parent.context).inflate(R.layout.eresource_item_view,
parent, false)
        return EResourceViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: EResourceViewHolder, position:
Int) {
        val eResourceItem = eresourcelist[position]
        holder.name.text = eResourceItem.getName()
    }
}

```

```

        holder.type.text = eResourceItem.getType()
    }
    override fun getItemCount(): Int {
        return eresourcelist.size
    }
}

class EResourceViewHolder (view: View) : RecyclerView.ViewHolder (view) {
    var name = view.findViewById<TextView>(R.id.textViewEResourceName)
    var type = view.findViewById<TextView>(R.id.textViewEResourceType)
}

```

Figure 24: Recycler View Adapter

After the adapter class is completed, it can then be declared as an adapter for the recyclerView component on the UI which is happened on the e-resource fragment and is able to add all the item to the list and display on it. Hence, the source code of it is shown as below.

```

val recyclerView = view.findViewById<RecyclerView>(com.example.libraryapp.R.id.RecycleViewEResource)
recyclerView.layoutManager = LinearLayoutManager(view.context)
recyclerView.itemAnimator = DefaultItemAnimator()

eresourceAdapter = EResourceRecyclerViewAdapter(eresourcelist)
recyclerView.adapter = eresourceAdapter

```

Figure 25: Recycler View Usage

3.5 Live Data & View Model

Every time the application retrieve data from the database, it will follow the flow of the data architecture shown on the Application Design section. Therefore, one of the samples will be demonstrated on this section which is Book Table.

```

@Entity(tableName = "books")
data class Book(

    @PrimaryKey var bookID: String,

    @ColumnInfo(name = "name") var name: String,
)

```

Figure 26: Book Table

First and foremost, the Book table will be created by using the @Entity notation from RoomDatabase and data keyword to configure the primary key and column attributes.

```
@Dao
interface BookDao {

    @Query(value: "SELECT * FROM books ORDER BY name ASC")
    fun getBooks(): Flow<List<Book>>

    @Insert
    suspend fun addBook(book: Book)
}
```

Figure 27: Book DAO

Then, a DAO for the table will be construct by using interface keyword to build an abstract class with queries methods for that table, suspend keyword will be used to temporary suspend the declared method.

```
@Database(entities = [Book::class], version = 1)
abstract class BookDatabase: RoomDatabase() {

    abstract val bookDao: BookDao
}

private lateinit var INSTANCE: BookDatabase

fun getBookDatabase(context: Context): BookDatabase {
    synchronized(BookDatabase::class.java) {
        if (!::INSTANCE.isInitialized) {
            INSTANCE = Room.databaseBuilder(context.applicationContext,
                BookDatabase::class.java,
                name: "books").build()
        }
    }
    return INSTANCE
}
```

Figure 28: Book Database

With the DAO built, the database class can be constructed by connect it with the DAO. In the same time, a `getDatabase` can also be declared to create a database file to the local computer if the database is newly used.

```
class BookRepository(private val bookDao: BookDao) {  
  
    val allBooks: Flow<List<Book>> = bookDao.getBooks()  
  
    @Suppress( ...names: "RedundantSuspendModifier")  
    @WorkerThread  
    suspend fun addBook(book: Book) {  
        bookDao.addBook(book)  
    }  
}
```

Figure 29: Book Repository

The next step is linking the DAO with the repository as mentioned on the data architecture diagram in the purpose of creating a clean API for UI to communicate with the database.

```
class BookViewModel (private val repositoryBook: BookRepository) : ViewModel() {  
  
    val allBooks: LiveData<List<Book>> = repositoryBook.allBooks.asLiveData()  
  
    fun addBook(book: Book) = viewModelScope.launch { this: CoroutineScope  
        repositoryBook.addBook(book)  
    }  
}  
  
class BookViewModelFactory(private val repositoryBook: BookRepository) : ViewModelProvider.Factory {  
    override fun <T : ViewModel> create(modelClass: Class<T>): T {  
        if (modelClass.isAssignableFrom(BookViewModel::class.java)) {  
            @Suppress( ...names: "UNCHECKED_CAST")  
            return BookViewModel(repositoryBook) as T  
        }  
        throw IllegalArgumentException("Unknown ViewModel class")  
    }  
}
```

Figure 30: Book View Model

Finally, the repository will be used to declare a View Model and View Model Factory where View Model Factory will be used on the UI site to create View Model instant, while View Model will be working on the coroutine scope to not blocking the main UI thread when executing the queries of the DAO.

