[redacted]

Natibundit Suntraranusorn- 300360042

INTRODUCTION

Vehicle owners often face difficulties in managing and scheduling their car maintenance appointments. They may struggle to find a reliable service provider, schedule appointments at a convenient time, or receive updates on the status of their appointments.

Features

The Car Service app is designed to provide a convenient and user-friendly experience for vehicle owners to interact with service providers. The following features has been included in the project:

- Registration and login: Allows users to register and log in to the app, providing a secure and personalized experience. Service providers are already registered to the system.
- Searching for service providers: Allows vehicle owners to search for service providers nearby based on location or view their favourite providers.
- Booking an appointment: Allows vehicle owners to request a service (e.g. oil change) and book an appointment with a service provider with an option between drop-off or pick-up options.
- Managing appointments: Allows vehicle owners and service providers to view or cancel their appointments. Service providers also can edit the appointments like changing the time and date.
- Viewing Service History: Allows users to view their service history, so that they can keep track of their services.
- Viewing and editing user profiles: Allows vehicle owners to view and edit their personal information and password.
- Searching customer: Allows service providers to search a customer with their name.
- Notifications: Application sends notification about upcoming appointments for both service providers and vehicle owners.
- Create, view, edit user information: Allows service providers to view, create a new user, or edit user information.
- Deleting a user: Allows service providers to delete a user from the system.
- Add, view, and edit the service details: Allows service providers to add, view and edit the services that they provide.
- Generating service reports: Allows service providers to generate reports on services provided.

Project Resource Codes : https://github.com/neslihanaydin/douglas-car-service

Database details

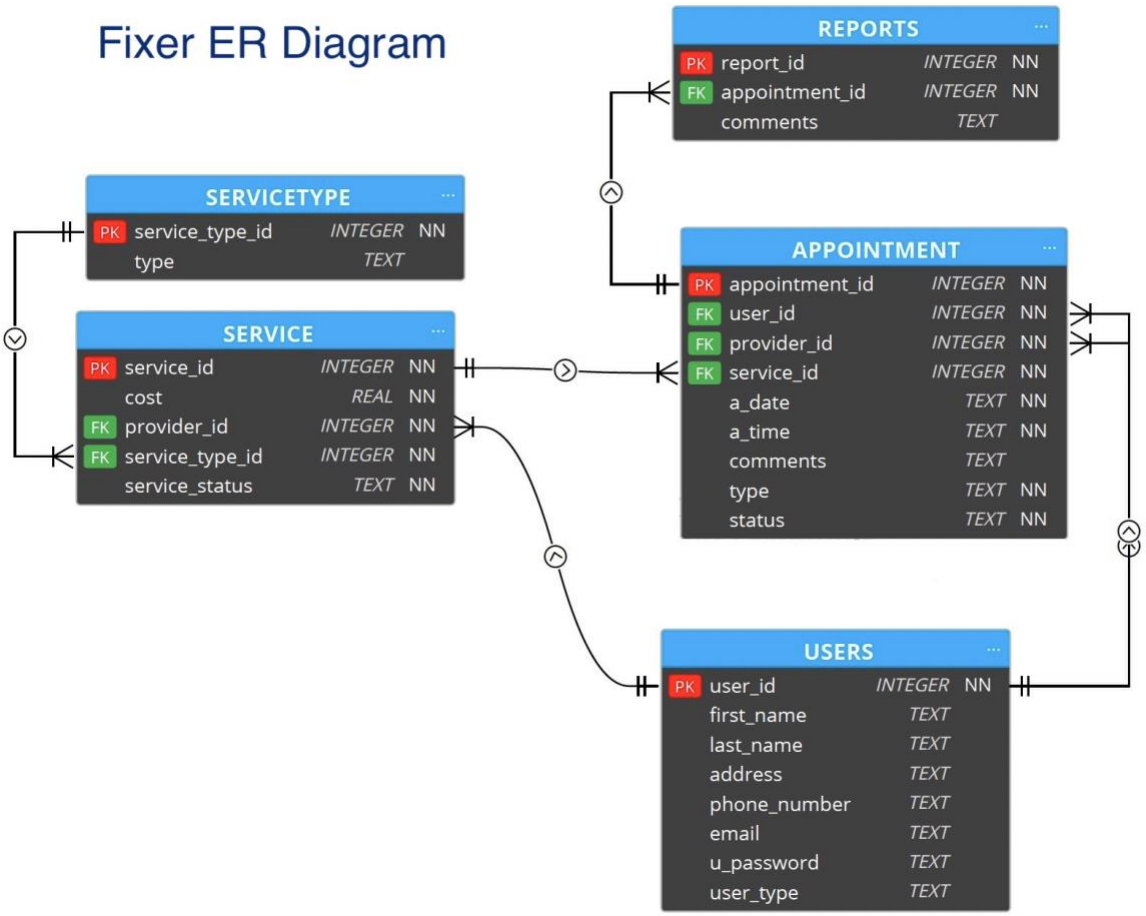
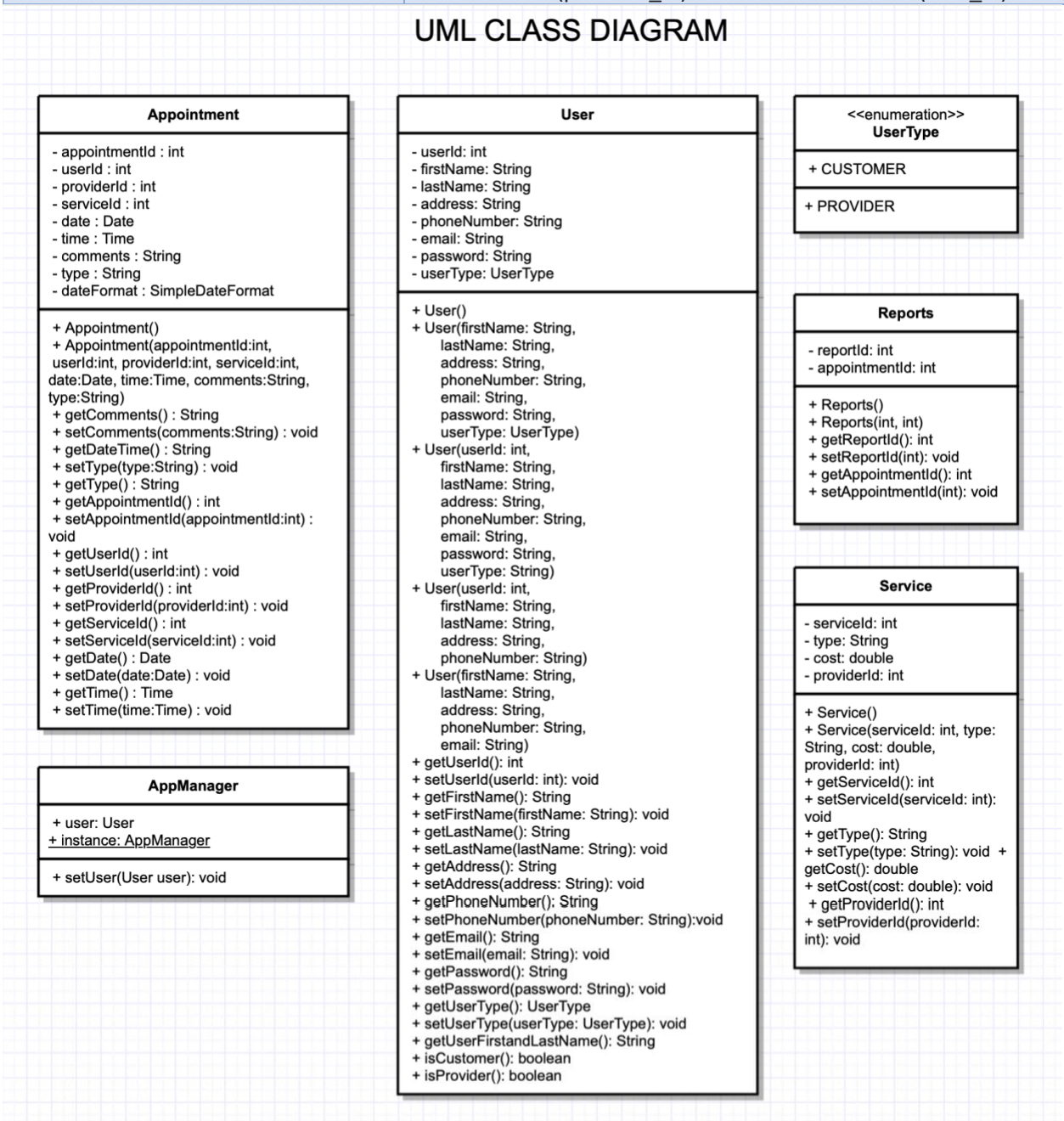


