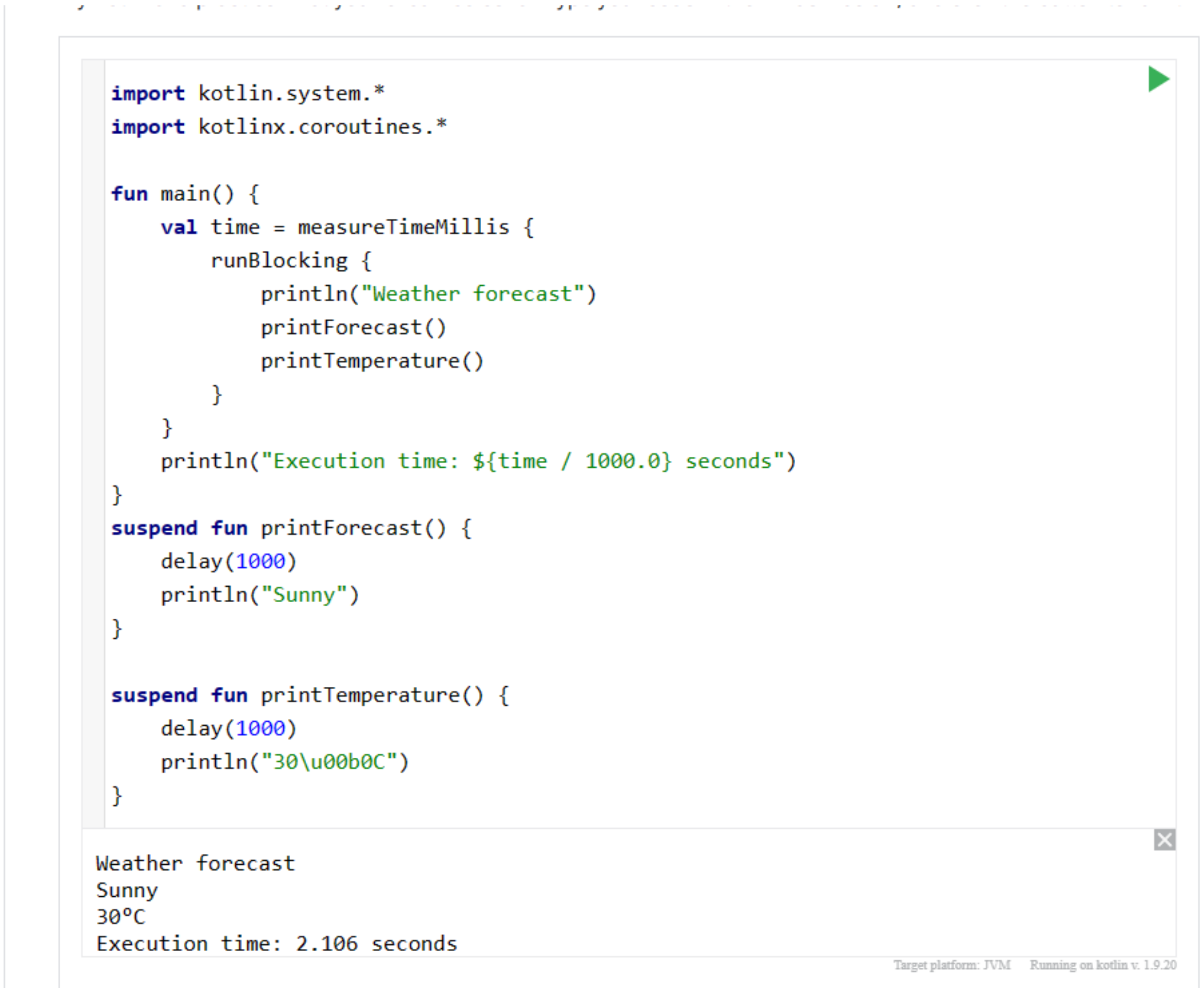
Introduction to Coroutines in Kotlin Playground

Synchronous code



```kotlin
import kotlin.system.*
import kotlinx.coroutines.*

fun main() {
    val time = measureTimeMillis {
        runBlocking {
            println("Weather forecast")
            printForecast()
            printTemperature()
        }
    }
    println("Execution time: ${time / 1000.0} seconds")
}
suspend fun printForecast() {
    delay(1000)
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000)
    println("30\u00b0C")
}
```

```
Weather forecast
Sunny
30°C
Execution time: 2.106 seconds
```

