

## Unit-3,4 Notes

Ques1) Define error. Explain the types of errors along with its examples.

Ans)

**Syntax Errors:** Syntax errors occur when the rules of the programming language are not followed. These errors are caught by the compiler or interpreter when the code is being translated into machine code. They prevent the program from being compiled or executed. Examples include missing semicolons at the end of statements, misspelled variable or function names, or incorrect indentation.

Example:

### ○ Syntax errors

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

**Runtime Errors:** Runtime errors occur while the program is running. They are not detected by the compiler or interpreter during the compilation process but instead happen when a particular line of code is executed. Runtime errors can be caused by a variety of factors, such as division by zero, accessing an array element out of bounds, or attempting to open a file that does not exist.

Example:

### ○ Runtime errors

```
private void saveData(String filename) {
    FileOutputStream fileos = openFileOutput(filename, Context.MODE_APPEND);
}
```

Unhandled exception: java.io.FileNotFoundException  
Add exception to method signature    More actions...

Figure 12-2. IDE reminder that you need to handle the Exception

```
private void saveData(String filename) {
    FileOutputStream fileos = openFileOutput(filename, Context.MODE_APPEND);
}
```

Add exception to method signature  
Surround with try/catch  
Split into declaration and assignment  
Surround with try-with-resources block  
Add method contract to 'openFileOutput'

**Logic Errors:** Logic errors occur when the program runs successfully but produces incorrect results due to a mistake in the program's logic. These errors are often the most challenging to debug because the program's syntax and structure are correct, but the algorithm or logic used to solve the problem is flawed.

Example:

## ○ Logic errors

When the app runs, you can see the Log messages in the **Logcat** window, as shown in Figure 12-4. You can get to the Logcat window either by clicking its tab in the menu strip at the bottom of the IDE or from the main menu bar, **View ► Tool Windows ► Logcat**.

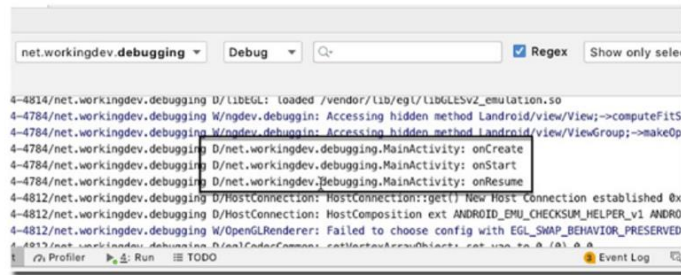


Figure 12-4. Logcat tool window

Ques3) Explain the types of testing in detail?

Ans)

- Functional testing (imp)
- Unit Testing (imp)
- Instrumented Testing (imp)
- Performance testing
- Compatibility testing
- Recovery testing

**Functional Testing:** Functional testing involves testing the application's functionality against the specified requirements. It ensures that the application behaves as expected and meets the user's needs. Test cases are designed to cover different features and use cases of the application.

**Unit Testing:** Unit testing involves testing individual units or components of the application in isolation. It helps identify bugs and issues early in the development process. Unit tests are typically written by developers and are automated to run frequently.

**Instrumented Testing:** Instrumented testing, also known as integration testing, involves testing the application's interactions with the device's hardware and software components. This type of testing is essential for mobile applications to ensure compatibility and performance across different devices and platforms.

**Performance Testing:** Performance testing involves testing the application's performance under various conditions, such as high load, low network connectivity, and limited resources. It helps identify performance bottlenecks and optimize the application for better performance.

**Compatibility Testing:** Compatibility testing involves testing the application's compatibility with different devices, operating systems, and screen sizes. It ensures that the application functions correctly on a wide range of devices and platforms.


**Recovery Testing:** Recovery testing involves testing the application's ability to recover from failures or errors gracefully. It includes testing scenarios such as app crashes, network failures, and interruptions to ensure that the application can recover and resume normal operation without data loss or corruption.

Ques4) Infer the steps for creating and monitoring geo fencing in android application?

Ans)

### Creating and Monitoring a Geofence

```
Geofence geofence = new Geofence.Builder()
    .setRequestId(GEOFENCE_REQ_ID) // Geofence ID
    .setCircularRegion( LATITUDE, LONGITUDE, RADIUS) // defining fence region
    .setExpirationDuration( DURATION ) // expiring date
    // Transition types that it should look for
    .setTransitionTypes( Geofence.GEOFENCE_TRANSITION_ENTER
        | Geofence.GEOFENCE_TRANSITION_EXIT )
    .build();
```



Deepika Kripanithi

### Geofence Transitions

- GEOFENCE\_TRANSITION\_DWELL: User spending sometime in the specified area
- GEOFENCE\_TRANSITION\_ENTER: Indication of the user entering the monitored region.
- GEOFENCE\_TRANSITION\_EXIT: Indication of the user exiting the monitored region.

Ques5) Design an android app to accept the name as: "VTU" and message as: "RV COLLEGE OF ENGINEERING" to store in SQLite data base with its steps.

Ans) Lab-Program5

Ques6) Devise a program to receive the latitude and longitude coordinates of a mobile device?