```
private val bookViewModel: BookViewModel by viewModels{  
    BookViewModelFactory(BookRepository(getBookDatabase(context: this).bookDao))  
}
```

Figure 31: Book View Model Factory

Therefore, in the manual borrow book activity, the View Model will be declared as shown above by using the View Model Factory, while the data can be accessed by the source code below.

```
bookViewModel.allBooks.observe(owner: this) { books ->
```

Figure 32: Live Data Observation

Observe is needed because the data is stored as live data and it will notify the system every time the data updated. From the allBooks of the View Model, the data will be declared as books in a list form which can then be separated and be used.

4.0 Application Testing

4.1 Espresso Test

```
@LargeTest
@RunWith(AndroidJUnit4::class)
class LoginTest {

    @Rule
    @JvmField
    var mActivityTestRule = ActivityTestRule(MainActivity::class.java)

    @Test
    fun loginTest() {
        val appCompatEditText = onView(
            allOf(
                withId(R.id.editTextID),
                childAtPosition(
                    childAtPosition(
                        withId(android.R.id.content),
                        0
                    ),
                    4
                ),
                isDisplayed()
            )
        )
        appCompatEditText.perform(replaceText("admin"),
            closeSoftKeyboard())

        val appCompatEditText2 = onView(
            allOf(
                withId(R.id.editTextPassword),
                childAtPosition(
                    childAtPosition(
                        withId(android.R.id.content),
                        0
                    ),
                    6
                ),
                isDisplayed()
            )
        )
        appCompatEditText2.perform(replaceText("admin"),
            closeSoftKeyboard())

        val materialButton = onView(
            allOf(
                withId(R.id.buttonSignIn), withText("Sign In"),
                childAtPosition(
                    childAtPosition(
                        withId(android.R.id.content),
                        0
                    ),
                    1
                ),
                isDisplayed()
            )
        )
    }
}
```



```

        materialButton.perform(click())
    }

    private fun childAtPosition(
        parentMatcher: Matcher<View>, position: Int
    ): Matcher<View> {

        return object : TypeSafeMatcher<View>() {
            override fun describeTo(description: Description) {
                description.appendText("Child at position $position in
parent ")
                parentMatcher.describeTo(description)
            }

            public override fun matchesSafely(view: View): Boolean {
                val parent = view.parent
                return parent is ViewGroup && parentMatcher.matches(parent)
                    && view == parent.getChildAt(position)
            }
        }
    }
}

```

Figure 33: Login Test

Figure above shows the login UI test for the application using espresso test. The result of the test is passed successfully without any issues and the report is shown as below.

Tests	Duration	Pixel_5_API_30
✓ Test Results	2 s	1/1
✓ LoginTest	2 s	1/1
✓ loginTest	2 s	✓

LoginTest: 1 total, 1 passed	1.55 s
Collapse Expand	
com.example.libraryapp.LoginTest	1.55 s
loginTest	passed 1.55 s

Figure 34: Login Test Result

4.2 Junit test