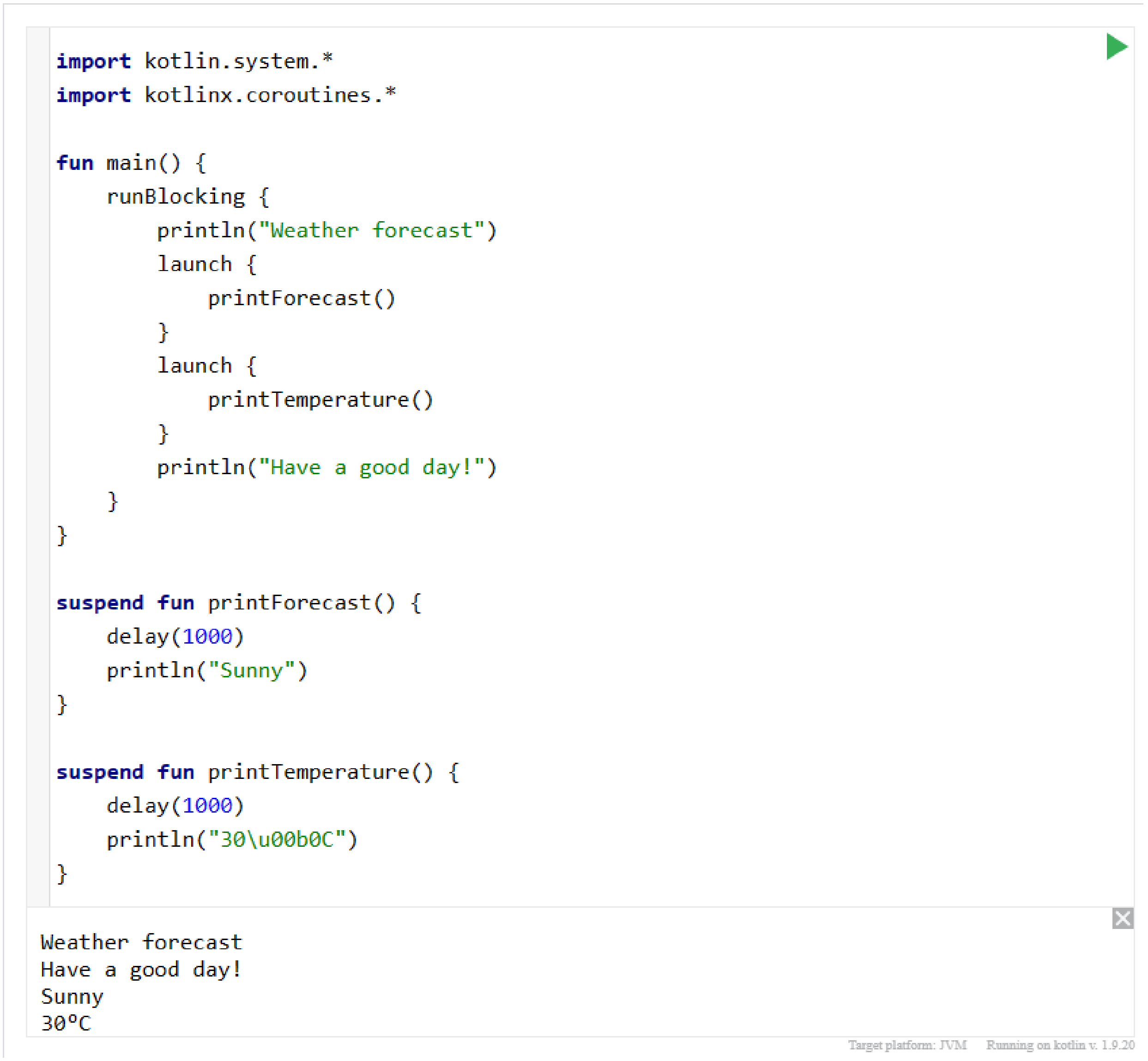
Asynchronous code

Test Launch()

```kotlin
import kotlin.system.*
import kotlinx.coroutines.*

fun main() {
    val time = measureTimeMillis {
        runBlocking {
            println("Weather forecast")
            launch {
                printForecast()
            }
            launch {
                printTemperature()
            }
        }
    }
    println("Execution time: ${time / 1000.0} seconds")
}

suspend fun printForecast() {
    delay(1000)
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000)
    println("30\u00b0C")
}
```

```
Weather forecast
Sunny
30°C
Execution time: 1.116 seconds
```

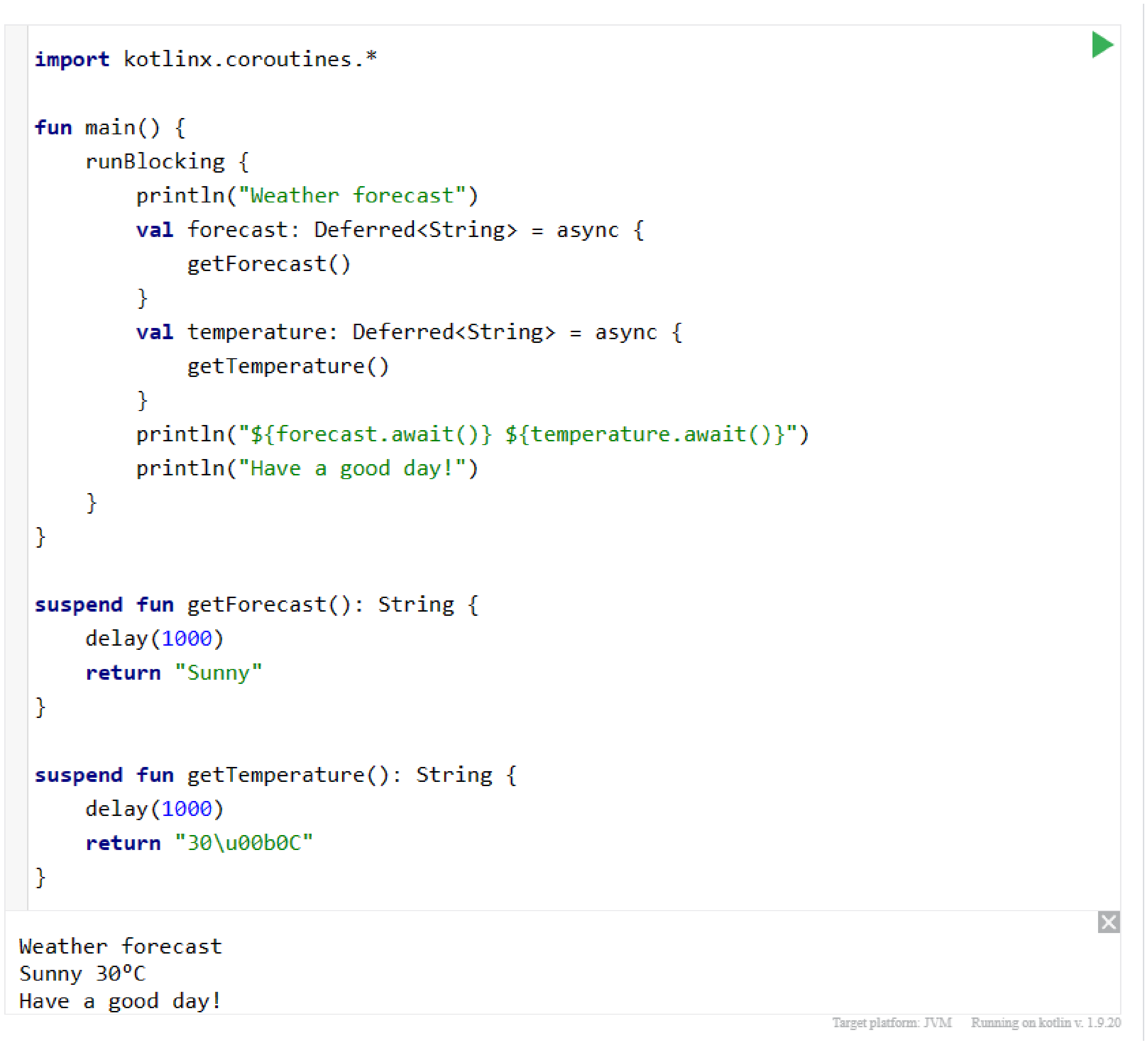
Modify the `runBlocking()` code to add an additional print statement before the end of that block.



```kotlin
import kotlin.system.*
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        launch {
            printForecast()
        }
        launch {
            printTemperature()
        }
        println("Have a good day!")
    }
}

suspend fun printForecast() {
    delay(1000)
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000)
    println("30\u00b0C")
}
```

```
Weather forecast
Have a good day!
Sunny
30°C
```

Test async()



```kotlin
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        val forecast: Deferred<String> = async {
            getForecast()
        }
        val temperature: Deferred<String> = async {
            getTemperature()
        }
        println("${forecast.await()} ${temperature.await()}")
        println("Have a good day!")
    }
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(1000)
    return "30\u00b0C"
}
```

```
Weather forecast
Sunny 30°C
Have a good day!
```

Parallel Decomposition

```kotlin
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async { getTemperature() }
    "${forecast.await()} ${temperature.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(1000)
    return "30\u00b0C"
}
```

```
Weather forecast
Sunny 30°C
Have a good day!
```

Target platform: JVM Running on kotlin v.