Ans)

steps:

1.inside manifest file

```
<user-permission android:name = "android.permission.ACCESS_COARSE_LOCATION"/>
```

```
<user-permission android:name = "android.permission.ACCESS_FINE_LOCATION"/>
```

2. MainActivity.java

```
Button button;
```

```
TextView textView;
```

```
LocationManager locationManager;
```

```
LocationListener listener;
```

```
textView = findViewById(R.id.txtView);
```

```
button = findViewById(R.id.btn);
```

```
locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
```

```

listener = new LocationListener() {

    @Override

    public void onLocationChanged(@NotNull Location location) {

        textView.setText("\n " +location.getLongitude() + " " + location.getLatitude());

    }

};

button.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.R){

            if (ActivityCompat.checkSelfPermission(this, ACCESS_COARSE_LOCATION) == PackageManager.PERMISSION_GRANTED) {

                getLocation();

            }

            else {

                ActivityCompat.requestPermissions(this, new String[] { ACCESS_COARSE_LOCATION},1);

            }

            locationManager.requestLocationUpdates("gps",5000, 0, listner);

        }

    }

});

```

Ques7) write a note on usage of log in logcat?

Ans)

In Android development, Logcat is a tool provided by the Android SDK that allows developers to view log messages from their applications and system messages. Logging is a critical practice in software development as it helps developers track the execution flow, identify issues, and debug problems in their applications. Here are some key points on the usage of logging in Logcat:

1.Logging Levels: Logcat provides different logging levels to categorize log messages based on their severity or importance. The common logging levels are:

- **VERBOSE:** Verbose logging messages that are typically used for debugging purposes. These messages provide detailed information about the application's execution.
- **DEBUG:** Debug logging messages that are used for debugging the application. These messages are helpful for developers to track the flow of the application and identify issues.
- **INFO:** Informational logging messages that provide general information about the application's execution. These messages are typically used to track the application's state.
- **WARN:** Warning logging messages that indicate potential issues in the application. These messages are used to alert developers about possible problems.
- **ERROR:** Error logging messages that indicate errors in the application. These messages are used to identify and fix errors in the application.
- **ASSERT:** Assert logging messages that are used to assert conditions in the application. These messages are typically used for testing purposes.
- **Logging Syntax:** In Android development, logging is typically done using the Log class, which provides static methods for logging messages. The basic syntax for logging messages is as follows:

```
Log.<LOG_LEVEL>(TAG, "Message");
```

```
Log.d("MainActivity", "Debug message");
```

## 2. Logging Best Practices:

- Use meaningful tags: Tags help identify the source of log messages. Use descriptive tags to easily identify the origin of log messages.
- Avoid excessive logging: Logging too many messages can impact the performance of your application. Use logging judiciously and only log messages that are necessary for debugging.
- Use different logging levels: Use different logging levels based on the severity of the message. For example, use DEBUG for debugging messages and ERROR for error messages.

3. Viewing Logcat Output: Logcat output can be viewed using the Android Device Monitor in Android Studio or the `adb logcat` command in the terminal. You can filter log messages based on tags, logging levels, or search terms to view specific messages.

Ques8) Elaborate any three log methods of logcat with an example?

Ans)

Logcat is a command-line tool provided by Android that displays messages from the system and applications as they are logged. It's useful for debugging and troubleshooting Android applications. Here are three common log methods used in Logcat:

1. **Log.d():** This method is used to log debug messages. These messages are typically used for debugging purposes and can provide information about the application's state, variables, and flow.

Syntax: `Log.d(String tag, String message);`

Example:

```
int x = 10; Log.d("MainActivity", "The value of x is: " + x);
```

In this example, the debug message will be logged with the tag "MainActivity" and will display the value of the variable x.

2. **Log.e():** This method is used to log error messages. These messages are used to log errors and exceptions that occur during the execution of the application.

Syntax: `Log.e(String tag, String message);`

Example:

```
try { // Some code that may throw an exception } catch (Exception e) { Log.e("MainActivity", "An error occurred: " + e.getMessage()); }
```

In this example, if an exception occurs in the try block, the error message will be logged with the tag "MainActivity" and will display the exception message.

3. **Log.i():** This method is used to log informational messages. These messages are used to provide information about the application's state or behavior.

Syntax: `Log.i(String tag, String message);`

Example:

```
String username = "john.doe"; Log.i("LoginActivity", "User logged in: " + username);
```

In this example, the informational message will be logged with the tag "LoginActivity" and will display the username of the user who logged in.

These are just a few examples of the log methods available in Logcat. There are other methods such as Log.v() for verbose messages, Log.w() for warnings, and Log.wtf() for logging a message that should never happen. Each of these methods is used for different types of logging messages and can be useful for different debugging scenarios.

Ques9) Explain the most common errors which occur during debugging in android programming?