```
class BalanceTest {

    @Test
    fun subtraction_isCorrect() {
        assertEquals(10, 20 - 10)
    }

    @Test
    fun check_Wallet_Subtraction() {

        val compensationPrice = 20
        val balance = 10
        val result = compensationPrice - balance

        assertEquals(10, result)
    }

    @Test
    fun add_isCorrect() {

        assertEquals(10, 0 + 10)
    }

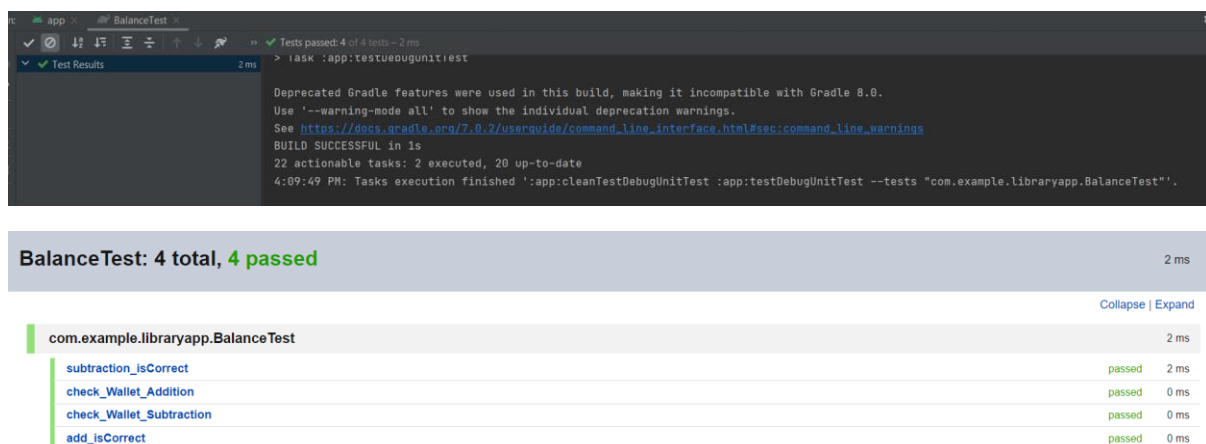
    @Test
    fun check_Wallet_Addition() {

        val currentBalance = 0
        val reload = 10
        val result = currentBalance + reload

