Contents

Introduction

This document details the design, implementation and evaluation of my created android application, aiming to provide a thorough understanding of the application and its development.

This document is will go through the applications background, the legal, social, ethical and professional consideration which were taken into account, the design choices and the reasons for them, the structure of how the apps features were implemented, how these features were tested to ensure functionality and finally an evaluation of the application.

Github Repository Link

Below is a link to the repository containing the android studio applications source code and a README file crediting my used assets.

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt.git

YouTube Video Link

Bellow is a link to a video displaying the application and all its feature running.

https://youtu.be/FPzQWHFugqo

Background

The application I created intends to streamline the interaction between the customer and the restaurant by providing multiple different services to the user, with its primary goal being to satisfy the product vision and functional requirements given.

The benefit of this app is that the user can quickly and easily make a reservation at the restaurant, being able to see whether a table is available for the users specified date, time and number of people. This is not the only functionality the application provides as the user can see the menu, find out the restaurant's location, see their reservations details with the ability to cancel or edit them, leave a review and read other users reviews and finally set their favourite meal and table.

The user of this app can be of any demographic looking to eat at the restaurant and therefore the application needed to be accessible and to cater to all. The primary goal for the user would be to make a reservation and to do this I created a user-friendly booking page in which the app connects to an API which holds everyone's reservation information and finds out which tables are available at the users specified date, time and the number of people for the table. Then the user can easily choose a table and finish the reservation.

Legal, Social, Ethical and Professional (LSEP)

Legal

The application itself only holds the data for the currently logged in user's username and phone number as well as the users favourite table and meal, while all other data is stored on an API (For the purpose of this app the reviews, users, menu and table details are stored in JSON files in internal storage. However, this is only to symbolise that the data is to be stored on another API and won't be stored locally.). As the user's data regarding reviews and reservations aren't stored locally the risk is minimized.

When the user signs out or deletes their account, the stored data is immediately deleted which complies with the required rights of the user in the Data Protection Act (Data Protection Act 2018). Confirmation of account deletion is also implemented as the user's data is immediately deleted and cannot be recovered.

Social

The application was designed to consider the needs and requirements of all users. To aid in this accessibility features were implemented to couple with androids built in accessibility mode and its UI was designed to be clear and consistent.

Ethical

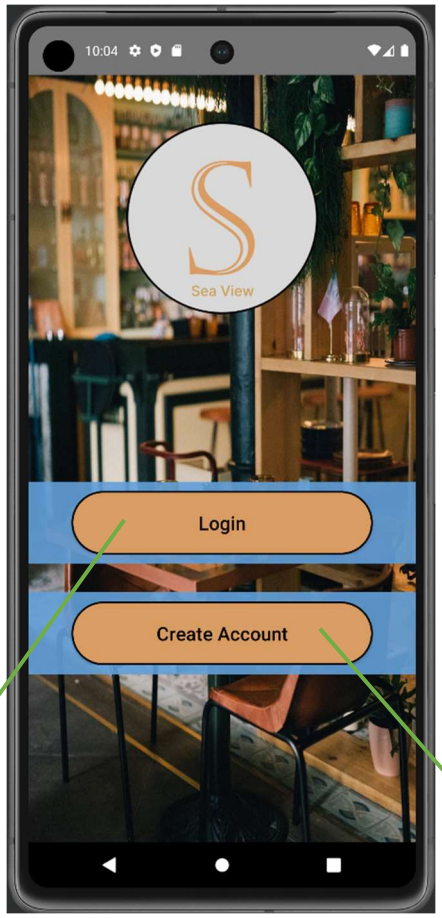
In regard to the ethics the user's data is only used for its purpose the user intended it to be and, as mentioned above, the user has full control over their stored data.

Professional

Full credits to all 3rd party assets and libraries used are provided in the github README file.