Ans)

Debugging in Android programming can be challenging due to the complexity of the Android platform and the variety of devices and configurations. Some of the most common errors that occur during debugging in Android programming include:

1. **NullPointerException:** This error occurs when trying to access or use a reference variable that is not pointing to any object (i.e., null). It often happens when developers forget to initialize an object or when a method returns null unexpectedly.
2. **ClassCastException:** This error occurs when trying to cast an object to a class type that is not compatible. For example, trying to cast a TextView to a Button can result in a ClassCastException.
3. **OutOfMemoryError:** This error occurs when the application runs out of memory. It can happen when loading large bitmaps, processing large data sets, or running multiple processes simultaneously.
4. **StaleDataException:** This error occurs in Android's ContentProvider framework when trying to access data that has become stale or outdated. It often happens when the underlying data has been modified or deleted by another process.
5. **IllegalStateException:** This error occurs when the application is in an illegal or inappropriate state. It can happen when trying to perform an operation that is not allowed in the current state of the application.
6. **NetworkOnMainThreadException:** This error occurs when trying to perform network operations on the main thread. Android does not allow network operations on the main thread to prevent blocking the UI. Developers should use background threads or AsyncTask to perform network operations.
7. **ResourceNotFoundException:** This error occurs when trying to access a resource (such as a layout file or drawable) that does not exist. It can happen when the resource is misspelled or when the resource file is missing from the project.
8. **SecurityException:** This error occurs when the application tries to perform a security-sensitive operation without the necessary permissions. It can happen when trying to access the camera, location, or other sensitive features without requesting the appropriate permissions in the manifest file.

Ques10) Elaborate any three debugging actions in Android Studio IDE?

Ans)

1. **Breakpoints:** Breakpoints allow developers to pause the execution of their application at a specific line of code. This allows them to inspect the state of the application at that point, including variable values, method calls, and stack traces. To set a breakpoint, simply click on the left margin of the code editor next to the line where you want to pause the execution. When the application reaches that line during debugging, it will pause, and you can use the debugging tools in Android Studio to inspect the state of the application.
2. **Debugging Toolbar:** The debugging toolbar in Android Studio provides various actions and controls for debugging your application. It allows you to start and stop debugging, step through your code, resume execution, and more. The debugging toolbar is located at the top of the Android Studio window when you are in debug mode. It contains buttons for common debugging actions, such as stepping into, over, and out of methods, as well as controls for pausing and resuming execution.
3. **Logcat:** Logcat is a logging tool in Android Studio that displays messages from the system and applications as they are logged. It can be used to debug issues in your application by displaying log messages that you have added to your code using the Log class. To view log messages in Logcat, open the Logcat tab at the bottom of the Android Studio window. You can filter log messages by log level (e.g., debug, error, info) and by tag to help you find the messages you are interested in.
4. **Watch Window:** The Watch Window in Android Studio IDE allows developers to monitor the values of specific variables or expressions during debugging. Developers can add variables or expressions to the Watch Window and track their values as the program executes. This feature is useful for inspecting how the values of variables change over time and can help developers identify issues in their code. The Watch Window provides a convenient way to keep track of important variables and expressions without cluttering the main debugging view.

Ques11) Create a program with a table of user credentials using SQLite database?

Ans) Lab-Program5

Ques) Difference between Unit Testing and Instrumented Testing?

Ans)

Here is a tabular comparison between Unit Testing and Instrumented Testing in Android:

Feature	Unit Testing	Instrumented Testing
Execution Environment	Runs on the local JVM	Runs on an Android device or emulator
Dependencies	Does not require a device or emulator	Requires a device or emulator
Speed	Faster	Slower
Scope	Tests individual units or components	Tests integration with Android system
Access to Resources	Limited access to Android resources	Full access to Android resources
Use Cases	Used for testing business logic, algorithms, etc.	Used for testing UI interactions, database access, etc.
Example Frameworks	JUnit, Mockito	Espresso, UI Automator

Ques) Write a short note on Debugger in Mobile Application Development?

Ans)

A debugger is a software tool used in mobile application development to help developers identify and resolve issues in their code. It allows developers to pause the execution of their application at specific points, inspect the state of the application, and track the flow of execution. Debuggers provide various features to aid in debugging, such as setting breakpoints, stepping through code, viewing variable values, and examining the call stack.

In mobile application development, debugging is crucial for identifying and fixing bugs, errors, and unexpected behaviors in the application. Debuggers help developers understand how their code is executing and find the root cause of issues quickly and efficiently. By using a debugger, developers can improve the quality, reliability, and performance of their mobile applications.

-----X-----X-----

Other Topic:

1.

## Data and file storage overview

Android uses a file system that's similar to disk-based file systems on other platforms. The system provides several options for you to save your app data:

- **App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- **Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
- **Preferences:** Store private, primitive data in key-value pairs.
- **Databases:** Store structured data in a private database using the Room persistence library.

2.

## Android Internal Storage

**Android Internal storage** is the storage of the private data on the device memory. By default, saving and loading files to the internal storage are private to the application and other applications will not have access to these files. When the user uninstalls the applications the internal stored files associated with the application are also removed. However, note that some users **root** their Android phones, gaining superuser access. These users will be able to read and write whatever files they wish.

### Reading and Writing Text File in Android Internal Storage

Android offers `openFileInput` and `openFileOutput` from the Java I/O classes to modify reading and writing streams from and to local files.

- **openFileOutput():** This method is used to create and save a file. Its syntax is given below:

```
FileOutputStream fOut = openFileOutput("file name",Context.MODE_PRIVATE);
```

The method `openFileOutput()` returns an instance of `FileOutputStream`. After that we can call write method to write data on the file. Its syntax is given below:

```
String str = "test data";  
fOut.write(str.getBytes());  
fOut.close();
```

3.

- **openFileInput():** This method is used to open a file and read it. It returns an instance of `FileInputStream`. Its syntax is given below:

```
FileInputStream fin = openFileInput(file);
```

After that, we call read method to read one character at a time from the file and then print it. Its syntax is given below:

```
int c;  
String temp="";  
while( (c = fin.read()) != -1){  
    temp = temp + Character.toString((char)c);  
}  
  
fin.close();
```

4.