        assertEquals(10, result)
    }
}
```

Figure 35: Balance Test

Figure above shows the balance unit test for the application using JUnit test since there is calculation occurring in that activity. The result of the test is passed successfully without any issues and the report is shown as below.



BalanceTest: 4 total, 4 passed 2 ms

Test Method	Result	Duration
subtraction_isCorrect	passed	2 ms
check_Wallet_Addition	passed	0 ms
check_Wallet_Subtraction	passed	0 ms
add_isCorrect	passed	0 ms

Figure 36: Balance Test Result

5.0 User Manual

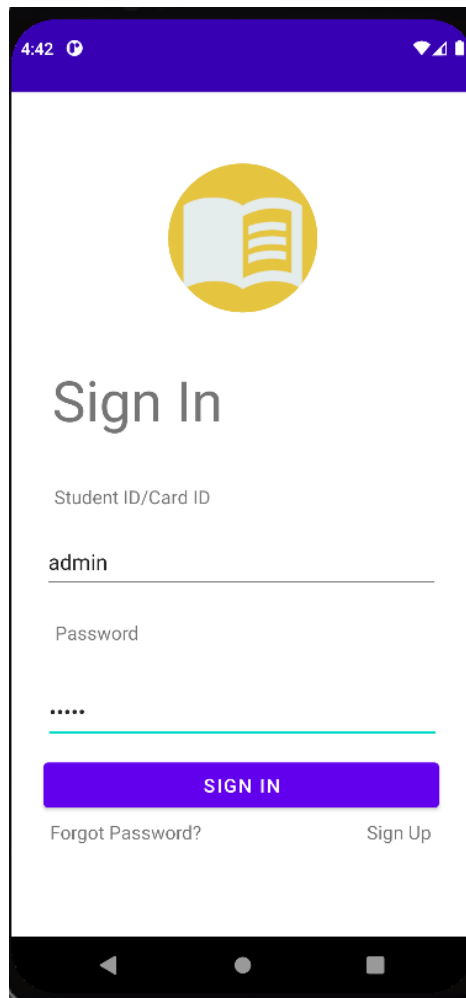


Figure 37: Login

This is the first screen that displayed when the application launched, for the simplicity of application interaction, the admin account is suggested to be used for the use of student and admin function at the same time. Hence, the ID and password are both “admin”. Nevertheless, a normal account can also be registered if the sign-up text is pressed.

The image shows a mobile application interface for account registration. At the top, there is a status bar with the time 4:44 and various icons. The form consists of several input fields, each with a label above it: 'Student ID/Username' with the value 'TP060810', 'Password' with three dots indicating a hidden password, 'Name' with the value 'Sia De Long', 'Phone Number' with the value '0123456789', 'Email' with the value 'Sia@outlook.com', and 'Role' with a dropdown menu currently showing 'Student'. Below these fields is a prominent blue button labeled 'SIGN UP'. The entire form is set against a white background within a black border representing the phone screen.