Design

App Open (Main) Activity

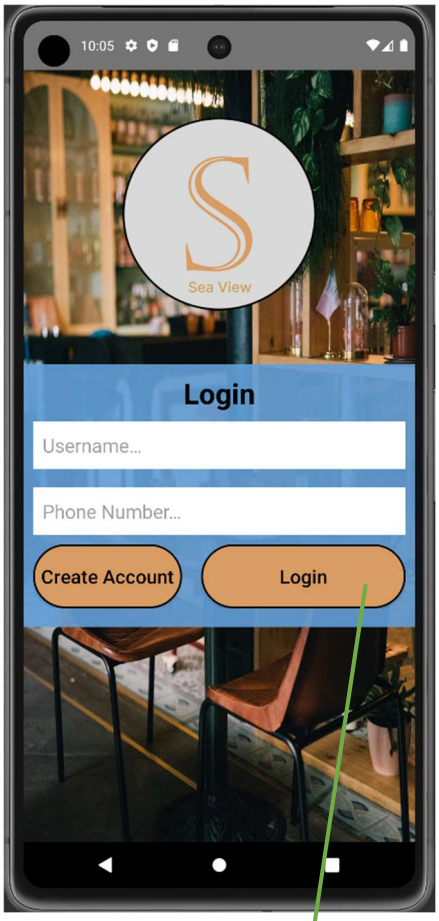


Key

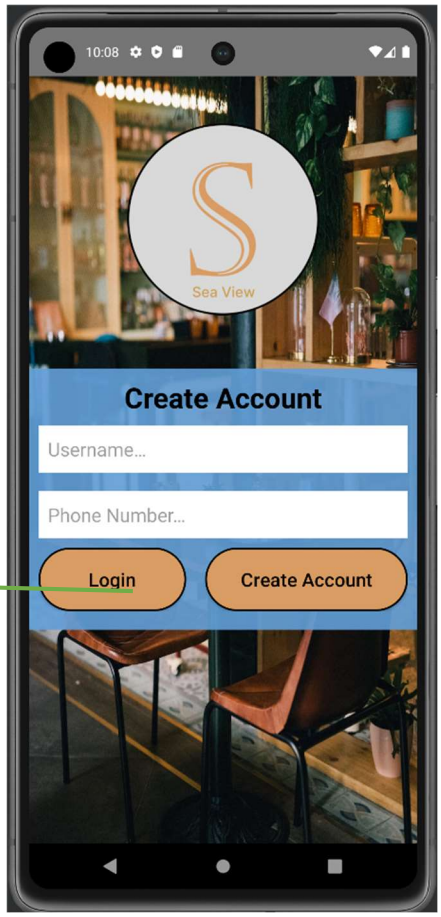
Green arrow is navigation to an activity

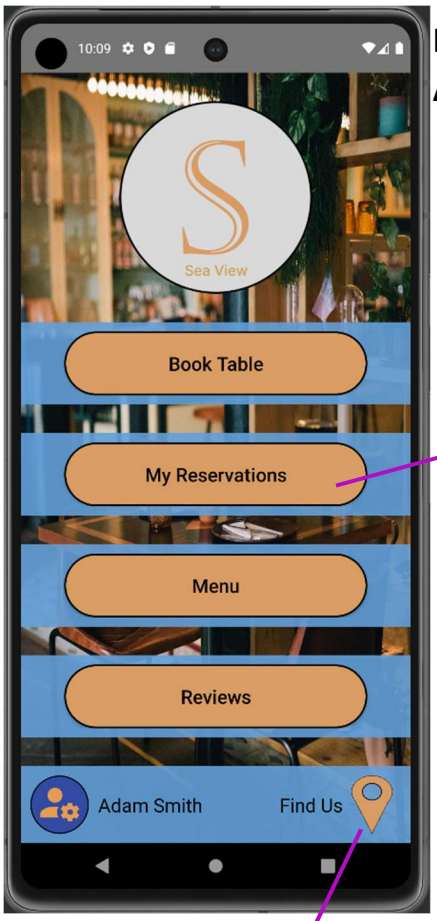
Purple arrow is showing fragment

Login Activity



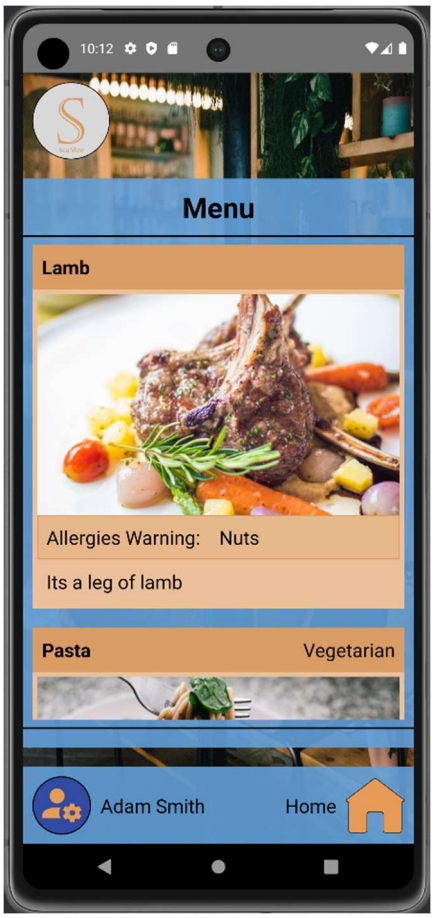
Create Account Activity

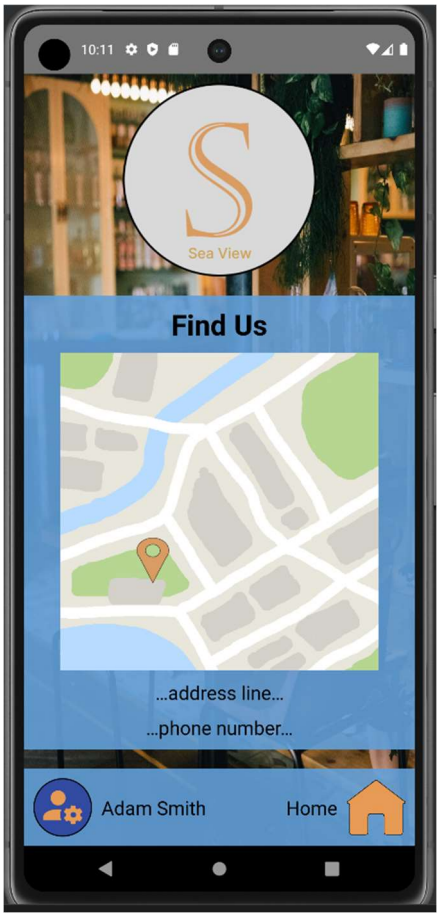




Home Activity

Menu Fragment





Find Us Fragment

The home activity is the main navigation hub.