## Android External Storage

External storage such as SD card can also store application data, there's no security enforced upon files you save to the external storage. In general there are two types of External Storage:

- **Primary External Storage:** In built shared storage which is "accessible by the user by plugging in a USB cable and mounting it as a drive on a host computer". Example: When we say Nexus 5 32 GB.
- **Secondary External Storage:** Removable storage. Example: SD Card

All applications can read and write files placed on the external storage and the user can remove them. We need to check if the SD card is available and if we can write to it. Once we've checked that the external storage is available only then we can write to it else the save button would be disabled.

5.

1. **Environment.getExternalStorageState():** returns path to internal SD mount point like `"/mnt/sdcard"`
2. **getExternalFilesDir():** It returns the path to files folder inside `Android/data/data/application_package/` on the SD card. It is used to store any required files for your app (like images downloaded from web or cache files). Once the app is uninstalled, any data stored in this folder is gone too.



## 6. Location / Geo fencing

- Location awareness offers many benefits to an app, so many in fact that even desktop apps now attempt to get the user's location. Location uses ranges from **turn-by-turn directions**, "**find the nearest**" applications, alerts based on location, and there are now even location based **games** that get exploring with the device.
- The **Google APIs** offer many rich features for creating location-aware applications and mapping features.
- First step - will look at obtaining the last known location on the device along with receiving updates as the location changes.
- If requesting location updates for a proximity location - take a look at using the Geofence option instead in the *Create and monitor a Geofence*

## 7. Geofence

### Creating and monitoring Geofence -

If the application needs to know when the user enters or exits a certain location, there's an alternative to continuously checking the user location: Geofencing.

A Geofence is a location (latitude and longitude) along with a radius. Create a Geofence and let the system notify when the user enters the location proximity you specified. (Android currently allows up to 100 Geofences per user.)

Geofence properties include:

1. **Location:** The longitude and latitude
2. **Radius:** The size of the circle (in meters)
3. **Loitering delay:** How long the user may remain within the radius before sending notifications
4. **Expiration:** How long until the Geofence automatically expires
5. **Transition type:**
  - GEOFENCE\_TRANSITION\_ENTER
  - GEOFENCE\_TRANSITION\_EXIT
  - INITIAL\_TRIGGER\_DWELL

## Unit-4

Ques1) with a neat diagram explain the progressive enhancement in web application and offline support in PWA?

Ans)

### Progressive Enhancement

As user's **browser** gets more modern, more **features** become available to **user**. Progressive enhancement is a strategy for web design that emphasizes **core web page** content first. This strategy then progressively adds more nuanced and technically rigorous layers of presentation and features on top of the content as the end-user's **browser/Internet connection allow**. True power of PWAs comes in - users' **experiences get progressively better as their browsers get better**.

**Progressive Enhancement in Web Applications:** Progressive Enhancement is a strategy for web design that emphasizes accessibility, flexibility, and performance. It starts with a basic, usable version of a website and then adds more features and enhancements as the user's browser or device supports them.



Diagram:

- Layer 1 (Baseline): Basic HTML content that is accessible to all users and devices.
- Layer 2 (Enhanced Styling): CSS is added to improve the presentation and layout of the content.
- Layer 3 (Interactivity): JavaScript is added to enhance the user experience with interactive elements and dynamic content.
- Layer 4 (Advanced Features): Additional features like animations, multimedia, or advanced functionality are added for modern browsers that support them.

### Offline Support

The main page of the app loads even while the user is **offline**. This is accomplished with **service workers**. Figure below shows how to **cache** your app's assets so that even if the users don't have the best Internet connection (or a connection at all)

**Offline Support in Progressive Web Apps (PWA):** Progressive Web Apps are web applications that provide a native app-like experience to users, including offline support. PWAs use service workers, which are scripts that run in the background and can intercept network requests, cache resources, and provide offline access to content.

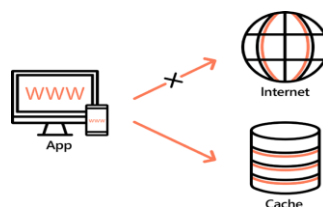


Diagram:

- Service Worker: A JavaScript file that manages caching and enables offline access to resources.
- Cache Storage: Stores cached resources such as HTML, CSS, JavaScript, and images.
- Network Requests: When online, the service worker intercepts network requests and can serve cached resources instead of fetching them from the network.

- Offline Access: Even when the device is offline, the PWA can still load and display content from the cache, providing a seamless user experience.