Exceptions and cancellation

Exceptions **with coroutines**

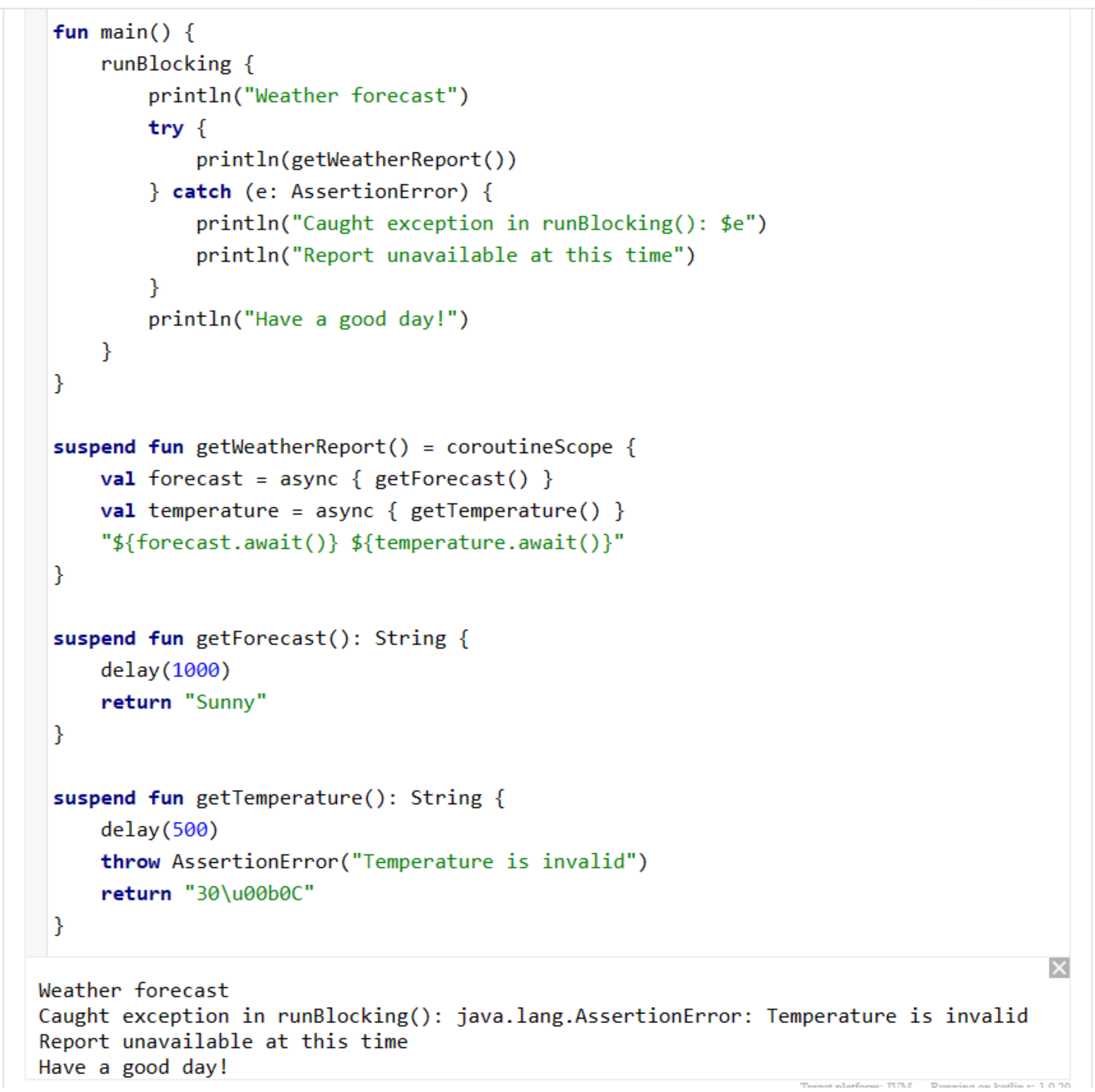
```kotlin
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async { getTemperature() }
    "${forecast.await()} ${temperature.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(500)
    throw AssertionError("Temperature is invalid")
    return "30\u00b0C"
}
```

```
Weather forecast
Exception in thread "main" java.lang.AssertionError: Temperature is invalid
   at FileKt.getTemperature (File.kt:24)
   at FileKt$getTemperature$1.invokeSuspend (File.kt:-1)
   at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith
(ContinuationImpl.kt:33)
```

Target platform: JVM   Running on kotlin v. 1.9.20

Try-catch exceptions

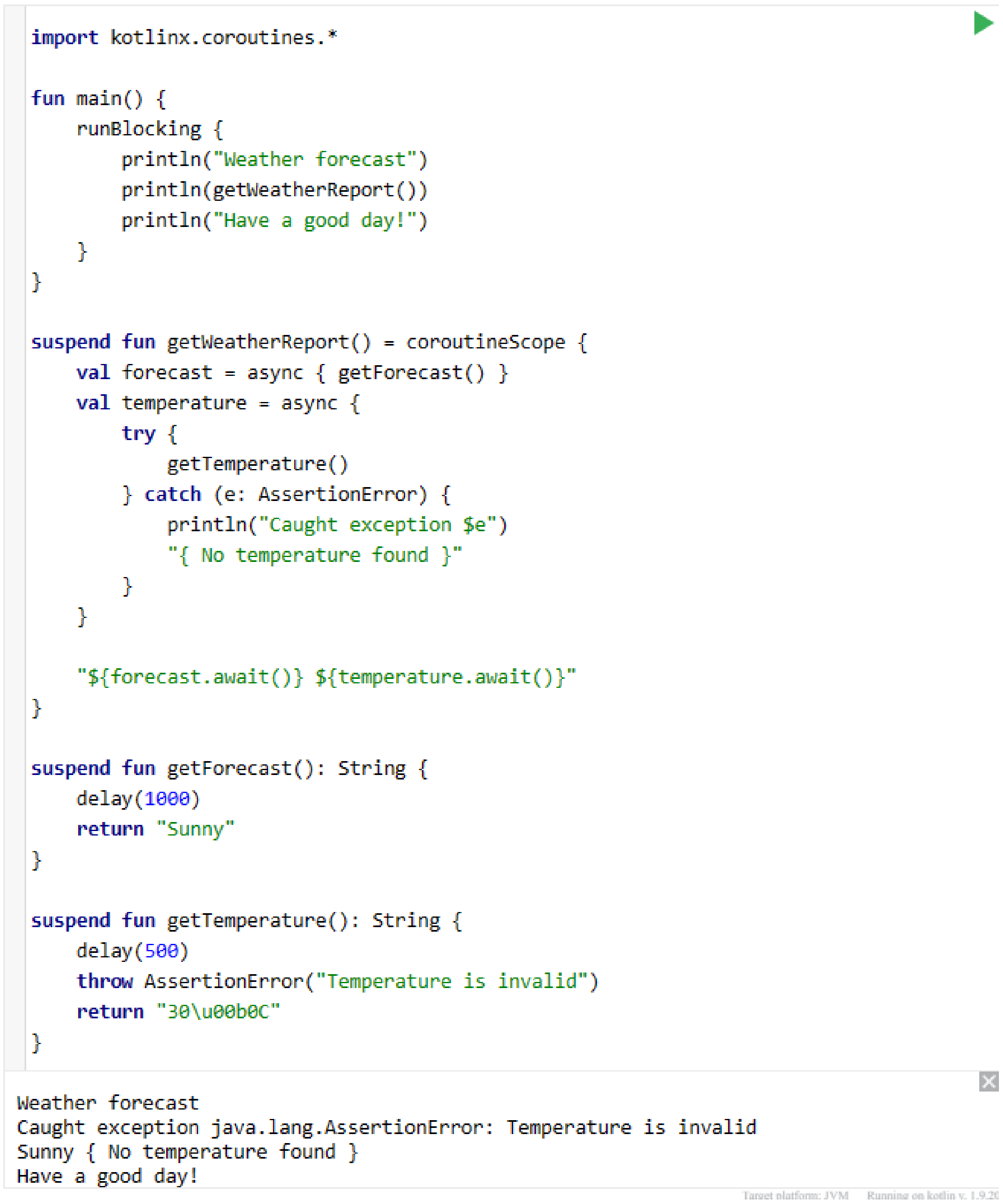
Within the `runBlocking()` function, add a try-catch block around the code that calls `getWeatherReport()`. Print out the error that is caught and also print out a message that the weather report is not available.



```kotlin
fun main() {
    runBlocking {
        println("Weather forecast")
        try {
            println(getWeatherReport())
        } catch (e: AssertionError) {
            println("Caught exception in runBlocking(): $e")
            println("Report unavailable at this time")
        }
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async { getTemperature() }
    "${forecast.await()} ${temperature.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(500)
    throw AssertionError("Temperature is invalid")
    return "30\u00b0C"
}
```

```
Weather forecast
Caught exception in runBlocking(): java.lang.AssertionError: Temperature is invalid
Report unavailable at this time
Have a good day!
```