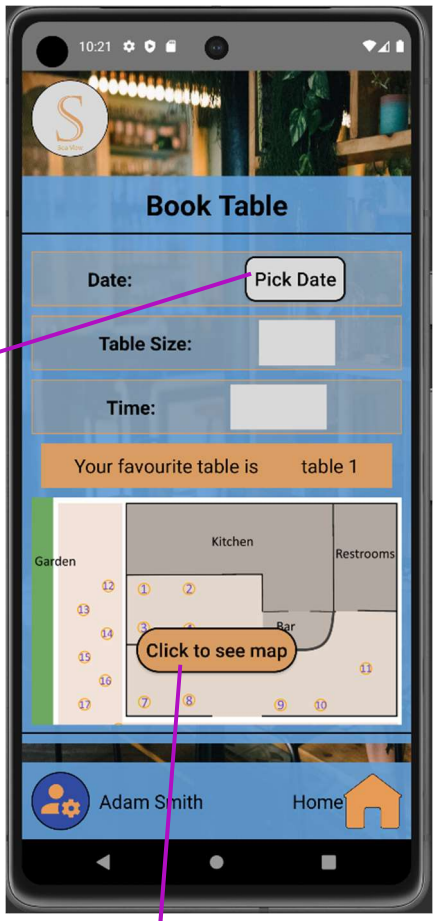
Bottom bar is consistent

Goes to Account Settings Activity

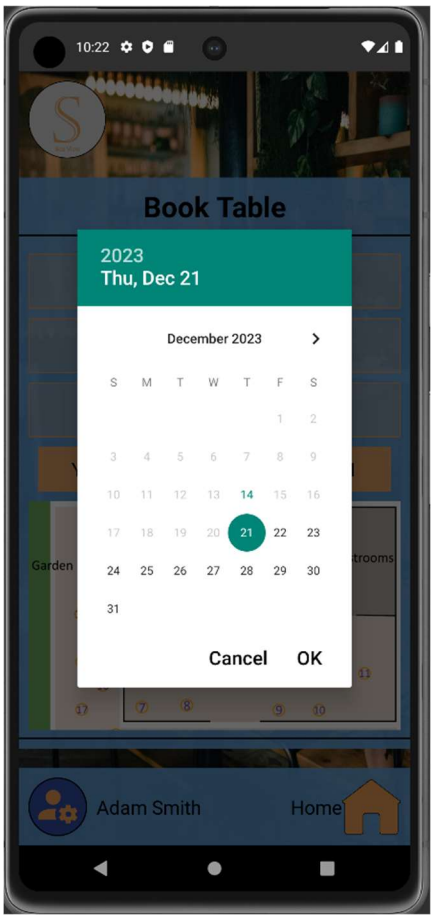
Goes to Home Activity

Home Activity

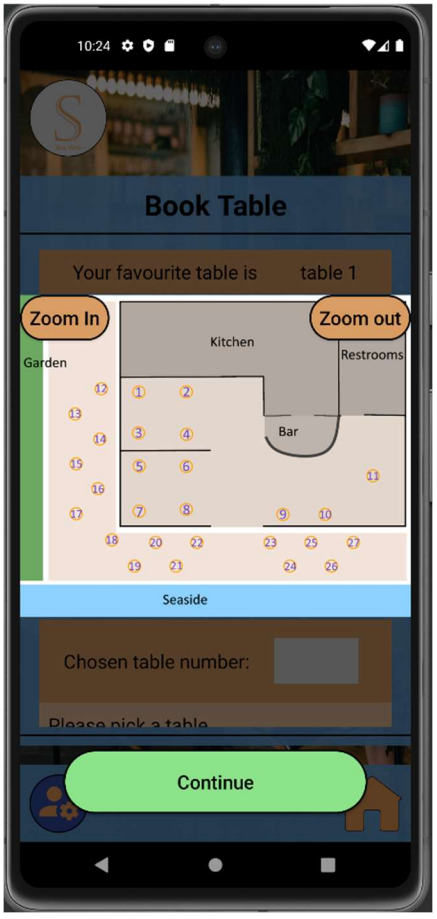
Book Table Activity



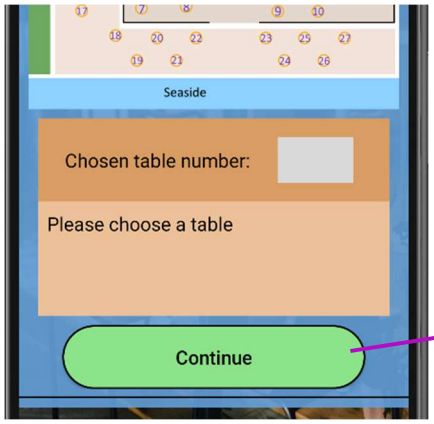
Calendar Dialog Fragment



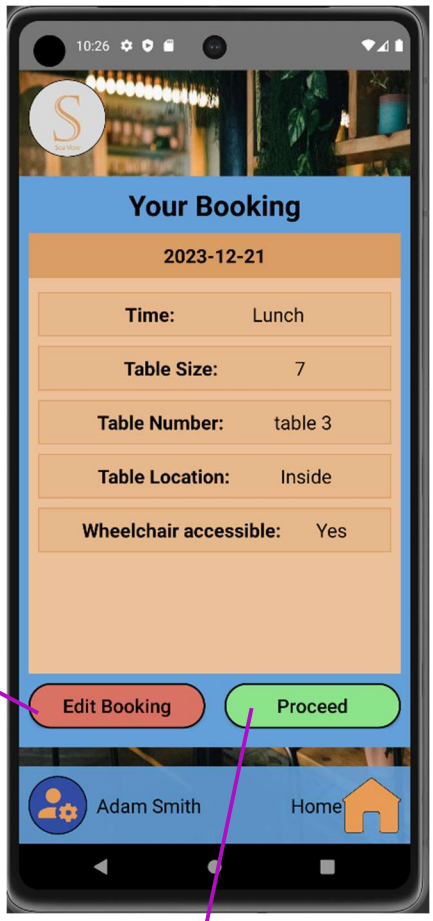
Fullscreen Map Dialog Fragment



Scrolled down booking activity



Booking Details Fragment

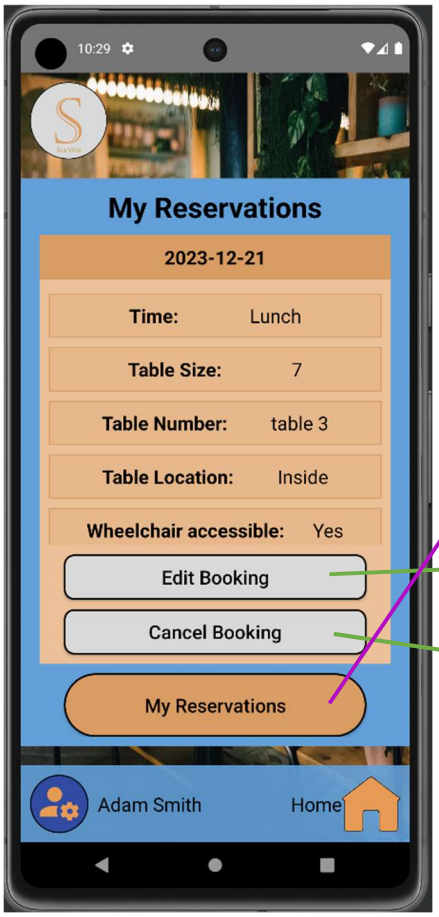


Goes to Home Activity

Home Activity

My Reservations Activity

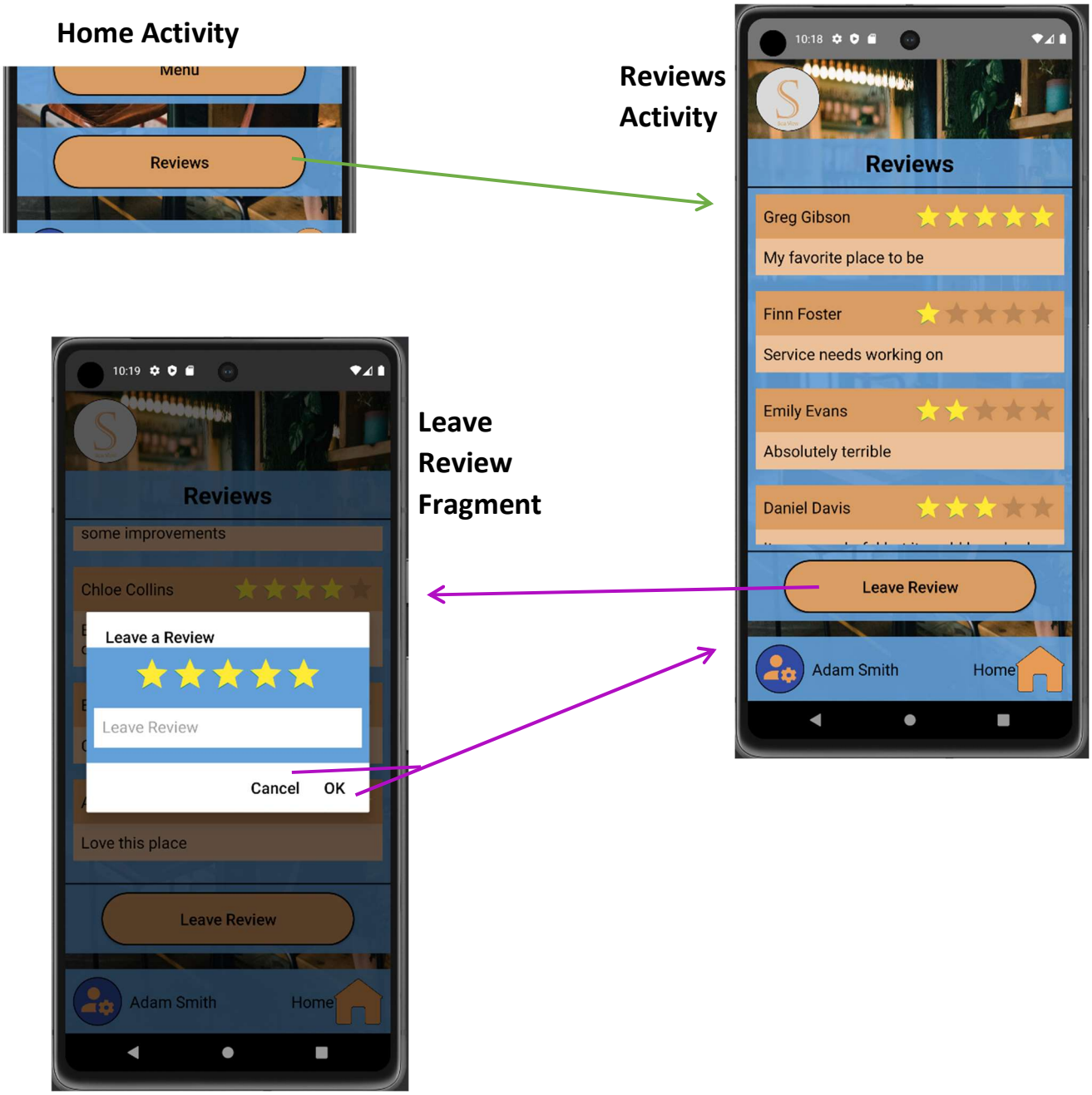
Reservations Details Fragment



Goes to Bookings Activity

Goes to Home Activity

The details fragment changes depending on which reservation the user chose.



The fragments for the activity are initialised on create but stay hidden until requested, at that time they show and are overlayed on top of the desired area. For all but the home activities fragment the fragment only overlays the main content meaning the bottom bar and logo don't have to be created again.

Implementation

Fragments

To create the fragments, I check to see if it already exists and if it doesn't, I create a new instance of the fragment. This instance will be hidden to start and I use the show method to bring it to the front.

```
private FragmentManager fragmentManager = getSupportFragmentManager();
if (fragmentManager.findFragmentByTag(MENU_FRAGMENT) != null){
    //fragment exists
    fragmentManager.beginTransaction().show(fragmentManager.findFragmentByTag(MENU_FRAGMENT)).commit();