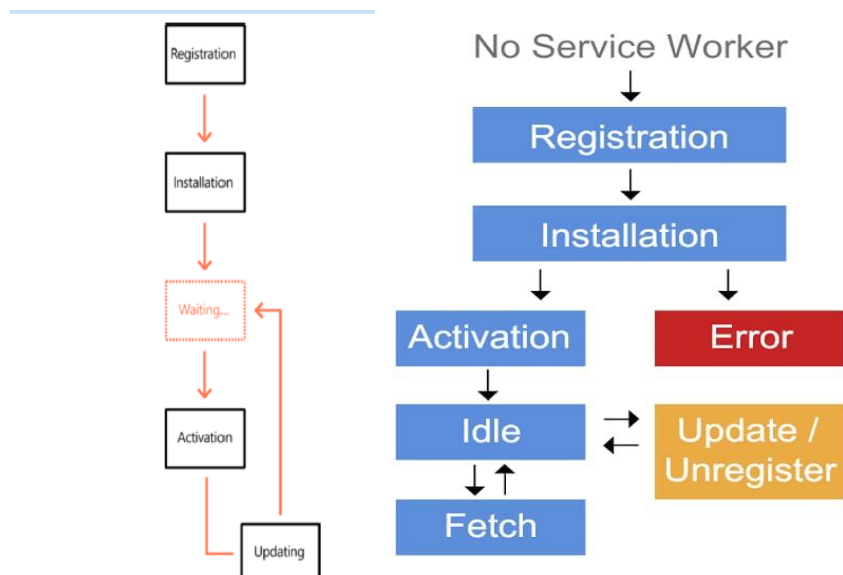
Ques2) Differentiate an android based mobile app with a progressive web application?

Ans)

Feature	Android-based Mobile App	Progressive Web Application (PWA)
Installation	Installed from app stores (Google Play Store)	Accessed through a web browser
Development	Developed using native languages (Java/Kotlin)	Developed using web technologies (HTML, CSS, JavaScript)
Platform Support	Specific to Android platform	Cross-platform (works on any device with a modern browser)
Offline Support	Full offline support	Limited offline support through caching (Service Workers)
Access to Device Features	Full access to device features (camera, sensors, etc.)	Limited access to device features (dependent on browser support)
Updates	Updates require user to download and install from app store	Updates automatically when user accesses the app (new content cached)
Discoverability	Listed in app stores for easy discovery	Requires users to find and bookmark the URL or be prompted to add to home screen
Storage	Can store large amounts of data locally	Limited storage capabilities (dependent on browser and device)
Performance	Generally faster and more responsive	Can be slower and less responsive, especially on older devices or poor network conditions

Ques3) With a neat diagram explain the service worker life cycle methods and working?

Ans)



## 1. Registration and scope

- To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser where your service worker is located, and to start installing it in the background.

## 2. Install

- A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the Service worker installs.

## 3. Activation

- When the service worker activates an activate event is triggered in the activating service worker. This event listener is good to clean up outdated caches.

**4. Fetch:** This method is called whenever the app makes an HTTP request. It intercepts the request and allows the service worker to respond with cached resources or fetch resources from the network.

**5. Updating:** When a new version of the service worker is available, it is installed in the background. The new version does not take control until the old version is no longer in use, ensuring a seamless update experience for the user.

**6. Waiting:** The "waiting" state refers to a service worker that has been registered but is waiting to become active. This happens when there is a previous version of the service worker already controlling the page. The new service worker is installed and enters the waiting state until the existing page is closed or unloaded, allowing the new service worker to take control.

**7. Error:** The "error" state occurs when a service worker encounters an error during its life cycle, such as an error during installation or activation. When an error occurs, the service worker is considered to be in the error state and may not function correctly. Developers can use this state to handle errors and implement fallback strategies.

**8. Idle:** The "idle" state refers to a service worker that is registered and active but not currently processing any events. In this state, the service worker is waiting for events such as fetch or message events to occur. The idle state is the normal state for a service worker when it is not actively handling events.

Ques4) List and explain the different audits available in lighthouse.

Ans)

Lighthouse is an open-source tool from Google that helps developers improve the quality of web pages. It provides a suite of audits that analyze various aspects of a web page, such as performance, accessibility, SEO, and best practices. Here are the different audits available in Lighthouse:

1. **Performance:** This audit measures the performance of a web page, including metrics such as First Contentful Paint (FCP), Speed Index, and Time to Interactive (TTI). It provides recommendations for improving the page's performance, such as optimizing images, reducing server response times, and minimizing render-blocking resources.
2. **Accessibility:** This audit checks the accessibility of a web page for users with disabilities. It identifies issues such as missing alternative text for images, low contrast text, and missing form labels. It provides suggestions for improving accessibility, such as using semantic HTML elements and providing keyboard navigation.
3. **Best Practices:** This audit checks for best practices in web development, such as using HTTPS, avoiding deprecated APIs, and ensuring that resources are loaded efficiently. It provides recommendations for improving the overall quality of the web page.
4. **SEO (Search Engine Optimization):** This audit checks for SEO best practices, such as using descriptive page titles, meta descriptions, and heading tags. It provides recommendations for improving the page's visibility in search engine results.

5. **Progressive Web App (PWA):** This audit checks if the web page meets the criteria for a Progressive Web App, such as being responsive, running over HTTPS, and having a valid web app manifest. It provides suggestions for turning the web page into a PWA.
6. **Accessibility Best Practices:** This audit checks for common accessibility issues, such as missing alternative text for images, low contrast text, and missing form labels. It provides suggestions for improving accessibility, such as using semantic HTML elements and providing keyboard navigation.
7. **Web Vitals:** This audit checks for metrics related to user experience, such as Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), and First Input Delay (FID). It provides recommendations for improving these metrics to provide a better user experience.