TABLE DETAILS	
USERS user_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL first_name TEXT last_name TEXT address TEXT phone_number TEXT email TEXT u_password TEXT user_type TEXT	**APPOINTMENT** appointment_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL user_id INTEGER NOT NULL provider_id INTEGER NOT NULL service_id INTEGER NOT NULL a_date TEXT NOT NULL a_time TEXT NOT NULL comments TEXT type TEXT NOT NULL status TEXT NOT NULL DEFAULT PENDING FOREIGN KEY (service_id) REFERENCES "SERVICE" (service_id) FOREIGN KEY (provider_id) REFERENCES "USERS" (user_id) FOREIGN KEY (user_id) REFERENCES "USERS" (user_id)
SERVICETYPE service_type_id INTEGER PRIMARY KEY NOT NULL type TEXT	

REPORTS	**SERVICE**
report_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL appointment_id INTEGER NOT NULL comments TEXT FOREIGN KEY (appointment_id) REFERENCES "APPOINTMENT" (appointment_id)	service_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL cost REAL NOT NULL provider_id INTEGER NOT NULL service_type_id INTEGER NOT NULL service_status TEXT NOT NULL DEFAULT ACTIVE FOREIGN KEY (service_type_id) REFERENCES "SERVICETYPE" (service_type_id) FOREIGN KEY (provider_id) REFERENCES "USERS" (user_id)

UML CLASS DIAGRAM



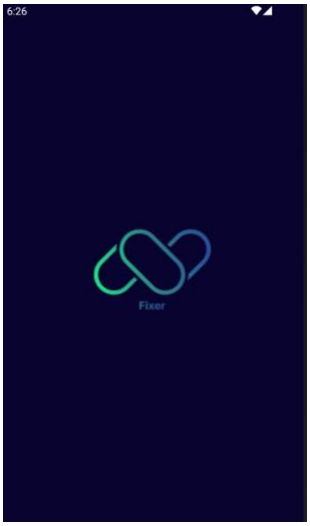
The Fixer application contains a total of 18 screens, including 7 for Service Providers, 8 for Vehicle Owners, and 3 for all users.

- Screens for all users
  - Splash Screen
  - Login Activity
  - Navigation Bar
- Screens for Vehicle Owners
  - Register Activity
  - Book an Appointment Activity
    - Book an Appointment First Activity
    - Booking Appointment Activity
  - Search Service Provider Activity
  - View Appointments Activity
  - Service History Activity
  - Profile Activity
- Screens for Service Provider
  - View Appointments Activity
    - Appointment Details Activity
  - Create Customer Activity
  - Search Customer Activity
    - Customer Details Activity
  - Service History Activity
  - Services Activity

- The application uses a Singleton class structure called AppManager to store login user information in the instance.user object within this class. Throughout the application lifecycle, the user can be accessed through this object when needed.
- The Util class defined under the util package contains methods that allow for conversion between date and time structures used in the database and objects.
- After the user logs in or registers, their information is saved using SharedPreferences, and the user automatically logs in when they access the application again. When the user logs out, the SharedPreferences data is deleted.
- At the start of the application, test data is added to the application's database through the methods for adding test data found in the DatabaseHelper class. This means that registered users and their appointments are already present when the application is opened.
- A custom FixerToolbar has been defined within the application to be used on screens. This allows for the back button to be disabled or the navigation button to be disabled on relevant screens.

Splash Screen

Contributors: Natibundit Suntraranusorn



It is an image to display during a loading process. For this app, it occurs when an emulator is immediately run that displays a waiting scene before the user login.

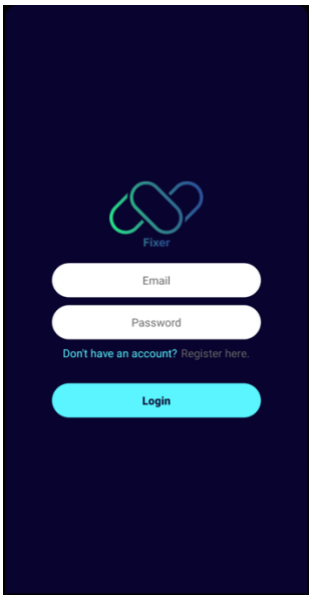
How was created:

- Creating a splash layout.
- Using TimeTask and Timer for managing a delay time of splash scene.

Functionality:

- Creating a Task object to run a splash scene by setting a time delay on a Timer object. After finishing a time delay, the login activity will be shown by using Intent.

Login



Contributors: [redacted]

This is the first interactive page of the application. The login page receives an email and password to give access to the user. Also, there is a link to send the user to the registration page.

How was created:

- The login page is built with a ConstraintLayout as the main container.
- The form is uses a LinearLayouts to hold horizontal aligments and its inputs.