} else{
    //fragment doesn't exist so add it to the fragment manager
    fragmentManager.beginTransaction().add(R.id.main_container, new MenuFragment(), MENU_FRAGMENT).addToBackStack( name: null).commit();
}

//if other fragments are visible hide them
if (fragmentManager.findFragmentByTag(FIND_US_FRAGMENT) != null){
    fragmentManager.beginTransaction().hide(fragmentManager.findFragmentByTag(FIND_US_FRAGMENT)).commit();
}
```

Navigation

To change between pages, I used the methods bellow depending on my current and future location.

Going to page (Activity to Activity)

```
private void openCreateAccountPage() {
    Intent intent = new Intent( packageContext: this, CreateAccountActivity.class);
    startActivity(intent);
}
```

Going to page (Fragment to Activity)

When going from a fragment to an activity I can use the same method as above, however if I want to go back to the same instance of the activity, I use a different method.

```
private void navigateToHomePage() {
    FragmentManager fragmentManager = requireActivity().getSupportFragmentManager();
    fragmentManager.popBackStack();
}
```

Going to page (Activity to Fragment)

The show method overlays the fragment on top of the activity.

```
fragmentManager.beginTransaction().show(fragmentManager.findFragmentByTag(MENU_FRAGMENT)).commit();
```

While the replace method is usually used for displaying fragments, I chose to use the show method for 2 reasons.

