Experiment No.3

- **Aim:** To include icons, images, fonts in Flutter app.

- **Theory:**

Icons:

1. Flutter provides support for both Material Design icons and Cupertino icons (iOS style).
2. Material icons can be used with the Icon widget by specifying the icon's name.
3. Cupertino icons can be used with the CupertinoIcons class.
4. You can customize the size, color, and other properties of icons using the size, color, and other parameters of the Icon widget.

Loading Images:

1. Images in Flutter can be loaded using the Image widget.
2. Flutter supports various image formats such as JPEG, PNG, GIF, WebP, and Animated GIFs.
3. Images can be loaded from the network using the NetworkImage class, from local assets using the AssetImage class, or from memory using the MemoryImage class.
4. Ensure to declare asset paths in the pubspec.yaml file under the flutter section to make them accessible in your app.

Customizing Fonts:

1. Flutter supports custom fonts, allowing you to use any TrueType (TTF) or OpenType (OTF) font in your app.
2. You can include custom fonts by adding the font files to your project's fonts directory and specifying them in the pubspec.yaml file.
3. After adding fonts to the project, you need to rebuild the app to make them available.
4. You can then use these custom fonts by specifying them in the TextStyle widget or directly in widgets like Text.

Image and Icon Optimization:

1. When using images, it's essential to consider their size and format to optimize app performance.
2. Use tools like flutter_image_compress to compress images without significantly reducing quality.
3. For icons, prefer vector-based formats like SVG whenever possible, as they scale without loss of quality and reduce app size.
4. Use the appropriate image loading techniques based on the image source (network, local assets, or memory) to minimize loading times and conserve device resources.

Asset Management:

1. Managing assets efficiently is crucial for maintaining a clean project structure and optimizing app performance.
2. Organize assets into logical folders within your project directory.
3. Use asset variants to provide different resolutions for images to ensure optimal display on different devices.
4. Consider using packages like flutter_svg for handling SVG images and google_fonts for easily integrating Google Fonts into your app.

- **Code:-**

1. Icon Widget :

```
const Icon(
  Icons.lock,
  size: 100,
), // Icon
```

This widget displays an icon representing a lock. It's used as a logo on the login page. To provide a visual representation of the app's branding or purpose.

2. Text Widget :

```
Text(
  'Welcome back you\'ve been missed!',
  style: TextStyle(
    color: Colors.grey[700],
    fontSize: 16,
  ), // TextStyle
), // Text
```

This widget displays a welcome message for the user. To greet the user and provide a friendly message.

3. MyTextField Widget :

```
MyTextField(
  controller: usernameController,
  hintText: 'Username',
  obscureText: false,
), // MyTextField
```

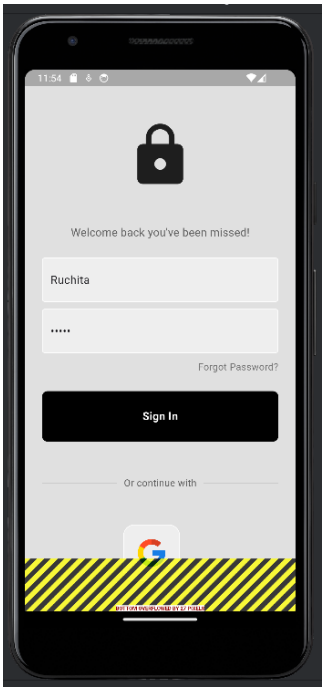
This custom widget (`MyTextField`) displays a text field for entering the username. To collect user input for the username.

4. MyButton Widget :

```
MyButton(
  onTap: signUserIn,
), // MyButton
```

This custom widget (`MyButton`) displays a button for signing in. To provide a clickable element for the user to initiate the sign-in process.

- **Output:-**



- **Conclusion:**

Incorporating icons, images, and fonts into the Flutter app has not only enhanced its visual appeal but also enriched its user experience. Leveraging these assets has enabled me to craft a more engaging interface while learning the importance of effective asset management and customization in Flutter development.