Functionality:

- The form validates if any of both inputs are empty or not by using the **isEmpty()** logical method.
- The form validates if the email entered has a valid email format.
- **checkUserCredentials()** is a Boolean function from the **DatabaseHelper** class that will evaluate if the user is part of the system or not. If it's true, the use will be pushed to the next screen with all its data.
- **SharedPreferences** are used to store the user's credentials and perform an auto login after the user has logged for the first time.
- A Singleton is used to hold the user object during the entire application. Once the user logs in, the data is carried through all applications as needed.

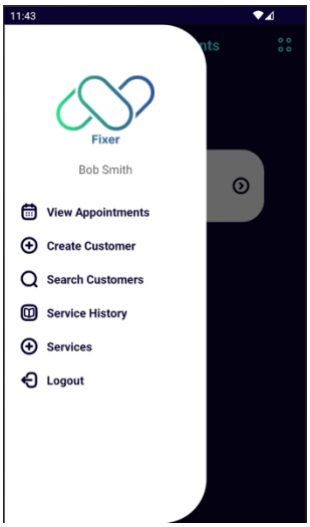
- In order to remove the status bar, the actionBar and make the background full screen, the **setFlags()** is used with **FLAG_FULLSCREEN**.

Extras

- Custom logo
- Custom editText and button design using a drawable background with rounder corners.
- Toast is used to provide feedback if the user cannot log in.

Navigation Bar

Contributors: Natibundit Suntraranusorn



A navigation bar is a toolbar providing multiple functions that facilitate shortcut commands for user access.

How was created:

- Creating a Navigation Layout that comprises of Linear Layout and Relative Layout (a side menu bar)
- Implement a Navigation activity
- Creating RecylerView and Main Adapter

Functionality:

1. **Navigation Activity**

- Initialize a drawerLayout.widget, menu bar image, RecyclerView, ArrayList<titleDrawer> for example, Book an appointment, Search service provider for customer and Adapter.
- Creating a Close Drawer method to control the start and the close function.
- On create method
  - Declaring the attributes of the navigation layout into variables.
  - Clear ArrayList<titleDrawer>.
  - Using Try/Catch to avoid program crashes and using IF statement to provide conditions to display customer navigation or provider navigation regarding ID login.
  - Setting RecyclerView by following MainAdapter (customize adapter).
  - Setting Onclick of btnMenu by using a Close Drawer method.
  - Set the condition to show a first page for the customer that is Book Appointment first page activity and display a first page for the provider, which is the View Appointment activity.
  - Creating on pause method to close a navigation bar.

2. **Main Adapter**

- Following the Recyclerview implementation that we learn in class.
  - OnCreateViewHolder, OnBindViewHolder
  - Setting text and icons and fixer custom logout dialog box

- Coding condition to access the customer's purpose page related to the position.
- The items list is created by taking different elements according to the type of the user in the AppManager class, and it is sent to the MainAdapter to create the content of the recycler view. When the corresponding items are clicked within the MainAdapter, the user is directed to the relevant activities.

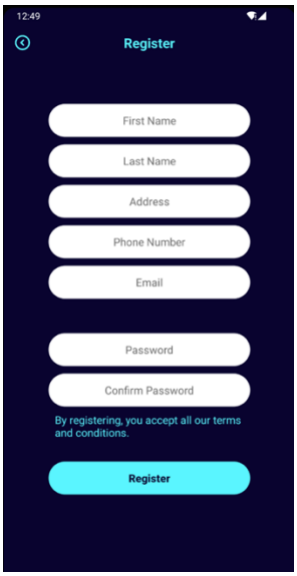
NavigationActivity.java

```
if(user.isCustomer()){
    items.add(new ItemDrawer(R.drawable.icon_add, "Book an Appointment"));
    items.add(new ItemDrawer(R.drawable.icon_search, "Search Service Provider"));
    items.add(new ItemDrawer(R.drawable.icon_calendar, "View Appointments"));
    items.add(new ItemDrawer(R.drawable.icon_history, "Service History"));
    items.add(new ItemDrawer(R.drawable.icon_user, "Profile"));
    items.add(new ItemDrawer(R.drawable.icon_logout, "Logout"));
}
```

MainAdapter.java

```
case 1:
    activity.startActivity(new Intent(activity, SearchProvider.class)
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
    break;
case 2:
    activity.startActivity(new Intent(activity, ViewAppointments.class)
            .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
    break;
```

Registration



Contributors: [redacted]

The registration page holds the main form to register a user.

How was created:

- A LinearLayout with vertical orientation is used to hold the entire form.
- All editText are made with a custom design using a drawable.

Functionality:

- Every editText field is validated if it's empty or not to avoid registering empty strings.
- The email is used to check if an user has previously registered it with another account. If so, the user is prompted to the login page. If not, the user is created in the Database by using the **addUser()** method from the **DatabaseHelper** class that receives an object of type user and push the user to the next page.

RegistrationActivity.java

```
//Adding user
User newUser = new User(fName,lName,address,phone,email,password,
User.UserType.CUSTOMER);
dbHelper = new DatabaseHelper(getApplicationContext());
dbHelper.addUser(newUser);
```

Extras

- This page as many others uses a custom Toolbar class that holds 4 arguments. The activity, which is being used to display the Toolbar, a back arrow icon, the activity title, and a menu button.

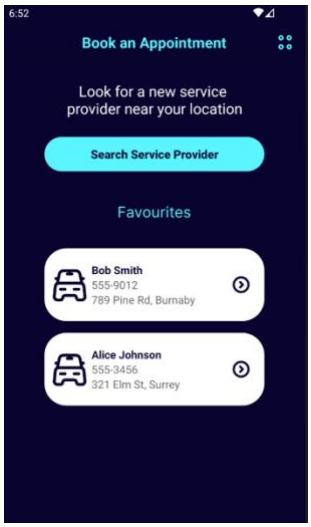
RegistrationActivity.java

```
FixerToolbar.setToolbar(this, "Register", true, false);
```

Book an Appointment

Booking Appointment First Page

Contributors: Natibundit Suntraranusorn



Booking appointment first page displays favorite providers by showing icons. Users can click an icon to reserve an appointment through these items or manually search providers by clicking search provider sorting by city.

How was created:

- Creating Booking an appointment first-page layout and coding activity
  - ➢ view/BookAppointmentFirstActivity.java
  - ➢ layout/activity_book_appointment_first.xml
  - ➢ layout/reclerview_favorite_provider.xml
- Creating Book an appointment adapter
  - ➢ adapter/BookAppointmentFirstAdapter.java

Function:

- **BookAppointmentFirstActivity**

Initialize a drawerLayout.widget, menu bar image, RecyclerView, DB Helper, List<User> provider, List<Appointment> upcomingAppointment

- ♦ Declaring all attributes into variables and settings.
- ♦ When user login, AppManager.instance.user is assigned as logged user.
- ♦ For a Search Service Provider, Intent is used to pass to the Search Provider page.

