CS385 MOBILE APP DEVELOPMENT

P2P Weddings

Team - *Michael Heneghan*

Outline

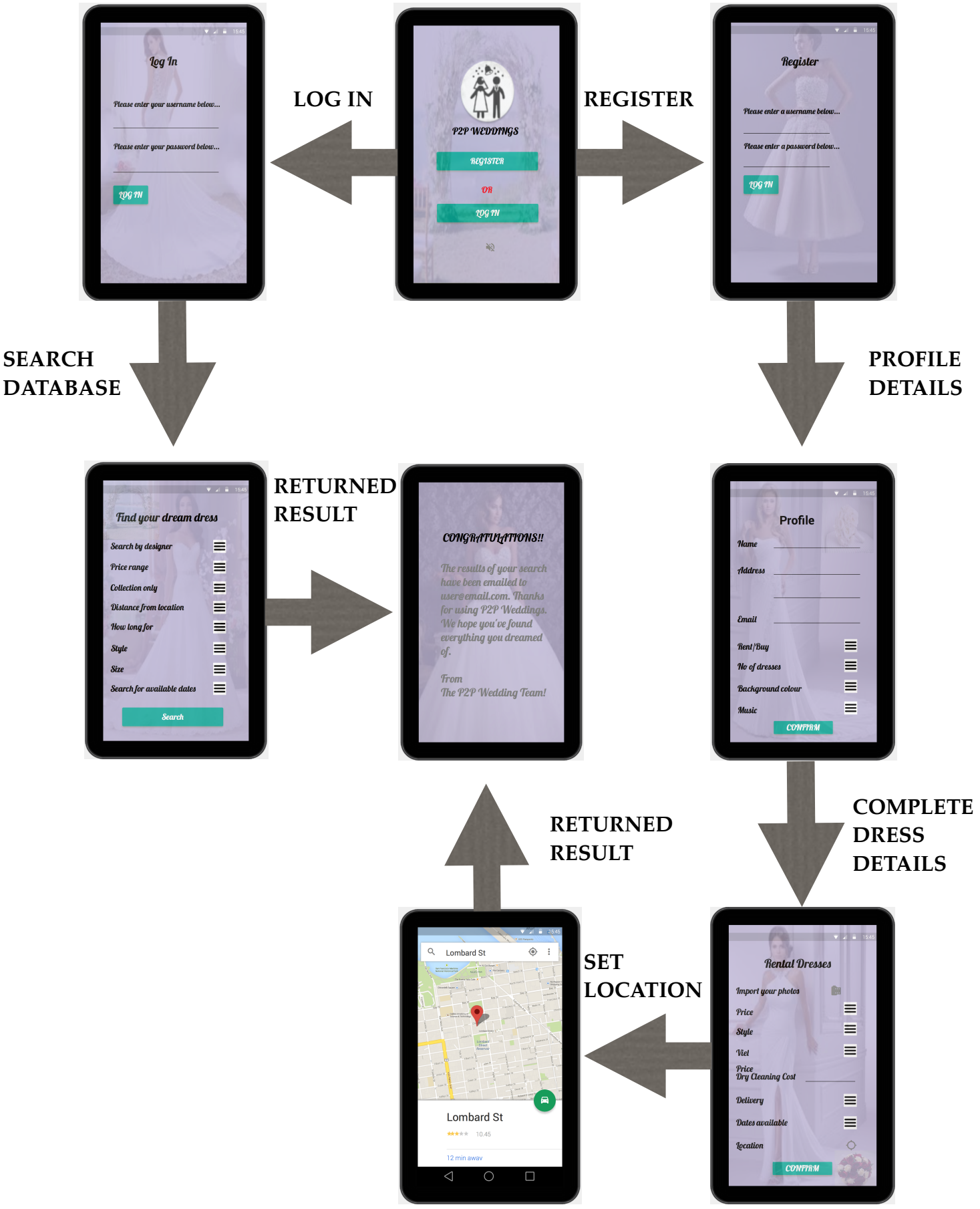
The idea for this App is to connect brides-to-be with already married women, with the hope of renting their wedding dresses for their own special occasion. Wedding dresses notoriously sit in wardrobes for years collecting dust, and in todays market, not many women can afford the dress that they would have dreamed off, while juggling the other costs involved in a modern-day wedding.