Move the error handling so that the try-catch behavior actually happens within the coroutine launched by `async()` to fetch the temperature. That way, the weather report can still print the forecast, even if the temperature failed.



```kotlin
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async {
        try {
            getTemperature()
        } catch (e: AssertionError) {
            println("Caught exception $e")
            "{ No temperature found }"
        }
    }

    "${forecast.await()} ${temperature.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(500)
    throw AssertionError("Temperature is invalid")
    return "30\u00b0C"
}
```

```
Weather forecast
Caught exception java.lang.AssertionError: Temperature is invalid
Sunny { No temperature found }
Have a good day!
```

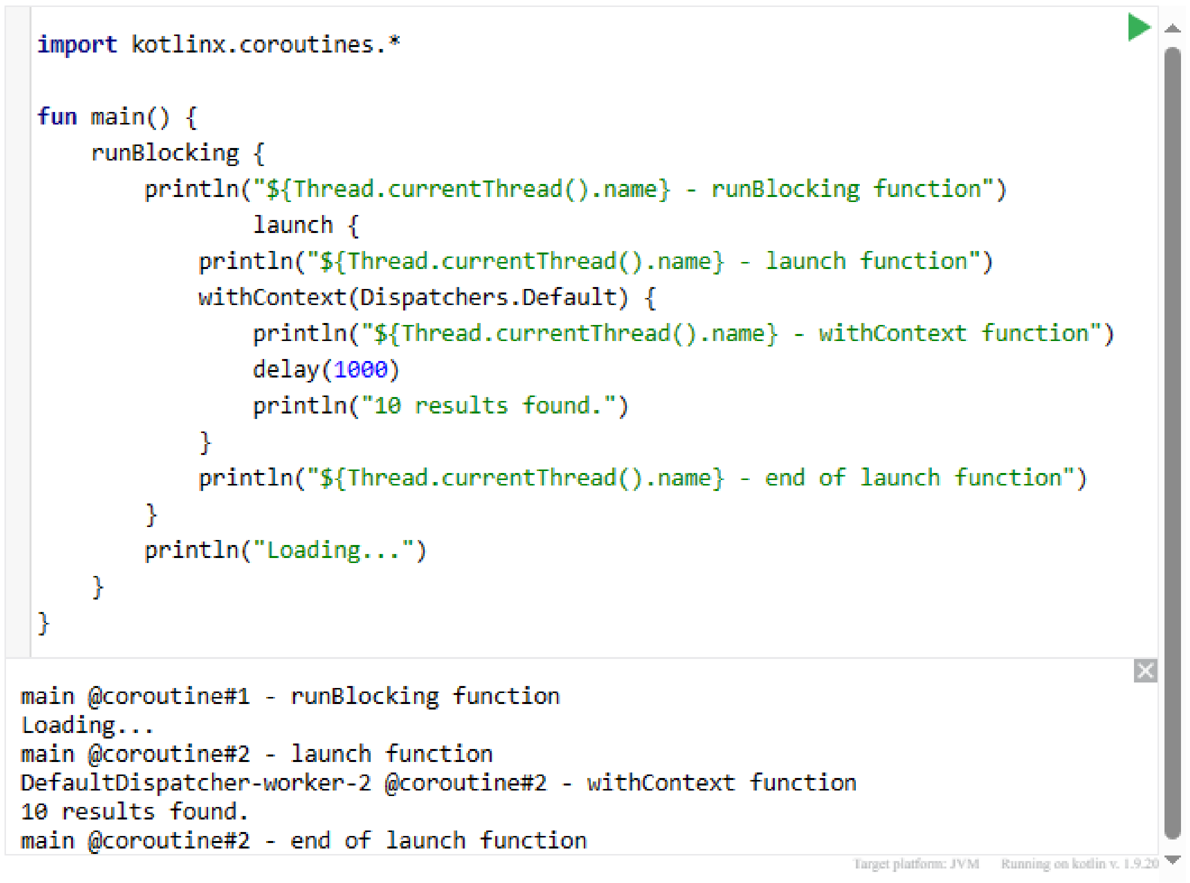
Cancellation

```kotlin
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        println(getWeatherReport())
        println("Have a good day!")
    }
}

suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async { getTemperature() }

    delay(200)
    temperature.cancel()

    "${forecast.await()}"
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(1000)
    return "30\u00b0C"
}
```

```
Weather forecast
Sunny
Have a good day!
```

Target platform: JVM   Running on kotlin v. 1.9.20

Coroutine concepts



```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("${Thread.currentThread().name} - runBlocking function")
                launch {
            println("${Thread.currentThread().name} - launch function")
            withContext(Dispatchers.Default) {
                println("${Thread.currentThread().name} - withContext function")
                delay(1000)
                println("10 results found.")
            }
            println("${Thread.currentThread().name} - end of launch function")
        }
        println("Loading...")
    }
}
```

```
main @coroutine#1 - runBlocking function
Loading...
main @coroutine#2 - launch function
DefaultDispatcher-worker-2 @coroutine#2 - withContext function
10 results found.
main @coroutine#2 - end of launch function
```

Introduction to Coroutines in Android Studio

Implement race progress

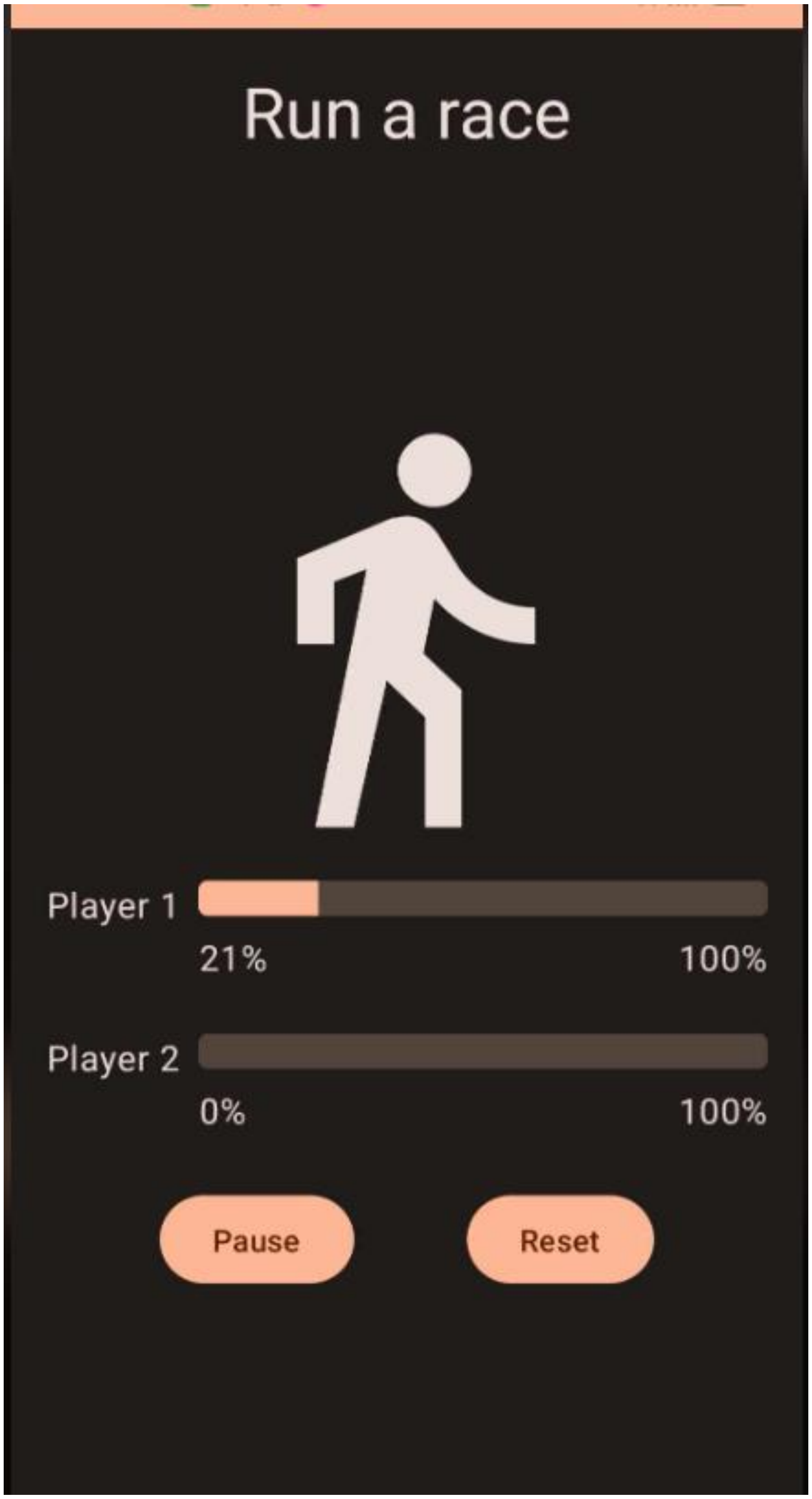
-Update new function at RaceParticipant class and use delay() To simulate different progress intervals in the race

```
suspend fun run() {
    while (currentProgress < maxProgress) {
        delay(progressDelayMillis)
        currentProgress += progressIncrement
    }
}
```