Figure 38: Sign-up

Before any main functionalities showed, this is the account registration page for the application that will be displayed if the user press on sign-up text on the login page. Every slot is compulsory to be filled else the system will not register the account and pop out a toast message to notify.

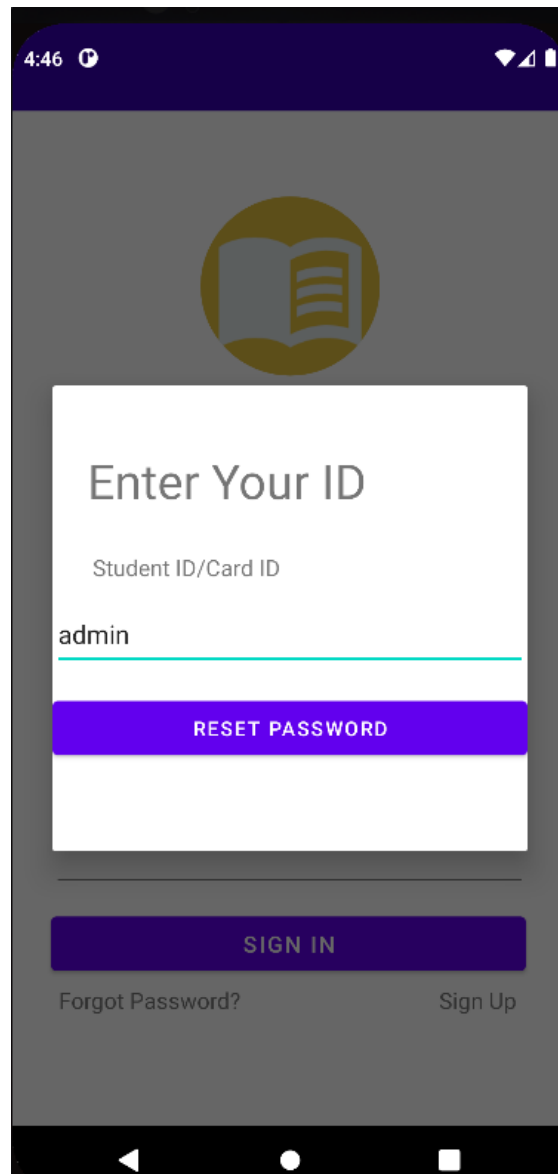


Figure 39: Forgot Password

To demonstrate the forgot password function, a dialog fragment will pop out upon pressing the forgot password text on the login page to request for the user ID to send an email to your account email address to solve the password issue.

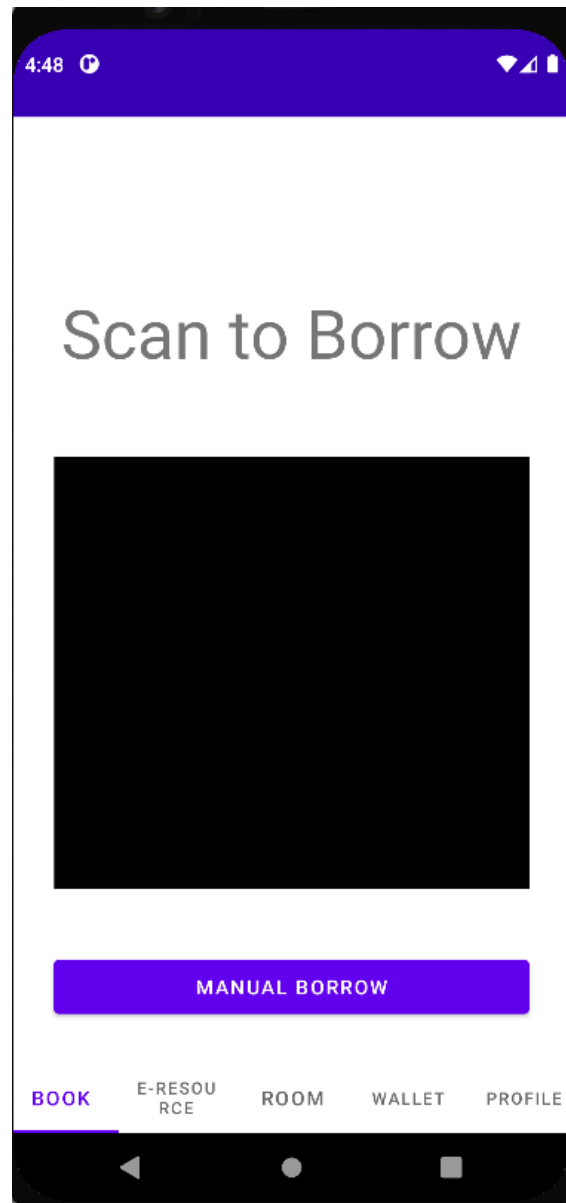


Figure 40: Scan to Borrow

After login to an account, the first page will be shown as above, users are free to switch between tabs to find the function that they want to interact to. First and foremost is the borrow book function, the black screen is supposing display the camera capture from the device to scan any QR code that it detects. However, the actual function is not fully implemented but as a prototype for the application. If user want to borrow a book, then press on the manual borrow button.

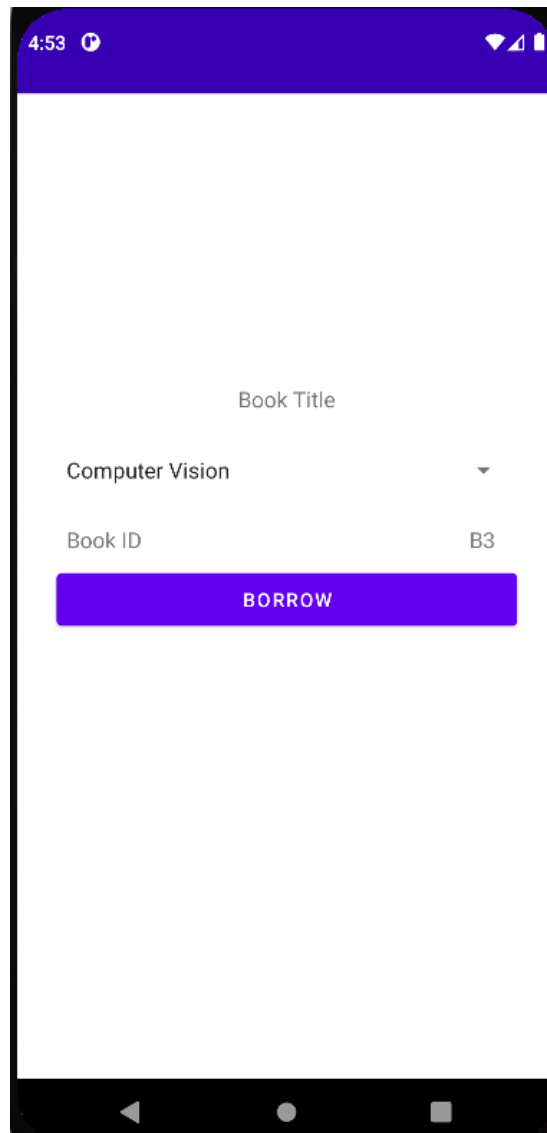


Figure 41: Manual Borrow

This is the manual borrow book screen where user can choose from the spinner to select book title that the user wants to borrow, after pressing the borrow button, the title will be removed from the spinner and add a record to the database.

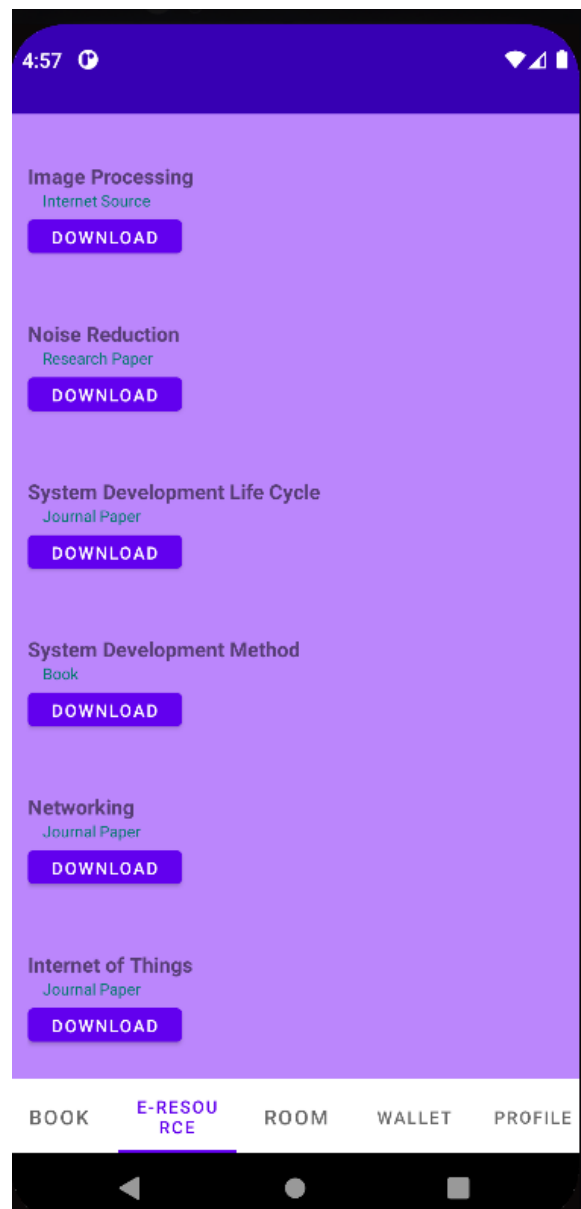