1: It doesn't need to close the content which is being replaced. This is to lower resource demand caused by the app the content won't need to be loaded, closed and reloaded constantly.

2: As the content doesn't disappear all user inputs are kept and therefore don't need to be immediately saved when loading a new fragment.

Using ".show" lead me to choose my activity and fragment structure. I grouped the activities and fragments together based on how the user would interact with the app to create an optimised structure.

Current User

The app knows the current user by storing the information in shared preferences.

```
//save current user details
SharedPreferences.Editor editor = getSharedPreferences( name: "CurrentUser", MODE_PRIVATE).edit();
editor.putString( s: "userId", userId);
editor.putString( s: "username", username);
editor.putString( s: "phoneNum", phoneNum);
editor.apply();
```

This can then be accessed anywhere to find out the users' details.

```
//display current user
TextView textViewUsername = findViewById(R.id.username);
SharedPreferences user = getSharedPreferences( name: "CurrentUser", MODE_PRIVATE);
textViewUsername.setText(user.getString( s: "username", s1: ""));
```

Data Storage

The users, reviews, menu and tables are initialised in the DataInitializer class.

```
//put test data into the internal storage
DataInitializer.initializeData( context: this);
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/helper/DataInitializer.java

Her JSON files are put into the internal storage. This is to simulate a remote storage where the app will access, for example an API to get the data.