Start the race

Create LaunchedEffect() with if statement and start the app to check

```kotlin
if (raceInProgress) {
    LaunchedEffect(playerOne, playerTwo) {
        playerOne.run()
        playerTwo.run()
        raceInProgress = false
    }
}
```

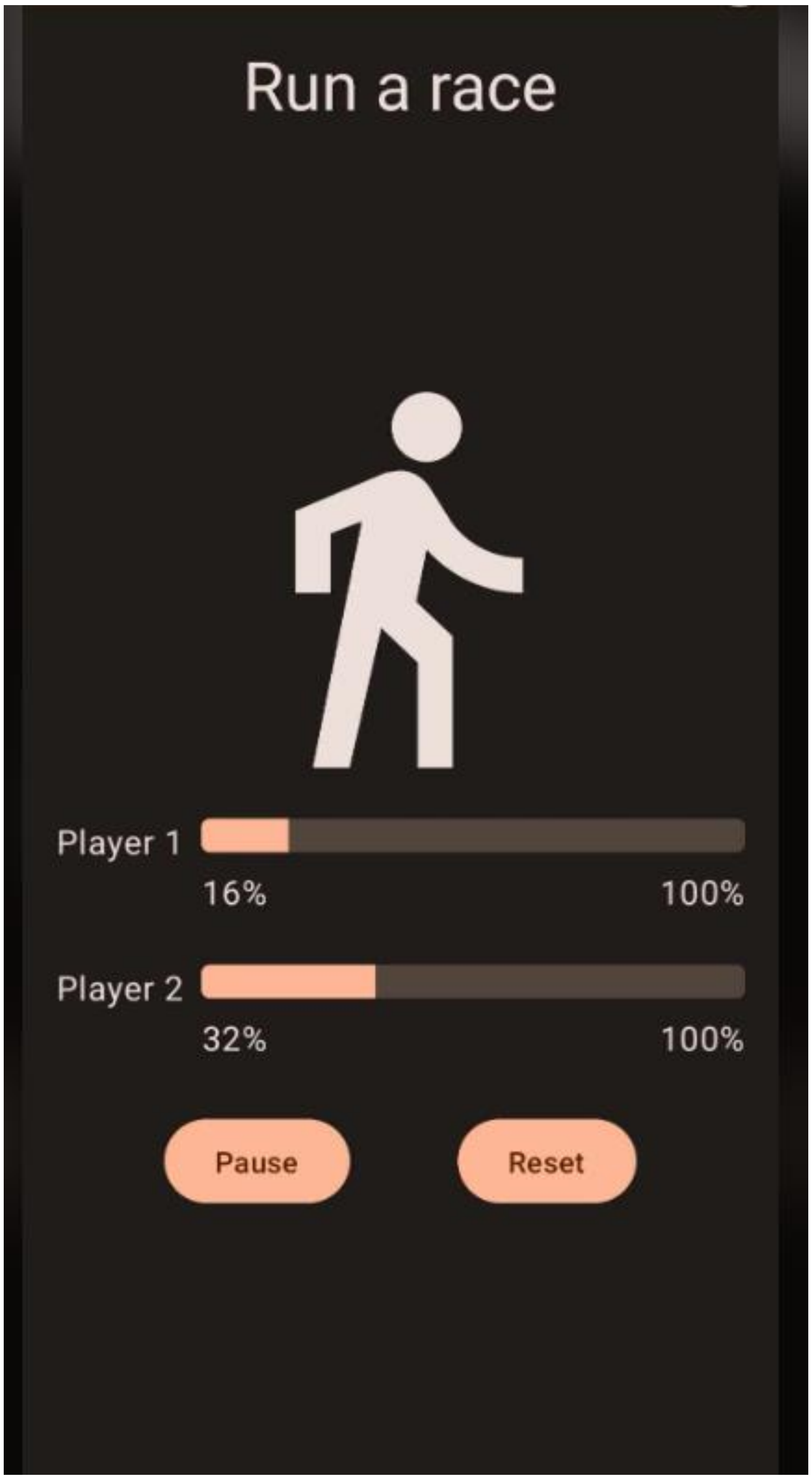


Run the app and see that only player 1 change the value but player 2 didn't

Launch concurrent tasks

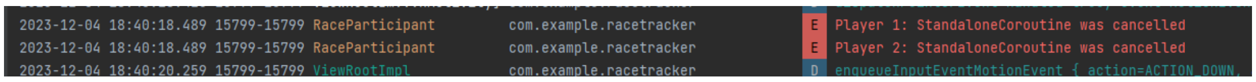
Using launch builder and Coroutine Scope for the execution

```
if (raceInProgress) {
    LaunchedEffect(playerOne, playerTwo) {
        coroutineScope {
            launch { playerOne.run() }
            launch { playerTwo.run() }
        }
        raceInProgress = false
    }
}
```



Run the app and see that player 2 now works and runs faster than 1 (according to the lab result) also upgrade run function in the previous class by using try-catch

```kotlin
suspend fun run() {
    try {
        while (currentProgress < maxProgress) {
            delay(progressDelayMillis)
            currentProgress += progressIncrement
        }
    } catch (e: CancellationException) {
        Log.e("RaceParticipant", "$name: ${e.message}")
        throw e // Always re-throw CancellationException.
    }
}
```



```
2023-12-04 18:40:18.489 15799-15799 RaceParticipant         com.example.racetracker              E  Player 1: StandaloneCoroutine was cancelled
2023-12-04 18:40:18.489 15799-15799 RaceParticipant         com.example.racetracker              E  Player 2: StandaloneCoroutine was cancelled
2023-12-04 18:40:20.259 15799-15799 ViewRootImpl            com.example.racetracker              D  enqueueInputEventMotionEvent { action=ACTION_DOWN,
```

It will show how coroutines are canceled when the user clicks the **Reset** button

Write unit tests to test coroutines

Add testImplementation("org.jetbrains.kotlinx:kotlinx-coroutines-test:1.6.4") into build.gradle.kts(app) (at the dependency)