Figure 42: E-Resource

If the user press on the e-resource tab, then all the e-resource will be shown as a list to be scrolled down. However, the download function is not implemented fully since it requires to connect to third-party bibliographic database and download as pdf file, it may involve permission request and copyright law.

The image shows a mobile application interface for reserving a room. At the top, there is a status bar with the time 5:00 and various icons. The main content area has a light blue background. It contains three input fields: 'Date' with the value '05/02/2022', 'Start Time' with the value '09:30', and 'End Time' with the value '13:00'. Below these fields is a red button labeled 'SEARCH FOR AVAILABLE ROOM'. At the bottom, there is a navigation bar with five tabs: 'BOOK', 'E-RESOURCE', 'ROOM' (which is highlighted with a red underline), 'WALLET', and 'PROFILE'. The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

Figure 43: Reserve Date

For the room tab of the homepage, the user is first need to input the date, start time and end time. The format of the date is expected to be dd/mm/yyyy while the time format is expected to be hh:mm with the use of 24-hour format to let the system work properly.

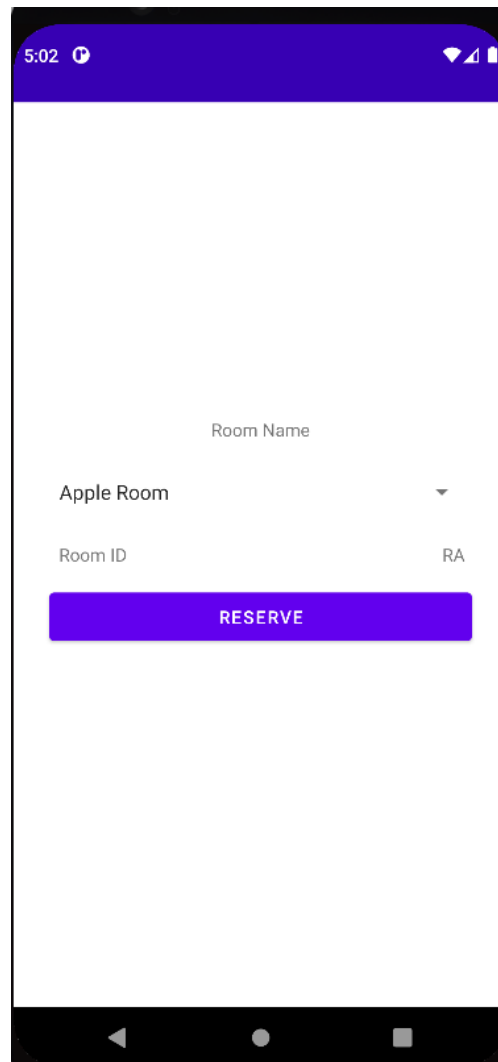


Figure 44: Select Room

The available room for that time slot will be display on the spinner to let the user to pick the room to reserve, after pressing the reserve button, the room name will be removed from the spinner and add a record to the database.

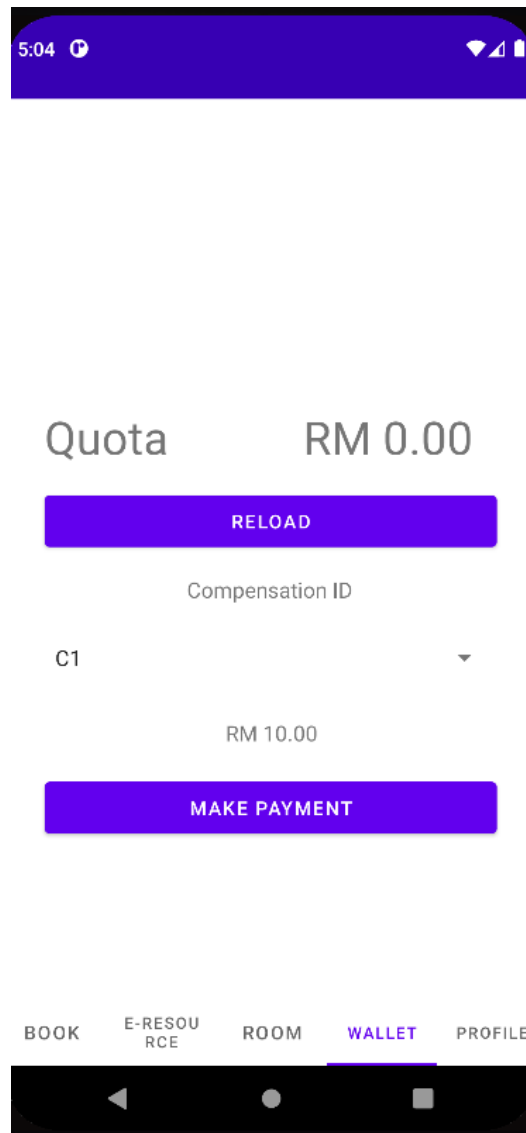