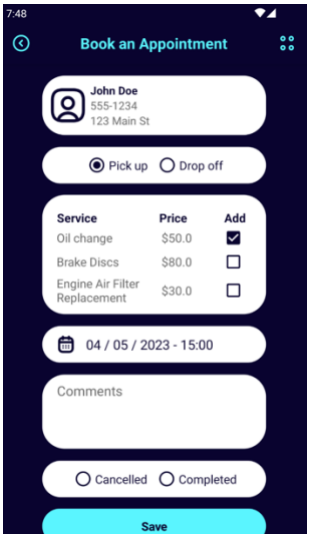
- When a favorite provider item is clicked, Book Appointment page is opened relying on provider ID and using Intent to pass data across the page.

- **BookAppointmentFirstAdapter**

  Following the Recyclerview implementation that we learn in class.
  - OnCreateViewHolder is used to associate between activity and layout.
  - OnBindViewHolder is used to change the contents of all elements of icons in specific order.
  - ItemView is a set of all elements of favorite provider.
  - ViewHolder class has been created for holding and managing widget elements.
  - The purpose of ViewHolder is to connect elements from the layout to ViewHolder.
  - ItemClickListener interface has been created to manage onClick process.
  - When user click the widget, Booking Appointment activity is started.

Booking Appointment Second Page

Contributors: [redacted]



The Book an Appointment page opens when the user searches for a service provider or selects one of their favorite providers on the "User Search Service Provider" page.

How was created:

- Creating Book an Appointment page layout and coding activity
  - view/ BookAppointment.java
  - layout/activity_book_appointment.xml
  - layout/recyclerview_services_costs.xml
- Creating Provider Services Adapter
  - adapter/ProviderServicesAdapter.java

When the user clicks on a provider, the provider's information is accessed through the intent and displayed on the Book an Appointment page.

The user then selects the service options using radio buttons for either pick-up or drop-off. A RecyclerView is used to display the list of services and their prices, which are retrieved from the database using an adapter specific to the provider's services.

The ProviderServicesAdapter adapter is used to list the services and prices. The checkedServiceList list, defined in the BookAppointment class, is updated when the user selects the checkboxes for the desired services.

To enable the user to select a suitable date and time, DatePickerDialog and TimePickerDialog are used. If the user selects a date but not a time, the system automatically adds 12:00.
The user can also add comments in the Comments section, but this is optional.

When the user clicks the Book an Appointment button, the validity of the relevant fields is checked, and a new appointment is added to the new database. If the user has selected multiple services, a separate appointment object is created for each service.

Within this activity, the user type is checked, and if a customer enters this page, the mentioned processes are carried out. However, if a provider enters the page, the reason for the provider to access the page is to modify an existing appointment. Therefore, additional processes are performed within that check for the provider. Details regarding the processes can be found on the "**View Appointments**" page.

The method below is triggered by the onclick method added to the checkboxes displaying the services:

BookAppointment.java

```
@Override
public void onItemClick(View view, int position, boolean isChecked)
{
    if(isChecked) checkedServiceList.add(serviceList.get(position));
    else checkedServiceList.remove(serviceList.get(position));
}
```

ProviderServicesAdapter.java

```
public ViewHolder(@NonNull View itemView) {
    super(itemView);
    txtServiceRc = itemView.findViewById(R.id.txtServiceRc);
    txtCostRc = itemView.findViewById(R.id.txtPriceRc);
    checkBox = itemView.findViewById(R.id.checkRc);
    checkBox.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(checkBox.isChecked()){
              itemClickListener.onItemClick(v, getAdapterPosition(), true);
            } else{
              itemClickListener.onItemClick(v, getAdapterPosition(), false);
            }
        }
    });
}
```

Related SQLite methods for Book Appointment Activity

- `serviceList = dbHelper.getServicesByProviderId(providerId);`

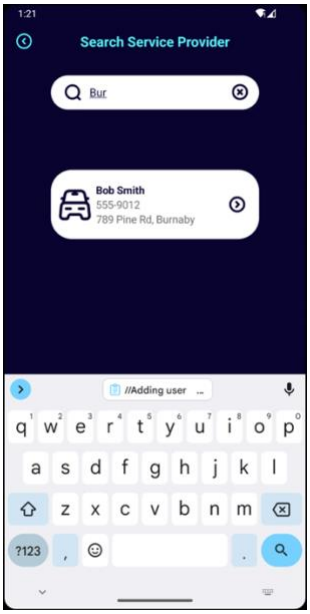
```
String query = "SELECT s.service_id, st.type, s.cost FROM SERVICE s INNER JOIN
SERVICETYPE st ON s.service_type_id = ST.service_type_id WHERE s.provider_id = "
+providerId + " AND s.service_status != 'PASSIVE'";
```

- `dbHelper.addAppointment(appointment);`

```
public void addAppointment(Appointment appointment){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(AP_COLUMN_UID, appointment.getUserId());
    cv.put(AP_COLUMN_PROVIDER_ID, appointment.getProviderId());
    cv.put(AP_COLUMN_SERVICE_ID, appointment.getServiceId());
    cv.put(AP_COLUMN_ADATE, appointment.getDate().toString());
```

```
    cv.put(AP_COLUMN_ATIME, appointment.getTime().toString());
    cv.put(AP_COLUMN_COMMENTS, appointment.getComments());
    cv.put(AP_COLUMN_TYPE, appointment.getType());
    cv.put(AP_COLUMN_STATUS, "PENDING");
    long result = db.insert(TABLE_APPOINTMENT, null, cv);
    if(result == -1){
        Toast.makeText(context, "Unexpected error in adding appointment.",
Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(context, "Appointment has been added successfully.",
Toast.LENGTH_SHORT).show();
    }
}
```

Search Service Provider

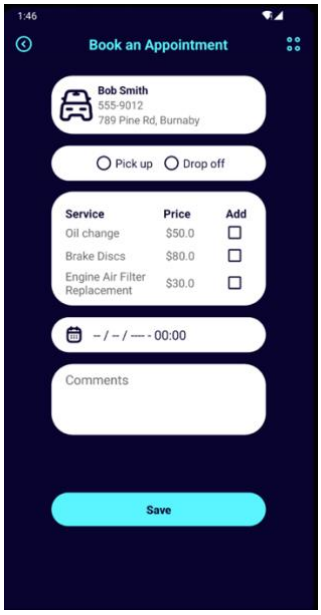


Contributors: [redacted]

Users can search for Service Providers by entering their Addresses or Cities using the Search Providers screen. They can pick any provider display by the search and create an appointment.