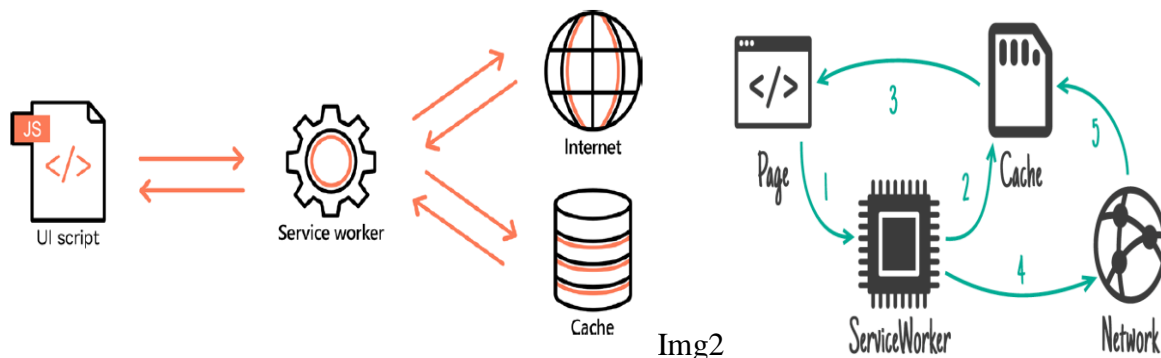
Ques5) Design and develop a program for inventory web site with the following:

- i) Manifest file(with at least 10 attributes).
- ii) Index page with proper imports and html tags

Ans) Lab-Program8

Ques6) With a neat diagram explain the service worker architecture?

Ans)



Img1

Img2

### Service worker

- A service worker is a script that runs in the **background** of the web application. A service worker is a **type of web worker**.
- It's essentially a JavaScript file that runs separately from the main browser thread, intercepting network requests, **caching or retrieving** resources from the cache, and delivering **push messages**.
- Service workers run in a **separate** thread from the UI, so they don't block or freeze the UI while they process.
- The whole point of a service worker is that it acts as an **intermediary** between the app and the Internet.

Diagram:

1. **Web Page:** The web page interacts with the service worker through APIs such as `navigator.serviceWorker` to register and communicate with the service worker.
2. **Service Worker:** The service worker is a JavaScript file that runs in the background, separate from the main browser thread. It intercepts network requests, caches resources, and manages background tasks.
3. **Cache Storage:** The service worker uses the Cache Storage API to cache resources such as HTML, CSS, JavaScript, and images. This allows the PWA to work offline by serving cached resources when the network is unavailable.
4. **Fetch Event:** When the web page makes an HTTP request, the service worker intercepts the request using the Fetch Event API. The service worker can then respond with cached resources or fetch resources from the network.

5. Network: The service worker can communicate with the network to fetch resources or send/receive data. It can also listen for push notifications from the server to display notifications to the user.

Ques6) Design and develop a program with proper code in service worker lifecycle methods to cache files and fetch during the time of offline. (files to be cached are-'pirates.html', '/styles/pirates.css', '/styles/pirates.tff', '/images/i-love-pirates.jpg', 'pirates-caribbean2.png')?

Ans) Lab program6

Ques7) Differentiate any ten features of manifest files of PWA and Android Studio?

Ans)

Feature	PWA Manifest File	Android Studio Manifest File
<b>1. Name</b>	Defines the name of the app.	Defines the name of the app.
<b>2. Short Name</b>	Defines a short name for the app (displayed on the home screen).	Not applicable.
<b>3. Icons</b>	Specifies icons for the app (used for home screen shortcuts).	Specifies icons for the app.
<b>4. Start URL</b>	Defines the URL that loads when the app is launched.	Not applicable.
<b>5. Display Mode</b>	Specifies how the app is displayed (e.g., fullscreen, standalone, minimal-ui).	Not applicable.
<b>6. Theme Color</b>	Specifies the color theme of the app.	Specifies the theme color of the app.
<b>7. Background Color</b>	Specifies the background color of the app.	Specifies the background color of the app.
<b>8. Orientation</b>	Specifies the default orientation of the app (portrait, landscape).	Specifies the orientation of the app.
<b>9. Permissions</b>	Specifies permissions required by the app (e.g., camera, geolocation).	Specifies permissions required by the app.
<b>10. Description</b>	Provides a description of the app.	Provides a description of the app.

Ques8) Describe Google Lighthouse and any three auditing functions?

Ans)

Google Lighthouse is an open-source tool for improving the quality of web pages. It provides audits for performance, accessibility, progressive web apps, SEO, and more. Here are three auditing functions provided by Google Lighthouse:

1. Performance Audit: The performance audit measures various aspects of a web page's performance, such as loading speed, responsiveness, and rendering performance. It provides recommendations for improving performance, such as optimizing images, minimizing render-blocking resources, and leveraging browser caching.
2. Accessibility Audit: The accessibility audit checks the accessibility of a web page for users with disabilities. It identifies issues such as missing alternative text for images, low contrast text, and missing form labels. It provides suggestions for improving accessibility, such as using semantic HTML elements and providing keyboard navigation.
3. SEO Audit: The SEO audit checks for search engine optimization best practices, such as using descriptive page titles, meta descriptions, and heading tags. It provides recommendations for improving the page's visibility in search engine results, such as using relevant keywords and creating a sitemap.

Ques9) Compare any five features of the service work in PWA to the gradle files in Android Studio?