The name "P2P Weddings" is simplistic in nature and a little ambiguous. This was done on purpose as the idea would be to expand the service out to include renting suits, bridal gowns, wedding arrangements. Another potential service is to connect people with bespoke properties, ideally suited for wedding venues, with the future bride and groom. With this in mind the name would be adapted to home in on each specialisation, for e.g. P2P Wedding Dresses, P2P Wedding Venues, P2P Wedding Suits, P2P Wedding Shoes etc…

Taking into account what was discussed above the icon or company logo (Top right of this page) also had to be instantly recognisable with the wedding industry, but relatable to each of the above categories, without looking out of place or misleading. Therefore I have gone for a simple, fun design . A bride and groom depicted as stickmen in their wedding outfits, along with a bell and confetti in the background . Easily identifiable and using black and white as it's feature colours it can be added to the theme of each subdivision. Without looking out of place with that branches colour scheme. This icon is also used as the app launcher icon. The colour schemes used are light pastel colours typically associated with wedding colour schemes and a custom font (Lobster) has been imported and used. It is a very commonly used font with weddings and is easy on the eye.

Features;

- **Splash screen** with music leading to the home page
- **Home screen** with the ability to turn in app music on/off and an option for logging in or registration
- **Secure log in** using a web server and SQLite Database, warning messages are displayed if incorrect log ins are entered
- **Profile page** for new users to include contact details, price of rental dress on offer and the option to choose what colour theme they would like the app displayed in with four options
- **Dress details page** which requires the mentor to add photos of the on offer along with the style, size and other important information, all of which are store on an in-app SQLite Database
- **Google Maps Splash screen**
- **Google Maps page** to set the location of the renter
- **Search page** for brides-to-be to search the database for dress that match their wishes
- **Results page** informing user that any suitable dresses have been emailed to them along with the contact details of the renter
- **Action bar** has also been added to the top of each activity to allow the customer to exit the app at any point (In certain themes only).


Log In
Please enter your username below...
Please enter your password below...
LOG IN
LOG IN
P2P WEDDINGS
REGISTER
OR
LOG IN
REGISTER
Register
Please enter a username below...
Please enter a password below...
LOG IN
SEARCH DATABASE
PROFILE DETAILS
Find your dream dress
Search by designer
Price range
Collection only
Distance from location
How long for
Style
Size
Search for available dates
Search
RETURNED RESULT
CONGRATULATIONS!!
The results of your search have been emailed to user@email.com. Thanks for using P2P Weddings. We hope you've found everything you dreamed of.
From
The P2P Wedding Team!
Profile
Name
Address
Email
Rent/Buy
No of dresses
Background colour
Music
CONFIRM
COMPLETE DRESS DETAILS
RETURNED RESULT
Lombard St
Lombard St
12 min away
SET LOCATION
Rental Dresses
Import your photos
Price
Style
Viel
Price Dry Cleaning Cost
Delivery
Dates available
Location
CONFIRM

Layout & Code implementation

Screen 1 - Splash Screen - Sound/Logo/Icon Launcher

Layout;

Upon launching the app the customer is greeted with an image of the company logo expanded to fill the full screen for 5 seconds along with an organ version of here-comes-the-bride.

Functionality & Java Implementation;

Below is is the thread and run method used to apply the music and intent to pass to Main screen (Time has been modified since below screen shot);

```
//.create takes two parameters, the first is context. The current state of the Activity
splashSong = MediaPlayer.create(Splash.this, R.raw.splashsong); //R.raw.match is the folder and name of the music file
splashSong.start();

// thread
Thread timer = run() → {
        try{
            sleep(1000);
        }catch (InterruptedException e){
            e.printStackTrace();
        }finally {
            Intent openMainActivity = new Intent("com.example.michaelheneghan.p2pweddings.MainActivity");
            startActivity(openMainActivity);
        }
};
timer.start(); // start is a method from the thread class
```

Screen 2 - Home Screen - Register/Log in/Toggle Music

Layout;

The home screen appears directly after the splash screen is released. At the top of the page, the company logo is placed along with the name of the App. Below are three buttons, customers are able to click to choose what path to take. If it is their first time using the app they will need to click the register button to take them through to the register screen or if they are a return user they are able to log in. The third button allows the customer to choose whether they wish to continue listening to the organ music.