Add new code line for the RaceParticipantTest and test it

```kotlin
package com.example.racetracker

import com.example.racetracker.ui.RaceParticipant
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.cancelAndJoin
import kotlinx.coroutines.launch
import kotlinx.coroutines.test.advanceTimeBy
import kotlinx.coroutines.test.runCurrent
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Test

@OptIn(ExperimentalCoroutinesApi::class)
class RaceParticipantTest {
    private val raceParticipant = RaceParticipant(
        name = "Test",
        maxProgress = 100,
        progressDelayMillis = 500L,
        initialProgress = 0,
        progressIncrement = 1
    )

    @Test
    fun raceParticipant_RaceStarted_ProgressUpdated() = runTest {
        val expectedProgress = 1
        launch { raceParticipant.run() }
        advanceTimeBy(raceParticipant.progressDelayMillis)
        runCurrent()
        assertEquals(expectedProgress, raceParticipant.currentProgress)
    }

    @Test
    fun raceParticipant_RaceFinished_ProgressUpdated() = runTest {
        launch { raceParticipant.run() }

        advanceTimeBy(raceParticipant.maxProgress *
raceParticipant.progressDelayMillis)
        runCurrent()
        assertEquals(100, raceParticipant.currentProgress)
    }

    @Test
    fun raceParticipant_RacePaused_ProgressUpdated() = runTest {
        val expectedProgress = 5
        val racerJob = launch { raceParticipant.run() }
        advanceTimeBy(expectedProgress * raceParticipant.progressDelayMillis)
        runCurrent()
        racerJob.cancelAndJoin()
        assertEquals(expectedProgress, raceParticipant.currentProgress)
    }

    @Test
    fun raceParticipant_RacePausedAndResumed_ProgressUpdated() = runTest {
        val expectedProgress = 5

        repeat(2) {
            val racerJob = launch { raceParticipant.run() }
            advanceTimeBy(expectedProgress *
raceParticipant.progressDelayMillis)
            runCurrent()
            racerJob.cancelAndJoin()
        }

        assertEquals(expectedProgress * 2, raceParticipant.currentProgress)
    }

    @Test(expected = IllegalArgumentException::class)
    fun raceParticipant_ProgressIncrementZero_ExceptionThrown() = runTest {
        RaceParticipant(name = "Progress Test", progressIncrement = 0)
    }

    @Test(expected = IllegalArgumentException::class)
    fun raceParticipant_MaxProgressZero_ExceptionThrown() {
        RaceParticipant(name = "Progress Test", maxProgress = 0)
    }
}
```

Get data from the internet

Web services and Retrofit

Add Retrofit dependencies

```
dependencies {

    // Import the Compose BOM
    implementation(platform("androidx.compose:compose-bom:2023.06.01"))
    implementation("androidx.activity:activity-compose:1.7.2")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.core:core-ktx:1.10.1")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.6.1")

    debugImplementation("androidx.compose.ui:ui-test-manifest")
    debugImplementation("androidx.compose.ui:ui-tooling")
    // Retrofit
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
// Retrofit with Scalar Converter
    implementation("com.squareup.retrofit2:converter-scalars:2.9.0")
}
```

Connecting to the Internet

Create MarsApiService.kt with following code for the retrofit setup:

```
package com.example.marsphotos.network

import retrofit2.http.GET
import retrofit2.Retrofit
import retrofit2.converter.scalars.ScalarsConverterFactory

private const val BASE_URL =
    "https://android-kotlin-fun-mars-server.appspot.com"

private val retrofit = Retrofit.Builder()
    .addConverterFactory(ScalarsConverterFactory.create())
    .baseUrl(BASE_URL)
    .build()

interface MarsApiService {
    @GET("photos")
    fun getPhotos(): String
}

object MarsApi {
    val retrofitService : MarsApiService by lazy {
        retrofit.create(MarsApiService::class.java)
    }
}
```

Call the web service in MarsViewModel

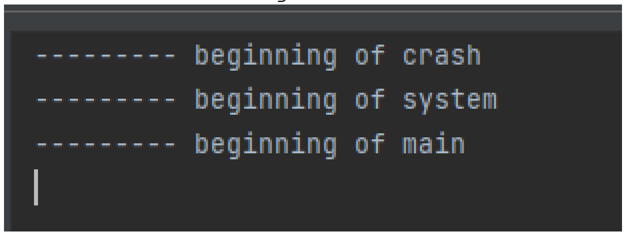
In the `MarsApiService.kt` file, make `getPhotos()` a suspend function to make it asynchronous and not block the calling thread. You call this function from inside a [viewModelScope](https://developer.android.com/topic/libraries/architecture/coroutines#viewmodelscope)

```
interface MarsApiService {
    @GET("photos")
    suspend fun getPhotos(): String
}
```

Update MarsviewModel.kt

```
/*
 * Copyright (C) 2023 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.example.marsphotos.ui.screens

import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch
import com.example.marsphotos.network.MarsApi

class MarsViewModel : ViewModel() {
    /** The mutable State that stores the status of the most recent request
*/
    var marsUiState: String by mutableStateOf("")
        private set

    /**
     * Call getMarsPhotos() on init so we can display status immediately.
     */
    init {
        getMarsPhotos()
    }

    /**
     * Gets Mars photos information from the Mars API Retrofit service and
updates the
     * [MarsPhoto] [List] [MutableList].
     */
    fun getMarsPhotos() {
        viewModelScope.launch {
            val listResult = MarsApi.retrofitService.getPhotos()
            marsUiState = listResult
        }


    }
}
```

```
And check the logcat now
```

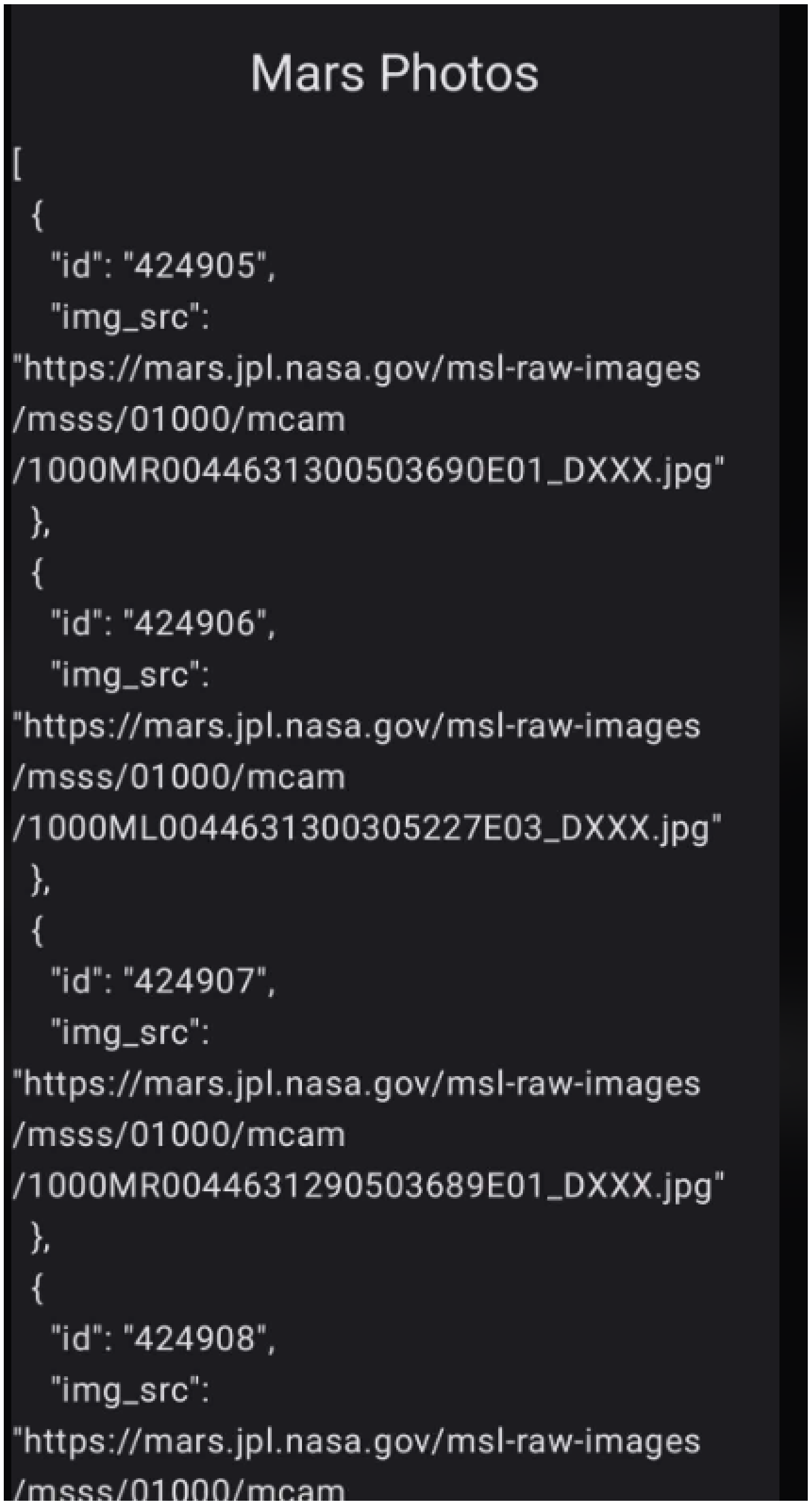


```
--------- beginning of crash
--------- beginning of system
--------- beginning of main
```