```
{
  "users": {
    "user 1": {
      "username": "Adam Smith",
      "phoneNum": "07777777777"
    },

    "user 2": {
      "username": "Beth Baron",
      "phoneNum": "07666666666"
    }
  }
}
```

Above are the users JSON file which stores the all the accounts and bellow is the reviews file which stores all of the restaurant reviews.

The Menu and Tables files are special as they cannot be updated or changed by the user and can only be changed outside off the app. When an employee then updates the menu or table all of the apps can access that change immediately.

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/assets/meals.json

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/assets/tables.json

```
{
  "tables": {

    "table 1": {
      "size":8,
      "accessible": "Yes",
      "location": "Inside"
    },

    "table 2": {
      "size":4,
      "accessible": "Yes",
      "location": "Inside"
    },
```

Worker Thread

Throughout the application I have used worker threads to handle both API connections and potentially lengthy tasks.

```
//finds the tables which are available for the users inputs
public void setValidTables(String dateChosen, String tableSizeChosen, String mealTimeChosen){
    //run tasks on another thread to lighten load on main
    executor = Executors.newSingleThreadExecutor();
    Handler handler = new Handler(Looper.getMainLooper());

    executor.execute(() -> {

        //check API for taken tables
        List<String> takenTables = getTakenTables(dateChosen, mealTimeChosen);

        //check JSON file table for valid sized tables
        List<String> validTablesForSize = findValidTablesForSize(tableSizeChosen);

        //remove taken tables from valid size tables from json
        assert validTablesForSize != null;
        assert takenTables != null;
        validTablesForSize.removeAll(takenTables);

        //reformat array with "" as array index 0 (so the default chosen spinner is "")
        String[] finalValidTables = new String[validTablesForSize.size() + 1];
        finalValidTables[0] = "";
        for (int i = 1; i < validTablesForSize.size() + 1; i++) {
            finalValidTables[i] = validTablesForSize.get(i - 1);
        }

        //update spinner values to the free tables
        handler.post(() -> spinnerHelperChosenTable.updateSpinnerData(finalValidTables));
    });
}
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookTableActivity.java

I opted to use executors and handlers as this allowed for asynchronous threading while giving me control of what gets processed in the worker thread and what goes to the main UI thread. The example above gets the users reservation details and finds out what tables fit the given criteria and whether they have already been booked.

API connection

For connecting to the API, I used volley requests and encapsulated them inside their own worker thread so the data can be processed being sent to the UI.

The example bellow is used to delete the user's reservation.

```
private void deleteBooking(){
    ExecutorService executor = Executors.newSingleThreadExecutor();
    Handler handler = new Handler(Looper.getMainLooper());

    executor.execute(() -> {

        String deleteUrl = API_URL + "/" + reservationId;

        RequestQueue queue = Volley.newRequestQueue(requireContext());
        JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.DELETE, deleteUrl,  jsonRequest: null,
                response -> {
                    //send notification
                    handler.post(() -> {
                        NotificationHelper notificationHelper = new NotificationHelper();
                        notificationHelper.notifyUpdatedBooking(requireContext().getApplicationContext());
                    });

                },
                error -> {
                    //error occurred to go back
                    handler.post(() -> {
                        Toast.makeText(requireContext(),  text: "Couldn't update reservation. Please try again.", Toast.LENGTH_LONG).show();
                        goBackToHomePage();
                    });
                }
        );
        queue.add(jsonArrayRequest);
    });
}
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookingDetailsFragment.java

Notifications

The notification helper is used to handle all of the notifications and contains default notifications which can be called from anywhere and gets initialised along when the app opens.

```java
public class NotificationHelper {

    private static final String ID = "SeaViewNotificationID";

    private static int notificationIdCounter = 0;

    //gives each notification a unique id
    public static int generateUniqueNotificationId() { return notificationIdCounter++; }

    public static void initializeNotificationChannel(Context context) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            String CHANNEL = "SeaViewNotificationChannel";
            NotificationChannel channel = new NotificationChannel(
                    ID,
                    CHANNEL,
                    NotificationManager.IMPORTANCE_DEFAULT
            );
            channel.setDescription("Handles all notifications from the SeaView app.");

            NotificationManager notificationManager = context.getSystemService(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }
    }

    private void createNotification(Context context, String title, String content) {
        if (ActivityCompat.checkSelfPermission(context, android.Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
            //notification intent
            Intent intent = new Intent(context, MainActivity.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, requestCode: 0, intent, PendingIntent.FLAG_IMMUTABLE);

            NotificationCompat.Builder builder = new NotificationCompat.Builder(context, ID)
                    .setSmallIcon(R.drawable.app_icon)
                    .setContentTitle(title)
                    .setContentText(content)
                    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
                    //When the user taps the notification
                    .setContentIntent(pendingIntent)
                    .setAutoCancel(true);

            NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
            notificationManager.notify(generateUniqueNotificationId(), builder.build());
        }
    }
```

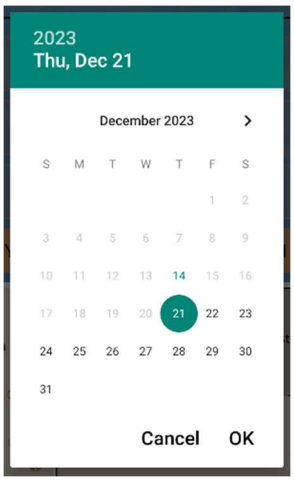
```java
public void notifyBookingSuccess(Context context, String date) {
    createNotification(context, title: "My Reservation", content: "Your reservation for the " + date +" was successful.");
}

public void notifyReviewThanks(Context context) {
    createNotification(context, title: "Review", content: "Thank you for leaving a review.");
}

public void notifyUpdatedBooking(Context context) {
    createNotification(context, title: "My Reservation", content: "You have successfully updated your reservation");
}

public void notifyCancelBooking(Context context,String date) {
    createNotification(context, title: "Cancellation", content: "You have canceled your booking for the "+ date + ". Sorry to see you go.");
}
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/helper/NotificationHelper.java

These are the default notifications.