How was created:

- Creating Search Provider page layout and coding activity
  - view/ SearchServiceProvider.java
  - layout/activity_search _provider.xml
  - layout/ recyclerview_search_provider.xml
- Creating Search Provider Adapter
  - adapter/SearchProviderAdapter.java



The Search Provider screen includes a SearchView component. The the EditText object is used to retrieve the search text entered by the customer. The EditText object is accessed using the searchProvider text within the SearchView.

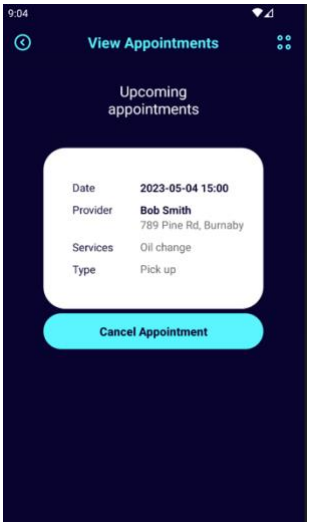
To enable the search, a SearchView component is created, and a listener (OnQueryTextListener) is defined to retrieve the user's search text. In the listener, when the user makes a new search (onQueryTextChange), the **search()** method is called to filter the search results and display them on the RecyclerView.

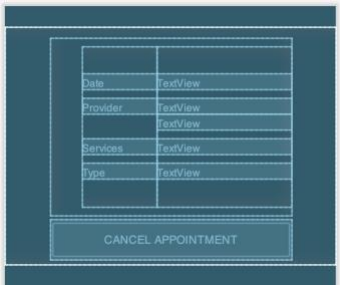
In the **onResume()** method, the **SearchProviderAdapter** is reinitialized with an empty list of users and set to the RecyclerView. Then, the searchView is cleared, focus is removed from it, the **providers** list is cleared and repopulated with all providers from the

database using the **dbHelper.getAllProviders()** method. This ensures that the RecyclerView always displays the latest list of providers from the database, and any previous search query or filter is cleared so that the user can start a new search or view the complete list.

View Appointments for Vehicle Owner

Contributors: [redacted]





recyclerview_viewappointments_customer

The View Appointments page has two different layout for different type of users. When the View Appointments page is opened by the vehicle owner, it displays the upcoming appointments that belong to the vehicle owner. The upcoming appointments are fetched from the DatabaseHelper class based on the date.
The upcoming appointments are displayed in a recyclerview and each appointment has a cancel button. When the user clicks on the cancel button, the application verifies it and calls the cancelAppointment method in the DatabaseHelper class to cancel the appointment.

How was created:

- Creating View Appointments page layout and coding activity
  - view/ ViewAppointments.java
  - layout/activity_view_appointments.xml
  - layout/ recyclerview_viewappointments_customer.xml
- Creating Provider Services Adapter
  - adapter/ViewAppointmentsCustomerAdapter.java

```
appointments = dbHelper.
getUpcomingAppointmentForCustomer(user.getUserId());
```

The userId parameter in the getUpcomingAppointmentForCustomer method is obtained from the AppManager class, which holds the logged-in user.

After the appointments list is fetched from the database and initialized, it is sent to the ViewAppointmentsCustomerAdapter class and associated with the adapter. The adapter class contains an inner class called ViewHolder, which contains the TextViews and button for the corresponding RecyclerView. Once the relevant fields are matched and created within the adapter, they are displayed in the RecyclerView in the ViewAppointments screen.

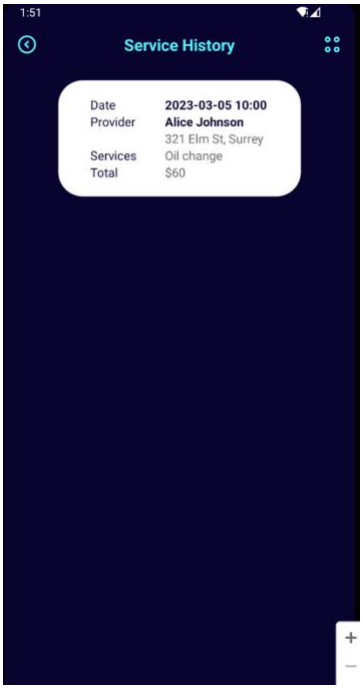
As in other adapter classes, the ItemClickListener interface is also defined in this adapter class, and the onClick method belonging to this interface enables the cancellation of the appointment. Since the ViewAppointments class exhibits different behaviors for two different types of users, the class implements two interfaces in its definition.

```
public class ViewAppointments extends AppCompatActivity implements
ViewAppointmentsProviderAdapter.ItemClickListener,
ViewAppointmentsCustomerAdapter.ItemClickListener
```

Later in the document, it is mentioned how the View Appointments screen behaves for a Provider user.

Service History

Contributors: [redacted]



The service history is the list of completed services a user has paid. It can be accessed directly through the **Navbar menu.**

How was created:

- An additional XML file is required to represent each provider card of the list.
- The **activity_service_history.xml** holds the RecyclerView that will display the list of services.
- The base is a LinearLayout with a custom drawable background with rounder borders.
- A second LinearLayout is used to hold all the fields of the box.
- Another LinearLayout is used to align a label and a TextView to fetch the data.

Functions:

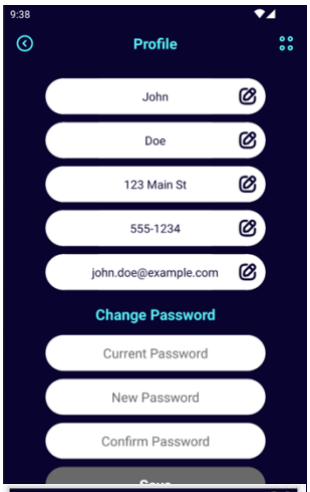
A **ServiceHistoryAdapter** is used to fetch all the list of services from the database with a COMPLETED status.

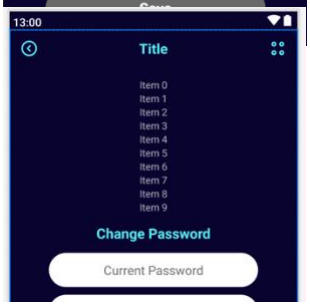
ServiceHistory.java

```
} else if (user.isCustomer()) {
    dbHelper = new DatabaseHelper(getApplicationContext());
    try {
        appointments = dbHelper.getServiceHistoryForUser(user.getUserId());
    } catch (ParseException e) {
        e.printStackTrace();
    }
    RecyclerView recyclerViewApp =
findViewById(R.id.recyclerServiceHistory);
    recyclerViewApp.setLayoutManager(new GridLayoutManager(this, 1));
    serviceHistoryCustomerAdapter = new ServiceHistoryCustomerAdapter(this,
appointments, this);
    recyclerViewApp.setAdapter(serviceHistoryCustomerAdapter);
}
```

Profile

Contributors: [redacted]





Profile screen allows users to update their own information and change their passwords. To edit the relevant field, the user clicks on the edit icon on the right side, and the field becomes editable. After entering the new information, the user clicks the Save button. The Save button is initially inactive, but it becomes active when the user makes a change in any field.