Ans)

Feature	Service Worker (PWA)	Gradle Files (Android Studio)
1. Purpose	Enables offline access, background sync, and push notifications in web apps.	Automates building, testing, and deploying Android apps.
2. File Extension	Uses a JavaScript file with the <code>.js</code> extension.	Uses text files with the <code>.gradle</code> extension.
3. Configuration	Configured using JavaScript code to intercept network requests, cache resources, and manage background tasks.	Configured using Groovy or Kotlin DSL to define build settings, dependencies, and tasks.
4. Integration	Integrated into web apps using the <code>navigator.serviceWorker</code> API.	Integrated into Android projects as part of the build process.
5. Functionality	Provides advanced features such as caching resources for offline access, intercepting network requests, and enabling background synchronization and push notifications.	Manages build configurations, dependencies, and tasks for building, testing, and deploying Android apps.

Ques10) Describe the manifest file instructing the browser to treat it as PWA?

Ans) Lab8

Ques11) Write a program to demonstrate the register and install events in PWA?

Ans) Lab7

Ques12) Mention Chrome developer tools for assessing PWA features?

Ans) To access Developer Tools ("DevTools") in Chrome, open a web page or web app in Google Chrome. Click the Chrome menu Chrome Menu Icon icon, and then select More Tools > Developer Tools.

1. Console: The Console panel displays log messages, errors, and warnings logged by the service worker and the web app. It is useful for debugging issues related to service workers, JavaScript, and other aspects of the web app.
2. Network: The Network panel shows all network requests made by the web app, including requests made by the service worker. It allows you to inspect request and response headers, view response bodies, and analyze network performance.
3. Application: The Application panel is specifically designed for working with web app features such as the manifest file, service workers, and caches. It provides tabs for inspecting the manifest file, managing service workers, and viewing cache storage.
4. Source: The Source panel allows you to debug and inspect the source code of your web app, including JavaScript, CSS, and HTML files. It provides features such as breakpoints, step-through debugging, and variable inspection.
5. Manifest: The Manifest panel displays information about the web app's manifest file, including the app's name, short name, start URL, display mode, and icons. It allows you to inspect the manifest file and ensure that it is correctly configured.
6. Service Worker: The Service Worker panel provides information about the service workers registered for the web app. It allows you to inspect the service worker's code, view its lifecycle events, and debug any issues related to service worker registration and execution.

### Ques13) Tools to Measure Progressive Web Apps?

Ans)

#### 1. *Lighthouse - A Light to Keep You Off the Rocks*

Categories Lighthouse tests for

- Progressive Web App
- Performance
- Accessibility
- Best Practices
- SEO – Search Engine Optimization

#### 2. *Chrome Dev Tools for PWA*

### Ques14) Differentiate with features that depict that PWA is better than native applications tabular?

Ans)

Feature	Progressive Web Apps (PWA)	Native Applications
<b>Installation</b>	Installed directly from the browser, no app store required.	Requires installation from an app store.
<b>Accessibility</b>	Accessible via a URL, can be easily shared and accessed.	Requires download and installation, not as easily shareable.
<b>Cross-Platform Compatibility</b>	Works on any device with a modern browser.	Requires development for specific platforms (iOS, Android, etc.).
<b>Updates</b>	Updates automatically when the user accesses the app.	Requires user to manually update from app store.
<b>Storage</b>	Uses less device storage as resources can be cached.	May consume more device storage, especially for large apps.
<b>Cost</b>	Typically cheaper to develop and maintain.	Development and maintenance costs can be higher.
<b>SEO</b>	Can be indexed by search engines, improving discoverability.	Not as easily discoverable by search engines.
<b>Offline Access</b>	Can work offline and in low network conditions.	Native apps may have limited offline capabilities.
<b>Development Time</b>	Faster development time, especially for simple apps.	Development time may be longer, especially for complex apps.
<b>Platform Limitations</b>	Not limited by platform restrictions.	Limited by platform-specific restrictions and guidelines.

### Ques15) List important features of PWA and describe at least 4 features that depict that PWA is better than Web Applications?

Ans)

Features that depict that PWA is better than traditional web applications:

1. **Offline Access:** PWAs can work offline or in low-connectivity environments, providing a seamless user experience even when the network is unreliable or unavailable.



2. App-Like Experience: PWAs can be installed on the user's device and launched from the home screen, providing an app-like experience without the need for app store installation.
3. Fast and Reliable: PWAs are designed to be fast and reliable, with quick load times and smooth navigation, even on slow or unreliable networks.
4. Engagement: PWAs can leverage features like push notifications to engage users and bring them back to the app, increasing user engagement and retention.
5. Cross-Platform Compatibility: PWAs work across different platforms and devices, reducing the need to develop separate apps for each platform and providing a consistent user experience.
6. Cost-Effective: PWAs can be more cost-effective to develop and maintain compared to native apps, as they can be built using web technologies and require less platform-specific development.

Ques17) Compare the Manifest Files of PWA with Android Studio?