Booking Calendar

When the user first starts to create a booking, they are given a button to open the calendar.




```
//calender
Button buttonCalender = findViewById(R.id.btn_pick_date);
buttonCalender.setOnClickListener(view -> {
    DatePickerDialogFragment calender = new DatePickerDialogFragment();
    calender.setDateSelectedListener(this);
    calender.show(fragmentManager,  tag: "datePicker");
    checkStartingUserInputs();
});
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookTableActivity.java

When the button gets clicked the calendar dialog appears with the earliest choice of date being 1 week in advance.

```
@NonNull
@Override
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    //get the current date
    final Calendar calendar = Calendar.getInstance();
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    //book a week in advance
    calendar.add(Calendar.DAY_OF_MONTH,  i1: 7);

    DatePickerDialog datePickerDialog = new DatePickerDialog(requireContext(),  listener: this, year, month, day);

    datePickerDialog.getDatePicker().setMinDate(calendar.getTimeInMillis());

    return datePickerDialog;
}

@Override
public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
    //date has been chosen


    //format the date
    Calendar selectedDate = Calendar.getInstance();
    selectedDate.set(year, month, dayOfMonth);
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault());
    String formattedDate = dateFormat.format(selectedDate.getTime());
    dateSelectedListener.onDateSelected(formattedDate);
}
```

Booking Spinners

The next user inputs are spinner which get initialised with an array (Table size: 1-10 and Time: Breakfast, Lunch, Dinner). A “” or empty string is put as index 0 in the array so that the spinner has an empty choice to be set to which ensures that the user has to input a value.

```
//choose size
Spinner chosenTableSize = findViewById(R.id.spn_table_size);
new SpinnerHelper(chosenTableSize, getResources().getStringArray(R.array.table_size),  callback: this);
```

The spinner helper is a class which helps me to create spinners.

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/helper/SpinnerHelper.java

While I could do everything without making the class, I made it as it allowed me to override the onItemSelected method and find which spinner chose which item.

```
@Override
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
    Spinner spinner = (Spinner) adapterView;
    //generic string selected item
    String selectedItem = adapterView.getItemAtPosition(i).toString();
    callback.onItemSelected(spinner, selectedItem);
}
```

Bellow is the onItemSelected method I used in the booking activity. It finds out which spinner was touched by the user and what choice they chose.

```
//for handling spinner selections
@Override
public void onItemSelected(Spinner spinner, String selectedItem) {

    Spinner whichSpinner = findViewById(spinner.getId());
    if (whichSpinner == findViewById(R.id.spn_table_size) || (whichSpinner == findViewById(R.id.spn_time))){
        //the user changed a input
        checkStartingUserInputs();

    } else if (whichSpinner == findViewById(R.id.spn_chosen_table)) {
        //the user chose a table
        try {
            JSONObject allTables = Objects.requireNonNull(FileHelper.readJsonFile( context: this, FileHelper.TABLES_JSON)).getJSONObject( name: "tables");
            JSONObject table = allTables.getJSONObject(selectedItem);
            setTableDetails(table);
        }catch (Exception ignored){

        }
    }
}
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookTableActivity.java

Table Spinner

The final user input is the choice of table number. Based on the previous user inputs the app checks whether the table is suitable (is the table big enough for the number of people) and whether the table has already been booked.

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookTableActivity.java

The spinner choices are then dynamically changed to show the available tables.

```
//finds the tables which are available for the users inputs
public void setValidTables(String dateChosen, String tableSizeChosen, String mealTimeChosen){
    //run tasks on another thread to lighten load on main
    executor = Executors.newSingleThreadExecutor();
    Handler handler = new Handler(Looper.getMainLooper());

    executor.execute(() -> {

        //check API for taken tables
        List<String> takenTables = getTakenTables(dateChosen, mealTimeChosen);

        //check JSON file table for valid sized tables
        List<String> validTablesForSize = findValidTablesForSize(tableSizeChosen);

        //remove taken tables from valid size tables from json
        assert validTablesForSize != null;
        assert takenTables != null;
        validTablesForSize.removeAll(takenTables);

        //reformat array with "" as array index 0 (so the default chosen spinner is "")
        String[] finalValidTables = new String[validTablesForSize.size() + 1];
        finalValidTables[0] = "";
        for (int i = 1; i < validTablesForSize.size() + 1; i++) {
            finalValidTables[i] = validTablesForSize.get(i - 1);
        }

        //update spinner values to the free tables
        handler.post(() -> spinnerHelperChosenTable.updateSpinnerData(finalValidTables));
    });
}
```

```
public void updateSpinnerData(String[] newDataArray) {
    adapter = new ArrayAdapter<>(spinner.getContext(), android.R.layout.simple_spinner_item, newDataArray);
    adapter.setDropDownViewResource(androidx.appcompat.R.layout.support_simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
    adapter.notifyDataSetChanged();
}
```

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/helper/SpinnerHelper.java

I opted to allow the user to choose the specific table number as the product vision wanted the user to be able to pick a favourite table. For that reason, instead of picking a location (inside, outside, garden, seaside) the user can pick a table which is in a specific area. To find out the area I they user can use the full screen map and see the description of the table when it is chosen.