How was created:

- Creating Profile page layout and coding activity
  - view/ Profile.java
  - layout/activity_profile.xml
  - layout/ edit_field_with_icon.xml
- Creating Profile Adapter
  - adapter/ProfileAdapter.java

The area where the user's information is displayed and can be edited in an editable form is created using a recyclerview named **edit_field_with_icon.**

Since the user's information is stored in the **AppManager.instance.user** variable, the adapter class receives this user.

In addition, the Save button is also inactive initially for the change password process. However, the **addTextChangedListener** added to the Confirm Password EditText for passwords checks whether the other fields are filled or not after any input is entered in that field, and makes the button active.

```
edtConfPass.addTextChangedListener(new TextWatcher() {
    @Override
    public void afterTextChanged(Editable s) {
        checkRequiredFields();
    }
});
private void checkRequiredFields() {
    boolean allFieldsFilled = true;
    if (TextUtils.isEmpty(edtCurrPass.getText())) { allFieldsFilled = false; }
    if (TextUtils.isEmpty(edtNewPass.getText())) { allFieldsFilled = false; }
    if (TextUtils.isEmpty(edtConfPass.getText())) { allFieldsFilled = false; }
    if (allFieldsFilled) {
        buttonSave.setEnabled(true);
        Profile.buttonSave.setBackgroundResource(R.drawable.button_enabled);
        int buttonTextColor = ContextCompat.getColor(getApplicationContext(),
R.color.fixer_black);
        Profile.buttonSave.setTextColor(buttonTextColor);
    } else { buttonSave.setEnabled(false); }
}
```

Update user information:

The **ProfileAdapter** class has a final variable called FIELD_COUNT, which holds the number of lines in the profile page. It is defined as 5 because the Profile page has fields for first name, last name, address, phone, and email.

When the user makes changes and clicks the Save button, a String[] fields array is defined to hold the contents of the fields. To access the content of each field, the **findViewHolderForAdapterPosition()** method is used to reach its content. The values in each field are transferred to the fields array.

```
String[] fields = new String[ProfileAdapter.FIELD_COUNT];
boolean isValid = true;
for (int i = 0; i < ProfileAdapter.FIELD_COUNT; i++) {
  //findViewHolderForAdapterPosition() is a method of the RecyclerView class
  // that is used to find the ViewHolder for a given position in the adapter.
  EditText editText =
recyclerViewEditFields.findViewHolderForAdapterPosition(i).itemView.findViewById(R.id.editField);
  if(TextUtils.isEmpty(editText.getText().toString())){
    editText.setError("Invalid value");
    isValid = false;
    break;
  } else {
    fields[i] = editText.getText().toString();
    isValid = true;
  } }
```

If there are no empty values, the user is updated in the database. Since the user's email address cannot be changed, it is not sent in the database query.

```
String firstName = fields[0];
 String lastName = fields[1];
 String address = fields[2];
 String phone = fields[3];
 User updatedUser = new User(firstName,lastName, address, phone, user.getEmail());
 long result = dbHelper.updateUserInfo(updatedUser);
```

DatabaseHelper.java

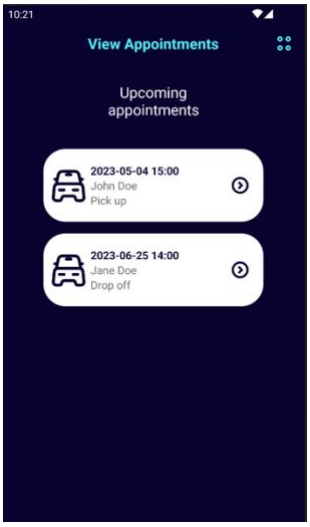
```
public long updateUserInfo(User user){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();
    cv.put(U_COLUMN_FNAME, user.getFirstName());
    cv.put(U_COLUMN_LNAME, user.getLastName());
    cv.put(U_COLUMN_ADDRESS, user.getAddress());
    cv.put(U_COLUMN_PHONE, user.getPhoneNumber());
    String selection = U_COLUMN_EMAIL + " = ?";
    String[] selectionArgs = { String.valueOf(user.getEmail()) };
    long result = db.update(TABLE_USERS, cv, selection, selectionArgs);
    return result;
}
```

If the password update fields are also filled, then the **updateUserPassword** method in the **DatabaseHelper** is called in addition to updating the user information in the database.

```
public long updateUserPassword(User user)
```

View Appointments for Service Provider

Contributors: [redacted]



The **View Appointments** page has two different layouts for different type of users. When the View Appointments page is opened by the service provider, it displays the upcoming appointments that belong to the service provider. The upcoming appointments are fetched from the DatabaseHelper class based on the date.
The upcoming appointments are displayed in a recycler view and each appointment is clickable. When the user clicks on the related widget, the application shows the application details with sending inside the intent for the **AppointmentDetailServiceProvider** activity.

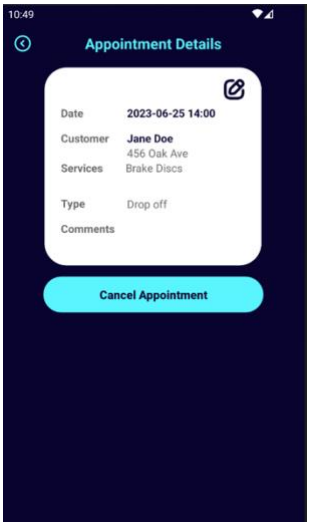
How was created:

- Creating View Appointments page layout and coding activity
  - view/ ViewAppointments.java
  - layout/activity_view_appointments.xml
  - layout/ recyclerview_viewappointments_provider.xml
- Creating AppointmentDetailServiceProvider page layout and coding activity
  - view/ AppointmentDetailServiceProvider.java
  - layout/ activity_appointment_detail_service_provider.xml
- Creating View Appointments Provider Adapter
  - adapter/ViewAppointmentsProviderAdapter.java

The process of associating the upcoming appointments with the recycler view is similar to what was done in the previous pages.

AppointmentDetailServiceProvider

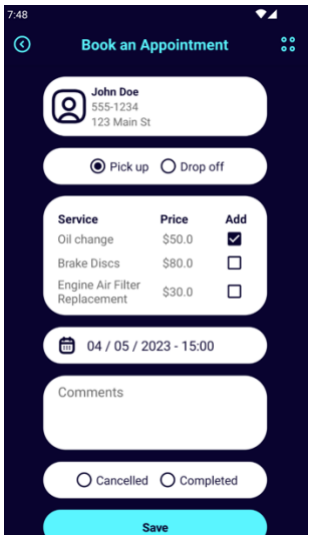
Contributors: [redacted]



This activity use constraint layout, and the appointment information comes with previous intent. Service Provider can cancel the appointment with **Cancel Appointment** button, or they can edit appointment details with clicking the edit icon on the top corner.