Ans)

Feature	PWA Manifest File	Android Studio Manifest File
<b>Purpose</b>	Defines metadata for the web app, including its name, icons, start URL, and display mode.	Defines essential information about the Android app, such as its package name, version, permissions, and activities.
<b>File Extension</b>	Uses a <code>.webmanifest</code> file extension.	Uses an <code>.xml</code> file extension.
<b>Location</b>	Typically located at the root of the web app directory.	Located in the <code>app/src/main</code> directory of the Android Studio project.
<b>Syntax</b>	Uses JSON syntax.	Uses XML syntax.
<b>Key Properties</b>	<ul style="list-style-type: none"> <li>- <code>name</code>: The full name of the app. &lt;br&gt;</li> <li>- <code>short_name</code>: A short name for the app. &lt;br&gt;</li> <li>- <code>start_url</code>: The URL that loads when the app is launched. &lt;br&gt;</li> <li>- <code>display</code>: Specifies how the app should be displayed (e.g., fullscreen, standalone). &lt;br&gt;</li> <li>- <code>icons</code>: Specifies icons for the app.</li> </ul>	<ul style="list-style-type: none"> <li>- <code>package</code>: The unique package name of the app. &lt;br&gt;</li> <li>- <code>versionCode</code>: An integer value that represents the version code of the app. &lt;br&gt;</li> <li>- <code>versionName</code>: A string value that represents the version name of the app. &lt;br&gt;</li> <li>- <code>uses-permission</code>: Specifies permissions required by the app. &lt;br&gt;</li> <li>- <code>application</code>: Contains information about the app's activities, services, and other components.</li> </ul>
<b>Usage</b>	Used to configure various aspects of the web app, such as its appearance and behavior.	Used to define the structure and behavior of the Android app, including its components and permissions.

Ques18) Explain the working of Background Sync for Offline Apps with Service Workers?

Ans)

Background Sync is a feature of service workers that allows web applications to synchronize data in the background, even when the application is not actively in use or the browser is closed. This feature is particularly useful for offline apps that need to periodically sync data with a server.

Here's how Background Sync works with service workers:

1. Registration: The web application registers a service worker that includes a sync event listener. This listener is triggered when a sync event occurs.
2. Request for Sync: When the application is online, it can request a sync event using the sync event tag and the registerSync method. For example, the application might request a sync event when a user saves data locally that needs to be synchronized with the server.
3. Background Sync: When the browser determines that it has a stable network connection and is suitable for syncing, it queues the sync event. The service worker then wakes up and can perform tasks such as fetching data from the server or sending data that was saved locally.

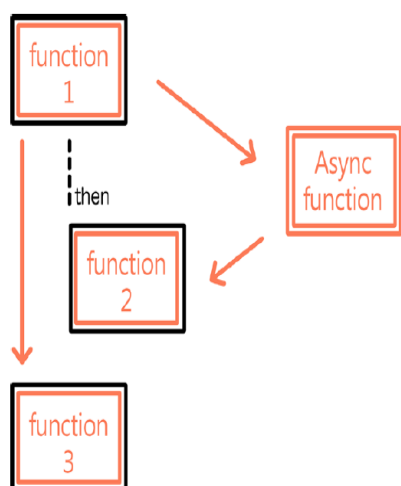
4. Data Synchronization: The service worker can use the Fetch API to make requests to the server and synchronize data. It can also update the local cache with any new data received from the server.
5. Completion: Once the sync event is complete, the service worker goes back to sleep, and the browser resumes its normal operation.

#### Ques20) Working with Promises?

Ans)

- JavaScript is single threaded
- When the app makes an API request, it's going to move on to the next line of code, not waiting for that request to finish
- Need some kind of mechanism to process the result of that API request
- Can use callback function – sometimes - nested several times, leading to **callback hell**
- Promises fix this problem by telling the asynchronous method that it “promises” to call a given function as soon as the async one is finished

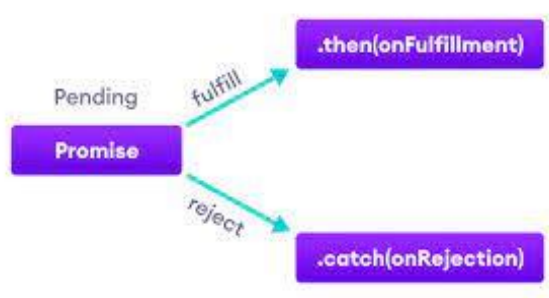
Execution order while using Promises



function1 could make an API call and then go right on to call

function2, even though function3 appears next sequentially. Once the asynchronous function is finished, *then*

function3 will execute



A Promise is in one of these states:

**pending:** initial state, neither fulfilled nor rejected

**fulfilled:** meaning that the operation was completed successfully

**rejected:** meaning that the operation failed

#### Ques21) Fetch API -

- Fetch is a native web platform API that allows you to make network requests that return promises
- The Fetch API interface allows web browser to make HTTP requests to web servers
- The fetch() requires only one parameter which is the URL of the resource that you want to fetch:  

```
let response = fetch(url);
```
- The fetch() method returns a Promise so you can use the then() and catch() methods to handle it:

```
fetch(url)  
.then(response => {  
    // handle the response  
})  
.catch(error => {  
    // handle the error  
});
```

Example -2

```
((() => {  
    fetch('https://opentdb.com/api.php?amount=1')  
    .then((response) => {  
        return response.json();  
    })  
    .then((data) => {  
        alert(data.results[0].question);  
        alert(data.results[0].correct_answer);  
    })  
    .catch((err) => {  
        alert(err);  
    });  
});
```

- don't need a separate function for the async call because that's essentially what it is doing
- Also don't need to create a promise this time; the fetch call does that
- Not going to call explicitly as - An IIFE is an immediately invoked function expression – () at end
- dropped the function keyword in place of arrow functions