This error message indicates the app might be missing the `INTERNET` permissions. The next task describes how to add internet permissions to the app and resolve this issue.

Add Internet permission and Exception Handling

Open `manifests/AndroidManifest.xml`. Add this line just before the `<application>` tag. Compile and run it :



Mars Photos

```
[
  {
    "id": "424905",
    "img_src":
"https://mars.jpl.nasa.gov/msl-raw-images
/msss/01000/mcam
/1000MR0044631300503690E01_DXXX.jpg"
  },
  {
    "id": "424906",
    "img_src":
"https://mars.jpl.nasa.gov/msl-raw-images
/msss/01000/mcam
/1000ML0044631300305227E03_DXXX.jpg"
  },
  {
    "id": "424907",
    "img_src":
"https://mars.jpl.nasa.gov/msl-raw-images
/msss/01000/mcam
/1000MR0044631290503689E01_DXXX.jpg"
  },
  {
    "id": "424908",
    "img_src":
"https://mars.jpl.nasa.gov/msl-raw-images
/msss/01000/mcam
```

Exception Handling

This session require to open airplane mode. However, I’m using directly via my mobile so there is some result cannot show.

Exceptions

At MarsViewModel.kt

Re-update fun

```
fun getMarsPhotos() {
    viewModelScope.launch {
        try {
            val listResult = MarsApi.retrofitService.getPhotos()
            marsUiState = listResult
        } catch (e: IOException) {

        }
    }

}
```

Add State UI

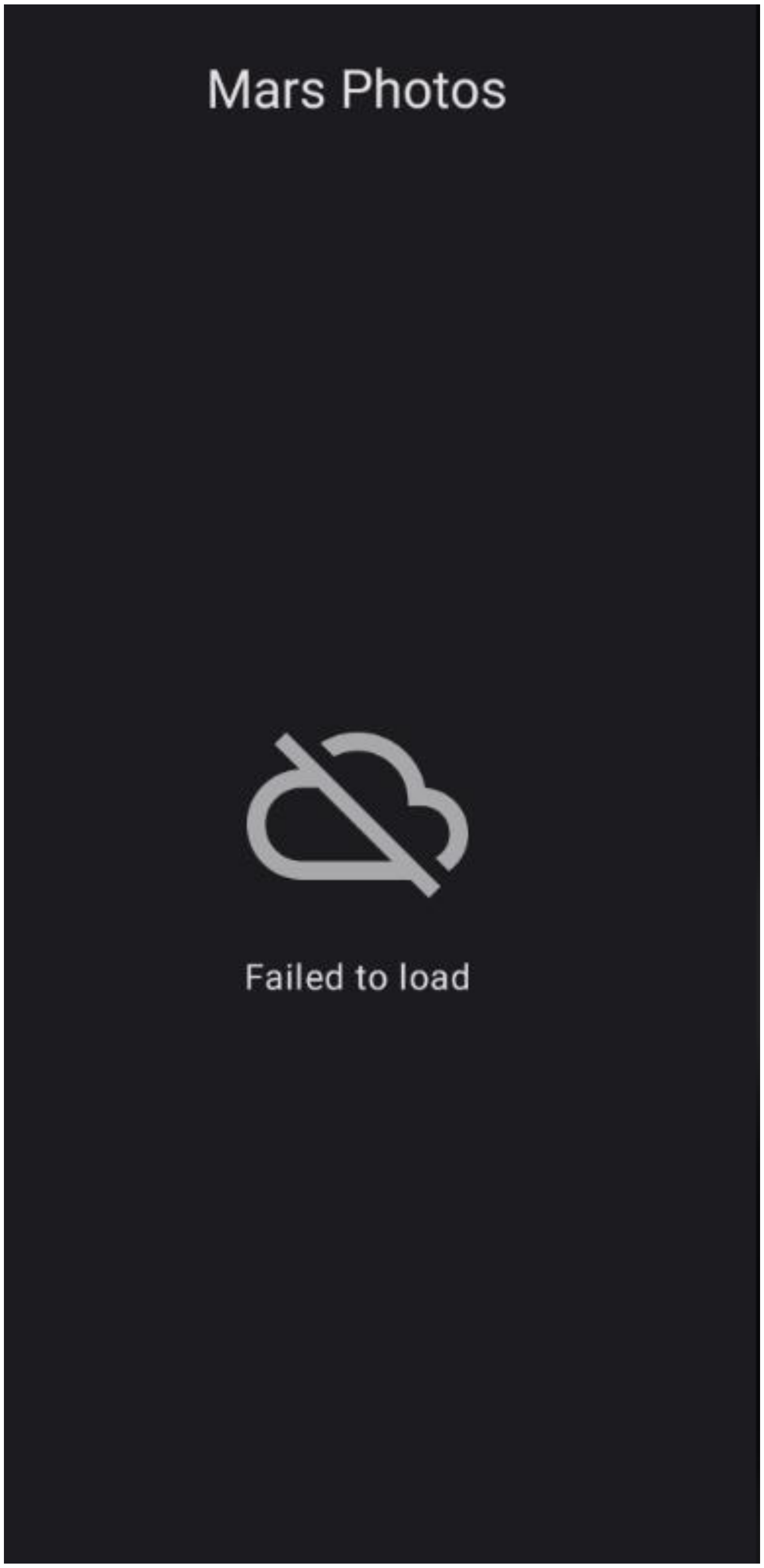
```
/*
 * Copyright (C) 2023 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.example.marsphotos.ui.screens

import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch
import com.example.marsphotos.network.MarsApi
import java.io.IOException

sealed interface MarsUiState {
    data class Success(val photos: String) : MarsUiState
    object Error : MarsUiState
    object Loading : MarsUiState
}

class MarsViewModel : ViewModel() {
    /** The mutable State that stores the status of the most recent request
*/
```

```
    var marsUiState: MarsUiState by mutableStateOf(MarsUiState.Loading)
        private set

    /**
     * Call getMarsPhotos() on init so we can display status immediately.
     */
    init {
        getMarsPhotos()
    }

    /**
     * Gets Mars photos information from the Mars API Retrofit service and
updates the
     * [MarsPhoto] [List] [MutableList].
     */
    private fun getMarsPhotos() {
        viewModelScope.launch {
            marsUiState = try {
                val listResult = MarsApi.retrofitService.getPhotos()
                MarsUiState.Success(listResult)
            } catch (e: IOException) {
                MarsUiState.Error
            }
        }
    }

}
```