Figure 45: E-Wallet

It is also a prototype for the wallet function because it requires to build a transaction network from the user selected bank merchant to the system bank account which require advanced technical skill. However, for demonstration purpose, the reload button can reload the balance by RM10.00 every time it is pressed. Besides that, the user can choose the compensation ID to be paid from the spinner with different amount of price, balance sufficiency will be checked every time the user trying to make a payment, after a successful payment, the compensation ID will be removed from the spinner.

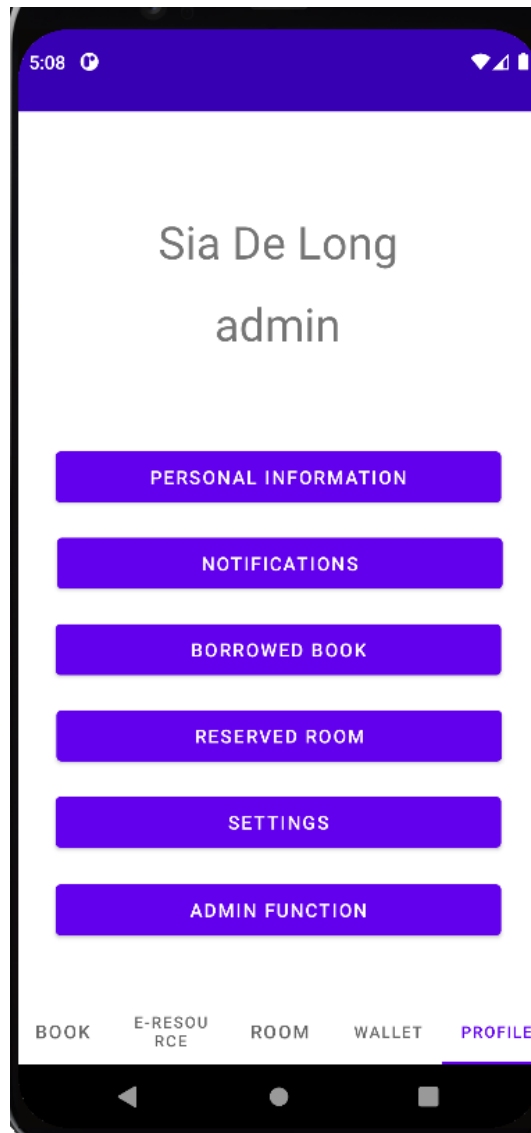


Figure 46: Profile

The profile tab functions are working as the mock-ups described aside from the notifications and settings which both only act as a prototype without actual function. Nevertheless, the admin function is working and will not show to users who are not admin role.

The screenshot shows a mobile application interface with a white background. At the top, there is a status bar with the time 5:15 and icons for signal, Wi-Fi, and battery. The main content area has a light gray background. It contains two sections: 'Add Book' and 'Add Room'. The 'Add Book' section has a label 'Book ID' above a text input field containing 'B6', a label 'Book Title' above a text input field containing 'Image Processing', and a blue button labeled 'ADD BOOK'. The 'Add Room' section has a label 'Room ID' above a text input field containing 'RZ', a label 'Room Name' above a text input field containing 'Zoo', and a blue button labeled 'ADD ROOM'. At the bottom, there is a black navigation bar with three white icons: a back arrow, a circle, and a square.