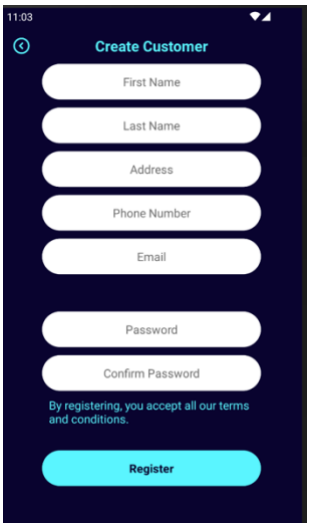
After clicking the edit button it starts the **Book Appointment** activity with intent extras. It takes the appointment details and fills related parts on the Book Appointment screen. The Service Provider can change related fields.

When the Service Provider click Book an Appointment, it update the previous appointment status as CANCELLED, and add a new one.



Create Customer

Contributors: [redacted]



The Create Customer screen uses the **Registiration** screen. In order to control its opening by the Provider, an intent extra named "**WITH_PROVIDER**" is added to the intent in the MainAdapter's onClick event.

MainAdapter.java

```
else if(AppManager.instance.user.isProvider()){
    switch(position1){
        case 0:
            activity.startActivity(new Intent(activity,
ViewAppointments.class)
                    .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
            break;
        case 1:
            Intent intent = new Intent(activity,
RegistrationActivity.class);
            intent.putExtra("WITH_PROVIDER", true);
activity.startActivity(intent
        .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
            break;
```

When the RegistirationActivity is created, it is checked with the **checkIntent** method. If it comes with an intent, the title of the FixerToolbar is changed to "Create Customer".

RegistrationActivity.java

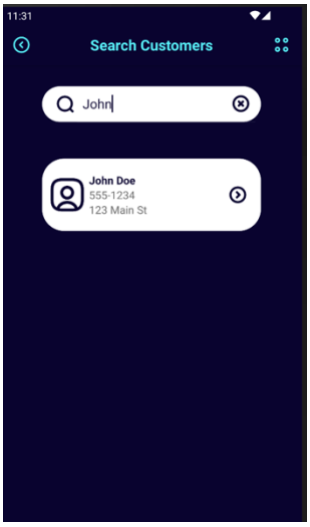
```
private boolean checkIntent(){
    Intent intent = getIntent();
    boolean withProvider = false;
    if(intent != null){
```

```
        withProvider = intent.getBooleanExtra("WITH_PROVIDER", false);
    }
    return withProvider;
}
```

After the necessary information is entered and the Register button is clicked, the intent is checked again. If this process was done with a provider, the activity is closed.

Search Customer

Contributors: [redacted]



Providers can search for customers by entering their names using the Search Customer screen. They can access and edit customers' information or delete them through this page.

How was created:

- Creating Search Customer page layout and coding activity
  - view/ SearchCustomer.java
  - layout/activity_search_customer.xml
  - layout/ recyclerview_searchcustomers_widget.xml
- Creating Search User Adapter
  - adapter/SearchUserAdapter.java

The Search Customer screen includes a SearchView component and the EditText object within the SearchView component is used to retrieve the search text entered by the user. The EditText object is accessed using the search_src_text ID within the SearchView.

```
searchView = findViewById(R.id.searchView);
searchView.clearFocus();
EditText txtSearch =
((EditText)searchView.findViewById(androidx.appcompat.R.id.search_src_text));
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    @Override
    public boolean onQueryTextChange(String newText) {
        filterList(newText);
        return true;
    }
});
```

To enable user search on the Search Customer screen, a SearchView component is created and a listener (OnQueryTextListener) is defined to retrieve the user's search text. In the listener, when the user makes a new search (onQueryTextChange), the filterList() method is called to filter the search results and display them on the RecyclerView.

`userList` is initialized with all customers in the beginning.

```
private void filterList(String text) {
    filteredList.clear();
    if(text.equals("")){
        searchUserAdapter.setFilteredList(filteredList);
    } else {
        for(User user : userList){
            if(user.getFirstName().toLowerCase().contains(text.toLowerCase())){
                filteredList.add(user);
            }
        }
        if(!filteredList.isEmpty()){
            searchUserAdapter.setFilteredList(filteredList);
        }
    }
}
```

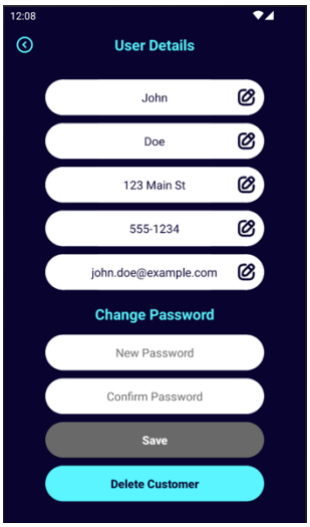
In the **onResume()** method, the **SearchUserAdapter** is reinitialized with an empty list of users and set to the RecyclerView. Then, the searchView is cleared, focus is removed from it, and the **userList** is cleared and repopulated with all customers from the database using the **dbHelper.getAllCustomers()** method. This ensures that the RecyclerView always displays the latest list of customers from the database, and any previous search query or filter is cleared so that the user can start a new search or view the complete list of customers.

```
@Override
protected void onResume() {
    super.onResume();
    searchUserAdapter = new SearchUserAdapter(this, new ArrayList<User>(), this);
    recyclerSearchCustomer.setAdapter(searchUserAdapter);
    searchView.setQuery("", false);
    searchView.clearFocus();
    userList.clear();
    userList = dbHelper.getAllCustomers();
}
```

onItemClick() is overridden method from the **SearchUserAdapter** class, which is used to handle item clicks on the RecyclerView. When the provider clicks on a user in the list, this method is triggered. If the **filteredList** is not empty and has a position that matches the clicked item, an intent is created to start the **CustomerDetail** activity, passing the **USER_ID** of the selected user as an extra.

Customer Detail

Contributors: [redacted]



The CustomerDetail screen has its own layout, but the content and adapter of the RecyclerView are the same as the **Profile** activity. The difference on this screen is that since the service provider doesn't know the user's current password, only the password update fields are available. Additionally, the service provider can delete the user from this screen.

When provider clicks Create Customer button, it shows a dialog box for confirmation, after confirmation the user is deleted with DatabaseHelper class method.