Fullscreen Map Dialog

The map is a dialog which pops up to allow the user to see the table layout and where all the tables are located so that they are informed in their choice of table.

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/BookTableActivity.java

https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/main/app/src/main/java/com/bsw/coursework2000/booking/FullscreenMapDialog.java

```java
//Display and create functionality for the fullscreen map
@SuppressLint("ClickableViewAccessibility")
private void showFullscreenImageDialog() {
    final Dialog dialog = new Dialog( context: this, R.style.FullscreenDialogTheme);
    dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
    dialog.setContentView(R.layout.dialog_fullscreen_map);

    Button btnZoomIn = dialog.findViewById(R.id.btn_zoom_in);
    Button btnZoomOut = dialog.findViewById(R.id.btn_zoom_out);

    ImageView fullscreenImage = dialog.findViewById(R.id.fullscreen_image);
    Button buttonContinue = dialog.findViewById(R.id.btn_continue);

    fullscreenImage.setImageResource(R.drawable.restaurant_map);

    //dragging image on screen
    fullscreenImage.setOnTouchListener((v, event) -> {
        float x = event.getX();
        float y = event.getY();
        float sensitivity = 1.0f;

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                lastX = x;
                lastY = y;
                break;

            case MotionEvent.ACTION_MOVE:
                float deltaX = (x - lastX) * sensitivity;
                float deltaY = (y - lastY) * sensitivity;

                // Move image
                float newX = fullscreenImage.getTranslationX() + deltaX;
                float newY = fullscreenImage.getTranslationY() + deltaY;

                fullscreenImage.setTranslationX(newX);
                fullscreenImage.setTranslationY(newY);

                lastX = x;
                lastY = y;
                break;

            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                break;
        }

        return true;
    });
```

```
    btnZoomIn.setOnClickListener(v -> {
        scale *= 1.2f;
        updateScaleAndPosition(fullscreenImage);
    });

    btnZoomOut.setOnClickListener(v -> {
        scale /= 1.2f;
        updateScaleAndPosition(fullscreenImage);
    });

    buttonContinue.setOnClickListener(v -> dialog.dismiss());

    dialog.show();
}
//change the map scale
private void updateScaleAndPosition(ImageView imageView) {
    //setting max and min image scale
    if (scale > 3.0f) {
        scale = 3.0f;
    } else if (scale < 1.0f) {
        scale = 1.0f;
    }

    imageView.setScaleX(scale);
    imageView.setScaleY(scale);
}
```

The panning and scaling are important parts as even when the map is touching the edge of the screen it can be hard to read, so they are retired to make the app more accessible.

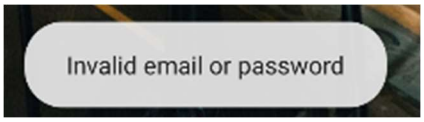
Heuristics

Visibility

To maintain visibility the app creates notifications and Toast messages for the user.

Match

All of the information passed to the user is designed to be easily understandable and familiar.



User Control and Freedom

The bottom bar fulfils this role as the home button is always available to take the user back to the home page. I also implemented the use of the android back button so the user can more freely navigate the application.

Consistency

I have endeavoured to remain consistent throughout the application. An issue I can foresee is how I have many “details” labels and use reservation and booking interchangeably when I should be more consistent.

Error Prevention

In regards to the error handling I have added many checks and conditions to ensure that the user can't cause errors, however I am aware that unexpected problems arrived so where I felt it was encaser I have added try catch blocks and gave the feedback when an error occurs.

Recognition Rather Than Recall

All information has been provided to the user when they need it. The only exception is during booking creation where the user would have to look at the map remember a table number then input it.

Aesthetic

I have done a good job at keeping the aesthetic consistent with the content being contained within the blue area however the background image I chose does introduce some complications into the minimalist design.

Help and Documentation

This is probably my poorest heuristic as I haven't provided the user with documentation and the help, I give the user is only in the form of labels indicating what the user should do.

Testing

Some data has been provided in the JSON files which I have used to test out various inputs and ensure the app if functioning properly. By using the device file manager, I can see how the data is being stored in real-time and this has helped me while debugging my application.

Summery

In summery the application fulfils the initial project vision and requirements, creating an easy user-friendly method to interact with the restraint via the app. Consideration has been given to the users of the application and the social, ethical and legal issues which might arise, and the design is has been thought through to provide a good experience for the user.

Bibliography

Data Protection Act. (2018). c3. *Available at: https://www.legislation.gov.uk/ukpga/2018/12/part/4/chapter/3/crossheading/rights.(Accessed: 12/12/2023)*.

README. (2023). *Available at: https://github.com/Plymouth-University/mainassessment-benjaminsanderswyatt/blob/a485f10f79343f6210c2a06006c6b348d4168c03/README.md*.