Functionality & Java Implementation;

A class intent is used to pass the customer through to the chosen screen upon clicking one of the two buttons, below is the on click code for the register button.

```
//Create a class intent to move to next activity when register button is clicked
registerButton.setOnClickListener((v) → {
            try {
                Class registerClass = Class.forName("com.example.michaelheneghan.p2pweddings.Register");
                Intent startRegister = new Intent(MainActivity.this, registerClass);
                startActivity(startRegister);
            }catch (Exception e){
                e.printStackTrace();
            }
});
```

A checkable toggle button has been used to enable customers to toggle the music on/off, as shown in the code below;

```
// Enables user to switch on/off organ in-app music
musicTB.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){
        if(musicTB.isChecked()){
            Toast.makeText(getApplicationContext(), "Background Music Started!", Toast.LENGTH_SHORT).show();
            music.start();
        }
        else {
            Toast.makeText(getApplicationContext(), "Background Music Paused!", Toast.LENGTH_SHORT).show();
            music.pause();
        }
    }
});
```

Screen 3 - Register Screen - Web Server/SQLite Database/ Show Characters Toggle

Layout;

Two TextViews have been used to inform customer what information needs to be inputted into the two EditText boxes, a hint has also been added to the EditText boxes to prompt the user.

On this screen you will also find a button to finalise registration once completed all necessary information, this will store the log in information on a web server . A text view that has an on click intent to pass the customer through to the log in page if they have previously registered.

There is also a checkable button enabling customers to show the characters in the password EditText (which are hidden by stars for security purposes).

Functionality & Java Implementation;

```
// Enables user to hide or show password characters - Default set to hide characters
registerCharTB.setOnClickListener((v) -> {
        if (!registerCharTB.isChecked()) {
            passwordET.setInputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD);
        } else {
            passwordET.setInputType(InputType.TYPE_CLASS_TEXT);
        }
});
```

Above you see the if/else statement which toggles revealing and hiding the password characters upon clicking the button.

A User class holds all the constructor to create a new user when registering.

User Constructor

```
String name, username, password;

public User(){

}

public User(String username, String password){

    this.username = username;
    this.password = password;

}
```

I have not provided screenshots of the following three classes to keep the size of the report down as there are several methods you are able to view in Android Studio along with comments explaining each method

- **A ServerRequests Class** to hold all the methods needed for the app to make a request to the server.
- **An Interface** to inform the activity calling the ServerRequests class, that the action is completed.
- **A UserLocalStore** class, essentially to hold getters and setters on the logged in status of the user and error message returns.

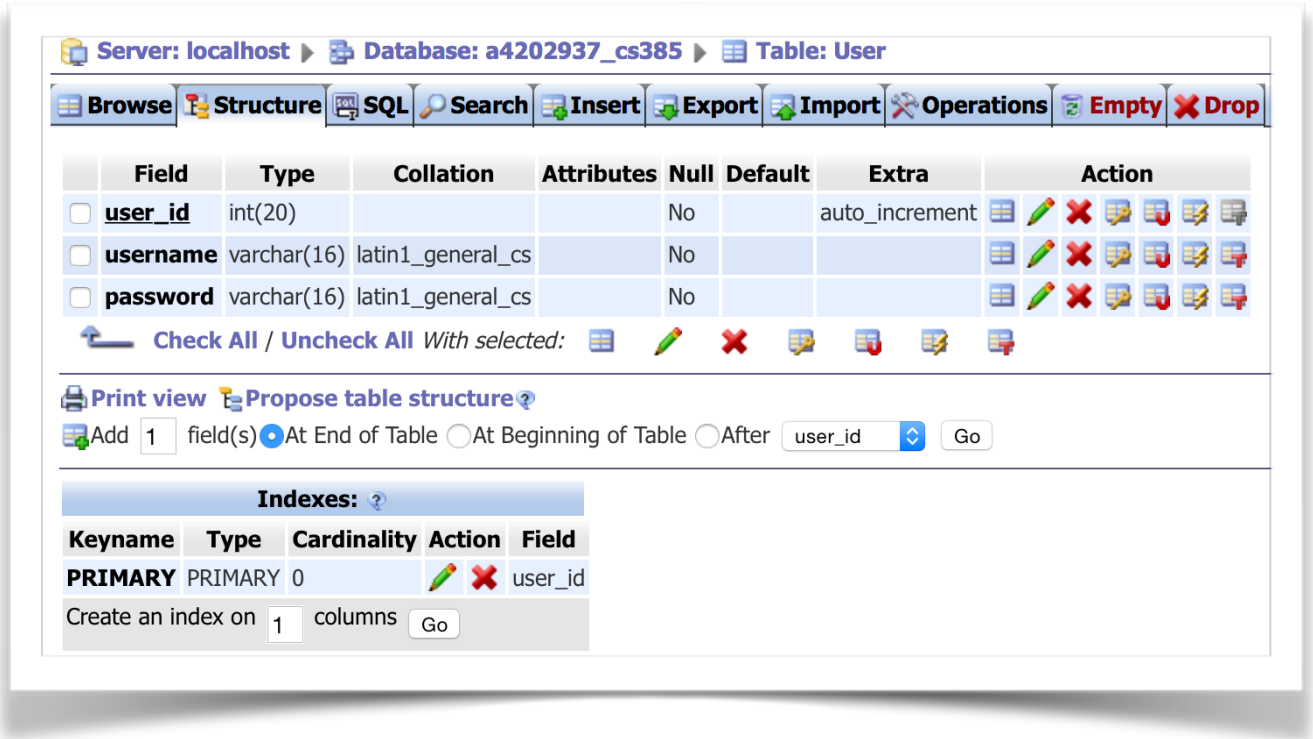
PHP & Web Server;