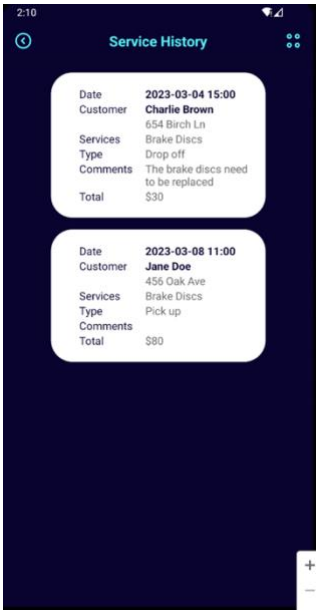
```
dbHelper.deleteUser(customer);
```

The deleteUser method first deletes the appointments that are related to the user, and then deletes the user itself.

```
public void deleteUser(User user){
    SQLiteDatabase db = this.getWritableDatabase();
    long result1 = db.delete(TABLE_APPOINTMENT, AP_COLUMN_UID + " = ?", new
String[]{String.valueOf(user.getUserId())});
    long result2 = db.delete(TABLE_USERS, U_COLUMN_ID + " = ?", new
String[]{String.valueOf(user.getUserId())});
    System.out.println(result1 + "-" + result2);
}
```

Service History

Contributors: [redacted]



The service history is the list of completed services a user has paid. It can be accessed directly through the **Navbar menu.**

How was created:

- An additional XML file is required to represent each provider card of the list.
- The **activity_service_history.xml** holds the RecyclerView that will display the list of services.
- The base is a LinearLayout with a custom drawable background with rounder borders.
- A second LinearLayout is used to hold all the fields of the box.
- Another LinearLayout is used to align a label and a TextView to fetch the data.

Functions:

A **ServiceHistoryAdapter** is used to fetch all the list of services from the database with a COMPLETED status. The same class is used to display both ServiceHistory for the customer and the provider with a slightly difference in the displayed data.

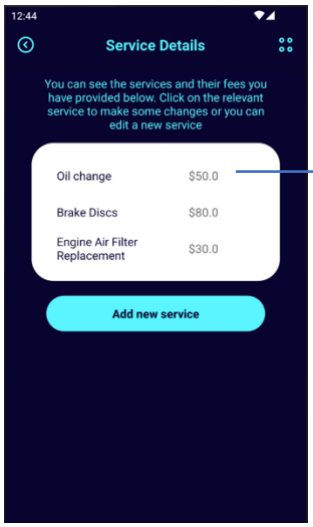
ServiceHistory.java

```
if (user.isProvider()) {
    dbHelper = new DatabaseHelper(getApplicationContext());
    try {
        appointments =
dbHelper.getServiceHistoryForProvider(user.getUserId());
    } catch (ParseException e) {
        e.printStackTrace();
    }
    RecyclerView recyclerViewApp =
findViewById(R.id.recyclerServiceHistory);
    recyclerViewApp.setLayoutManager(new GridLayoutManager(this, 1));
    serviceHistoryProviderAdapter = new ServiceHistoryProviderAdapter(this,
appointments, this);
    recyclerViewApp.setAdapter(serviceHistoryProviderAdapter);
```

Provider Services

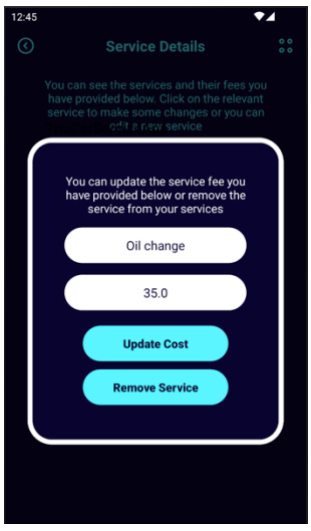
Contributors: [redacted]

Service providers can update the prices of the services they offer, decide to stop providing a service, or add a new service to their offerings. Services and their costs are shown by RecyclerView and an adapter.

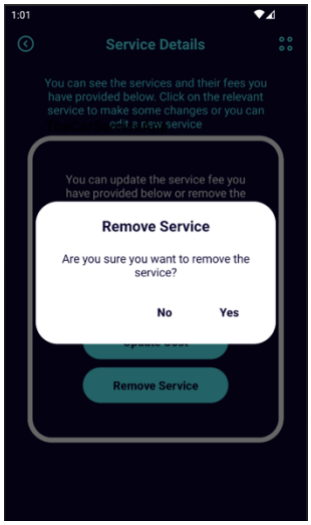


Every field in the recycler view has onItemClick Listener and it passes the service information to the PopupActivity.

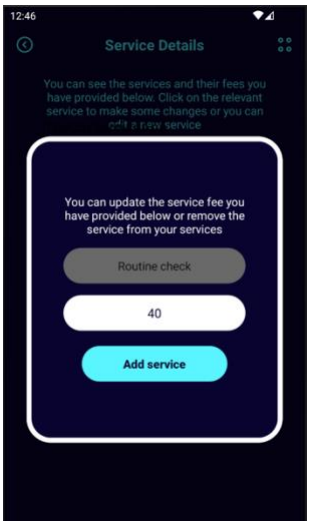
```
@Override
public void onItemClick(View view, int
position) {
    Intent intent = new
Intent(ProviderServices.this,
PopupActivity.class);
    intent.putExtra("SERVICE_ID",
serviceList.get(position).getServiceId());
  intent.putExtra("SERVICE_TYPE",
serviceList.get(position).getType());
    intent.putExtra("SERVICE_COST", String.
valueOf(serviceList.get(position).getCost()));
    startActivity(intent);
}
```



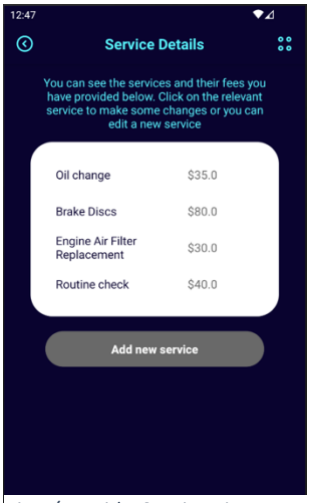
Providers can update the costs of their services, remove existing services or add new services to their offerings. In the "**Add Service**" screen, a spinner is used to list the services that the provider hasn't offered yet. Once the provider has added all their services, the "**Add New Service**" button becomes inactive.



view/PopupActivity.java
layout/activity_popup.xml



view/PopupAddActivity.java
layout/activity_popup_add.xml



view/ProviderServices.java
layout/activity_provider_services.xml
layout/recyclerview_service_details.xml
view/ProviderServicesAdapter.java

List of other components and code used.

- Custom Dialog box
- Custom notification Toast
- ItemMenu class for the Navbar Menu
- Custom icons
- Custom Navigation Drawer