Figure 47: Admin Function

Admin can freely add new book and new room to the database as shown above, while there will be a checking for avoiding ID duplication in the database before insert the data to the table.

6.0 Conclusion

In a nutshell, a simple mobile application is developed successfully to partially overcome the problem since some of the idea for the application is not implemented yet and it require for a more advanced knowledge to achieve. With that being said, the improvement of the application can be done in the future to completely solve the issue by conducting study for the skill to develop advance functionalities. I hope the community will aware of the problem and cooperative with each other to bring out a perfect solution for it, either the library can be making full use by the user or the library might evolve to another form.

References

- Andrew, A. (6th May, 2021). *Publisher Weekly*. Retrieved from Report Urges Library Leaders to Address Decline in Public Library Usage Stats: 2021
- Cambridge Dictionary. (2022). *Cambridge Dictionary*. Retrieved from Library: <https://dictionary.cambridge.org/dictionary/english/library>
- Jacquelyn, B. (4th January, 2022). *techjury*. Retrieved from How Fast Is Technology Advancing in 2021?: <https://techjury.net/blog/how-fast-is-technology-growing/#gref>

Appendix

Presentation video:

<https://drive.google.com/file/d/1v9v9wBgwOLmm2SEmDGbpKrwydSScrvHT/view?usp=sharing>