Below is the Register PHP file held in the web server to allow the app access to the database. It inserts the inputted information into the database using an inner SQL statement for later access.

Register

```
<? php
    $con=mysqli_connect("mysql7.000webhost.com", "a4202937_mikehen", "1Arsenal", "a4202937_cs385");

$username = $POST[username];
$password = $POST[password];

$statement = mysli_prepare($con, "INSERT INTO User(username, password) VALUES (?, ?) ");
mysqli_stmt_bind_param($statement,  "ss", $username, $password);
mysqli_stmt_execute($statement);
mysqli_stmt_close($statement);


mysqli_close($con)

?>
```

Below you will find the table as it exists in the web server, an auto-incremented id is used as the primary key, which is not shown or accessible to the user. A limit of 20 users has been set as I am not expecting to release this app. There is also a character limit of 16 on both the username and password. If the user enters no information the app will return a message prompting the user to add this information first.



Screen 4 - Login Screen - Web Server/SQLite Database/ Show Characters Toggle

Layout;

Two TextViews have been used to inform customer what information needs to be inputted into the two EditText boxes, a hint has also been added to the EditText boxes to prompt the user.

On this screen you will also find a button to login once they have filled in their username and password. This will check the SQLite database to confirm it is, in fact, the correct logins. The app will need to use the ServerRequests class explained above to access the web server. There is a text view that has an on click intent to pass the customer through to the Search page once confirmed.

There is also a checkable button enabling customers to toggle on/off password character visibility (like in the Register Class).

Functionality & Java Implementation;

This class has the same java code as the Register class shown above.

PHP & Web Server;

The FetchUserData class performs a select query allowing the app to access the previously stored user login data. If it is not correct it will return an error message, if it successful it will pass the user to the Search database screen. It is the same web server accessed in the Register class which you have already seen.

Fetch User Data

```
<? php
    $con=mysqli_connect("mysql7.000webhost.com", "a4202937_mikehen", "1Arsenal", "a4202937_cs385");

$username = $POST[username];
$password = $POST[password];

$statement mysqli_prepare($con, "SELECT * FROM User WHERE username = ? AND password = ?") ;
mysqli_bind_param($statement, "ss", $username, $password);
mysqli_stmt_execute($statement);

mysqli_stmt_store_result($statement);
mysqli_stmt_bind_result($statement, $user_id, $username, $password);

$user = array();
while(mysqli_stmt_fetch($statement)){
    $user[username] = $username;
    $user[password] = $password;
}

echo json_encode($user);

mysqli_stmt_close($statement);

mysqli_close($con)
```

Screen 5 - Profile Screen - SQLite Database/Profile Table/ Spinner

Layout;

Six TextViews have been used to inform customer what information needs to be inputted into the corresponding EditText boxes along with hints. A spinner has been added so the app can determine whether the user is planning on adding a dress to the database - 'rent' , or just browse the available dresses - 'buy'.

A submit profile button has been added, which when clicked will add the users details to the database and take them through to either the Search activity screen if they have selected 'buy' or through to the Dress details activity screen if they selected 'rent' to input dress details.

Functionality & Java Implementation;

An in-app database has been created within the UserPreferences class along with a profile table with create and insert statements. The primary key which is an id auto incremented upon each new user created and is not accessible to the user. Nine profiles have been hardcoded and inserted upon the OnCreate statement being called for your purposes, this will enable you (Tom) to run check the functionality of the table later in the app).

Create Statement

```
myDB = this.openOrCreateDatabase("ProfileDB",
        MODE_PRIVATE, null);

// Exectute SQL statement to create the profile table

myDB.execSQL("CREATE TABLE IF NOT EXISTS profile " +
        "(id integer primary key AUTOINCREMENT, username VARCHAR, useraddress VARCHAR, useremail VARCHAR, rentbuy VARCHAR, rentalprice VARCHAR);")
```

Insert Statement

```
//Execute SQL statement to insert new data
myDB.execSQL("INSERT INTO profile (username, useraddress, useremail, rentbuy, rentalprice) VALUES ('" + userName + "', '" +
        userAddress + "', '" + userEmail + "', '" + rentBuy + "', '" + dressRentalPriceET + "');");
```

For the spinner, two methods were created. One to create the adapter and set the values of the dropdown list to those contained in the Values.XML file. And another to set the onClick Listener and store the result in a variable which can then be used in the insert statements. This syntax was duplicated and reused for every other Spinner in this app, in hindsight a separate Spinner and Adapter class should have been created to improve efficiency and save duplicate code.

I have also included a delete database and delete contact method for the administrator of the app to manipulate data when needed, these options are not available for the user to do themselves to improve concurrency. A request would need to be made to the Administrator to perform this task by the user.

Screen 6 - Dress details Screen - SQLite Database/Dressdetails JOINS Table/Spinners/ Import Image/ Google Maps/ Byte Array Conversion

Layout;

Seven Spinners have been used for customers to pick which options match the style, size etc of their dress. A import photo button will navigate customers into their device gallery and allow them to pick one photo to demonstrate what their dress looks like. A location button lets users store their location on GoogleMaps for other interested users to get to. When the user clicks the 'Submit' button the app will store the details of their dress in a separate table connected to their profile details.

Functionality & Java Implementation;

A second JOINS database (1-Many) has been created and linked to the profile table using a foreign key (the primary key of the profile table).

Create Statement

```
//Execute SQL statement to insert new data
myDB.execSQL("INSERT INTO profile (username, useraddress, useremail, rentbuy, rentalprice) VALUES ('" + userName + "', '" +
        userAddress + "', '" + userEmail + "', '" + rentBuy + "', '" + dressRentalPriceET + "');");
```

Insert Statement

```
// Execute SQL statement to insert new data
myDB.execSQL("INSERT INTO dressdetails(profile_id, image1, designer, style, size, viel, drycleaning) VALUES ('" +
        idReceived + "', '" + yourSelectedImage + "', '" +
        Designer + "', '" + Style + "', '" + WhatSize + "', '" + WantViel + "', '" + DryCleaningCost + "');");
```

To insert the foreign key into the database, an intent to pass key value pairs from one activity to another is needed, along with another intent to receive this information and store it within a variable.

Send Key Info Intent

```
/// Intent to send foreign key to Dressdetails Class
Intent myIntent2 = new Intent(UserPreferences.this,DressDetails.class); myIntent2.putExtra("idPassed",getId(userEmail));
startActivity(myIntent2);
```

Receive Key Info Intent

```
/// Intent to receive foreign key passed from UserPreferences Class ///
String idReceived = getIntent().getExtras().getString("idPassed");
// Toast to check activity received foreign key, will be removed before lauch so user cannot view ///
Toast.makeText(this, idReceived, Toast.LENGTH_SHORT).show();
```

An intent was created to give users access to their device gallery as shown below.

Retrieve photo Intent

```
photoImportButton.setOnClickListener((v) -> {
        Intent photoPickerIntent = new Intent(Intent.ACTION_PICK); photoPickerIntent.setType("image/*");
        startActivityForResult(photoPickerIntent, SELECT_PHOTO);
});
```

A location button will take users to google maps using an intent.

GoogleMaps Intent

```
Intent startGoogleSplash = new Intent(DressDetails.this, GoogleMapsSplash.class); startActivity(startGoogleSplash);
Toast.makeText(DressDetails.this, "Button Clicked", Toast.LENGTH_SHORT).show();
```

A GoogleMaps splash screen was added to inform user they are being passed to google maps. This is similar to the opening splash screen of the app, along with a MapsActivity class, which inflates a fragment to display the map. It also includes commands to set their location on the map using latitude and longitude, zoom controls as well as enabling traffic, indoor and buildings.

GoogleMaps Class methods

```
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);

// Place dot on current location
googleMap.setMyLocationEnabled(true);

// Turns traffic layer on
googleMap.setTrafficEnabled(true);

// Enables indoor maps
googleMap.setIndoorEnabled(true);

// Turns on 3D buildings
googleMap.setBuildingsEnabled(true);

// Show Zoom buttons
googleMap.getUiSettings().setZoomControlsEnabled(true);

// Create a marker in the map at a given position with a title
Marker marker = googleMap.addMarker(new MarkerOptions().
        position(DerekPos).title("Hello"));
```

A google map API Key needed to be generated for the app from the google developers page. As well as adding certain permissions to the Android Manifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<permission
    android:name="com.example.michaelheneghan.p2pweddings.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"/>
<!-- Enables me to use openGL -->
<activity
    android:name=".GoogleMapsSplash"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.GOOGLEMAPSSPALSH" />

        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="@string/google_maps_key" />

<activity
    android:name=".MapsActivity"
    android:label="@string/title_activity_maps" >
</activity>
```

Screen 7 - SearchDatabase - SQLite Database/JOINS Select Statment/Export Image/Email Intent

Layout;

Four text views and spinners are used for the user to specify what designer, style and size they require and whether they wish to include a veil. There is also a clickable search button once all the options have been made.

Functionality & Java Implementation;

Users chosen options are stored in variables and then used in a JOINS select query. A cursor is used to navigate to matching tuples in dress details profile. The image of the matching dress is then stored as a string, along with the cost of dry cleaning on the applicable dress.

```
// Set cursor for any dress that match users input
Cursor c = myDB.rawQuery("SELECT image1, drycleaning FROM dressdetails INNER JOIN profile ON (" +
        "profile.id = dressdetails.profile_id) WHERE dressdetails.size LIKE '%" + size + "%' OR dressdetails.style LIKE '%" + style + "%' OR dressdetails.viel LIKE '%"
        + viel + "%';", null);
if(c.moveToFirst()){
    do {
        // store it as a string in a variable
        queryImageRetrieved = c.getString(c.getColumnIndex("image1"));
        dryCleaningDetails = c.getString(c.getColumnIndex("drycleaning"));
    } while (c.moveToNext());
}
c.close();
```

JOINS select statement & cursor

Images over the size of 10mb aren't possible to store within an in-app SQLite Database, therefore it was necessary to convert them to byte array using BitFactory and BitMap before inserting them into the database. Insert below;

String to Byte conversion

```
// Convert the string back into a byte array and decode
try{
    byte[] bytes = queryImageRetrieved.getBytes("UTF-8"); BitmapFactory.decodeByteArray(bytes, 0, queryImageRetrieved.length());
    // Catch any I/O exceptions
}catch (Exception e){
    e.printStackTrace();
}
```

Once a suitable dress has been identified, the retrieved image is stored. The program needs to retrieve the contact details of the dress owner from the profile table and store those in variables to send to the customer in an email. This is done using another cursor and empty string variables.

Retrieve contact details of dress owner using a select statement

```
// method 2 for returning contact details of matching dress
String rentorName = "";
String rentorEmail = "";
String rentalprice = "";
//Cursor c1 = myDB.rawQuery("SELECT username, useremail, rentalprice FROM profile where id = " + Integer.parseInt(queryIdRetrieved) + ";", null);
Cursor c2 = myDB.rawQuery("SELECT username, useremail, rentalprice FROM profile where id = ?", new String[] {queryImageRetrieved});
if(c2.moveToFirst()) {
    // Store the values needed in variables so we can send it to the user in an email with the image
    do {
        rentorName = c2.getString(c.getColumnIndex("username"));
        rentorEmail = c2.getString(c.getColumnIndex("useremail"));
        rentalprice = c2.getString(c.getColumnIndex("rentalprice"));
        sellerContactDetails = "The name of the seller is " + rentorName + ", their email address is " + rentorEmail +", the cost per" +
                "day is " + rentalprice + ", and the dry cleaning cost is " + dryCleaningDetails + ";";

        // Check that it has worked - Debug tool
        Toast.makeText(this, sellerContactDetails, Toast.LENGTH_SHORT).show();

        // Continue until no more dresses match the set criteria
    } while (c2.moveToNext());
}

// Close the cursor and the database
c.close();
myDB.close();
```

Once matching dresses have been found and stored in variables, these variable are then inputted into an email intent to send the image, dress details and contact details

Email intent

```
// Send results to users email
Log.i("Send email", "");

String[] TO = { rentorEmail };
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");


emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "P2P Weddings");
emailIntent.putExtra(Intent.EXTRA_TEXT, sellerContactDetails);

try {
    startActivity(Intent.createChooser(emailIntent, "Send mail..."));
    finish();
    Log.i("Finished sending email ", "");
} catch (android.content.ActivityNotFoundException ex) {
    Toast.makeText(SearchCriteria.this,
            "There is no email client installed.", Toast.LENGTH_SHORT).show();
}
```

Screen 8 - Results Message - Log out/Message To User

Layout;

Four TextViews are used to display a message to the user thanking them for their custom. Additionally, the message will inform them that any suitable dresses have been emailed to them along with the contact details of the dress owner. There is also a button to log out and exit the app.

Functionality & Java Implementation;

The logout button runs a method from the UserLocalStore class logging the user out and informing the server. A message is displayed to the user on screen informing them it was successful and exits the app.

```
/// Clear user data and logout user ///
userLocalStore.clearUserData();
userLocalStore.setUserLoggedIn(false);

/// Set intent to return to login page ///
Intent logout = new Intent(ResultsMessage.this, LogIn.class);
startActivity(logout);
Toast.makeText(this, "You have succesfully logged out, thanks for useing My P2P Weddings", Toast.LENGTH_SHORT).show();

////////// Methods to check username and password and display to screen //////////
```

Issues/Problems

Security
Originally I had planned and implemented shared preferences to store the username and passwords of users. I soon realised this was not a secure method and sought out other more secure ways for users to login/register and learnt about the ability to store them in databases, held in extern web servers, which solved the problem.

Database
I decided to follow Derek Banas' video tutorial on how to create a database in Android Studio as it looked the most simplistic. Unfortunately, as I came to realise, my app required two tables, one of which being a Joins table which is a lot more complicated. With that in mind, and in hindsight, if I was to implement a database again I would use the more conventional method of creating a separate class which holds the query methods to insert, select and update the database.

Importing/Exporting Images
Unfortunately inserting photos into a database is not a straightforward process due to their file size. It was necessary to store the images as integers then convert them to byte arrays before inserting them into the database, then the opposite when retrieving them. This was a very complex process and above my programming level, with some help from the website StackOverflow I was able to achieve this.

Size
I originally chose to do this project on my own as I was considering pursuing it as a business opportunity, but after further research, I realised that in order for it to be profitable I would need a large volume of transactions (due to the low profit on each rental). This would take a long time to achieve and a large amount of investment would be needed to have a professional website capable of doing what was needed. This left me with a very large volume of work to do, if I wanted to create the app how I had originally intended.

Future Development

- Select statement for searching joins table is not currently working
- Additional languages to support users in other countries like Spain
- Portrait and Landscape layout modes
- Another activity to view the results of the users search using a fragment in horizontal mode with a side panel. When each dress is selected from the side panel, the image and dress details would be displayed in the main fragment
- Action bar created nut not implemented it into my customised themes
- I have created four themes which change the colour of the app background, which the user chooses in their profile but ran out of time before creating the button to do so