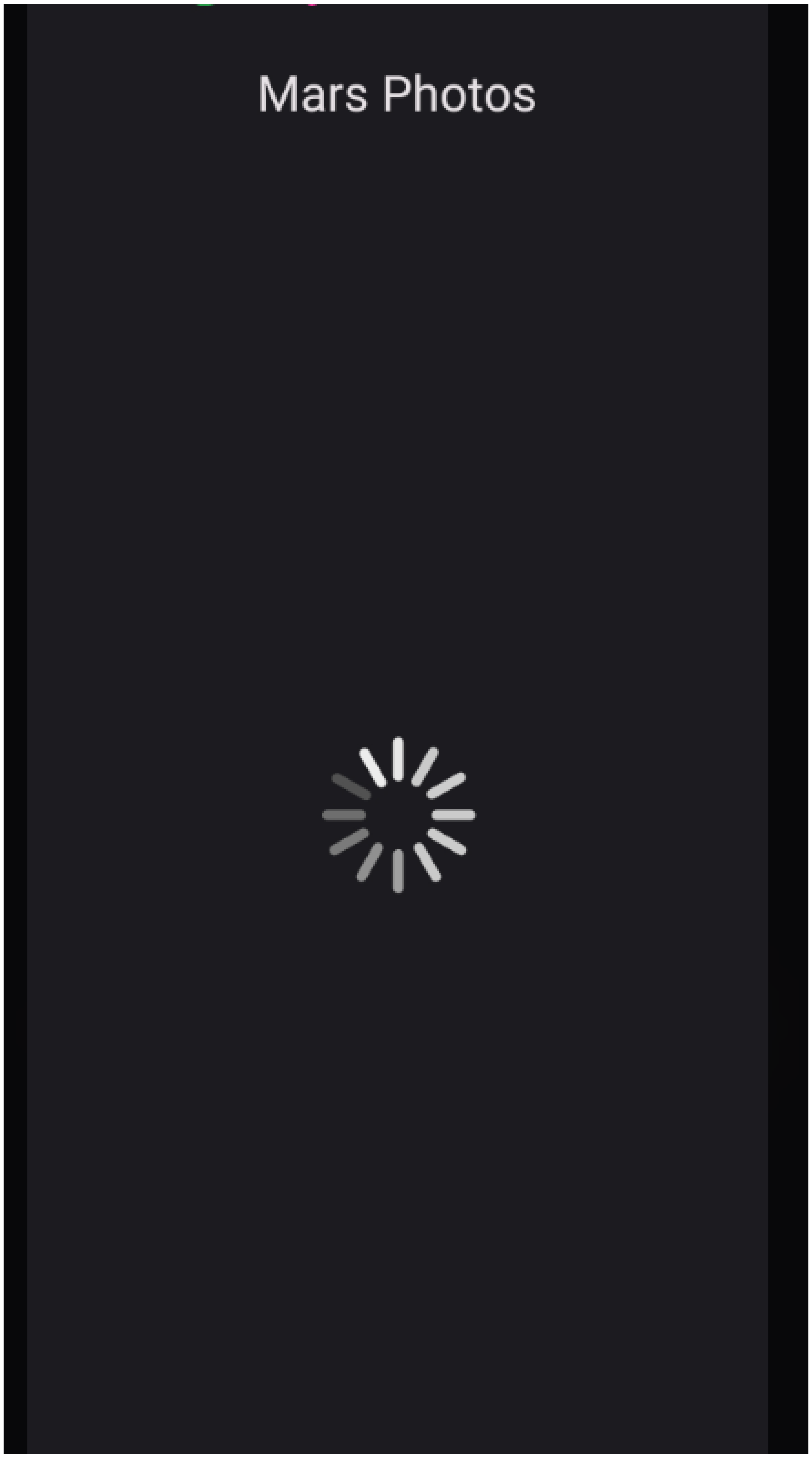
HomeScreen.kt

```
package com.example.marsphotos.ui.screens

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.marsphotos.R
import com.example.marsphotos.ui.theme.MarsPhotosTheme

@Composable
fun HomeScreen(
    marsUiState: MarsUiState, modifier: Modifier = Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier =
modifier.fillMaxSize())
        is MarsUiState.Success -> ResultScreen(
            marsUiState.photos, modifier = modifier.fillMaxWidth()
        )

        is MarsUiState.Error -> ErrorScreen( modifier =
modifier.fillMaxSize())
    }
}


/**
 * ResultScreen displaying number of photos retrieved.
 */
@Composable
fun ResultScreen(photos: String, modifier: Modifier = Modifier) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
    ) {
        Text(text = photos)
    }
}

@Preview(showBackground = true)
@Composable
fun ResultScreenPreview() {
    MarsPhotosTheme {
        ResultScreen(stringResource(R.string.placeholder_result))
    }
}
@Composable
fun LoadingScreen(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier.size(200.dp),
        painter = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.loading)
    )
}
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error),
contentDescription = ""
        )
        Text(text = stringResource(R.string.loading_failed), modifier =
Modifier.padding(16.dp))
    }
}
```

run app
when open airplane mode



When turn off



[Parse the JSON response with kotlinx.serialization]

Add kotlinx.serialization library dependencies

```
@file:Suppress("UnstableApiUsage")

plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("org.jetbrains.kotlin.plugin.serialization") version "1.8.10"
}

android {
    namespace = "com.example.marsphotos"
    compileSdk = 33

    defaultConfig {
        applicationId = "com.example.marsphotos"
        minSdk = 24
        targetSdk = 33
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary = true
        }
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
    buildFeatures {
        compose = true
    }
    composeOptions {
        kotlinCompilerExtensionVersion = "1.4.7"
    }
    packaging {
        resources {
            excludes += "/META-INF/{AL2.0,LGPL2.1}"
        }
    }
}
```

```
dependencies {

    // Import the Compose BOM
    implementation(platform("androidx.compose:compose-bom:2023.06.01"))
    implementation("androidx.activity:activity-compose:1.7.2")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.core:core-ktx:1.10.1")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.6.1")

    debugImplementation("androidx.compose.ui:ui-test-manifest")
    debugImplementation("androidx.compose.ui:ui-tooling")
    // Retrofit
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
// Retrofit with Kotlin serialization Converter

    implementation("com.jakewharton.retrofit:retrofit2-kotlinx-serialization-
converter:1.0.0")
    implementation("com.squareup.okhttp3:okhttp:4.11.0")
    // Kotlin serialization
    implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.5.1")
}
```

Implement the Mars Photo data class

Create MarsPhoto.kt

```
package com.example.marsphotos.model

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable

/**
 * This data class defines a Mars photo which includes an ID, and the image
URL.
 */
@Serializable
data class MarsPhoto(
    val id: String,
    @SerialName(value = "img_src")
    val imgSrc: String
)
```

Update MarsApiService and MarsViewModel

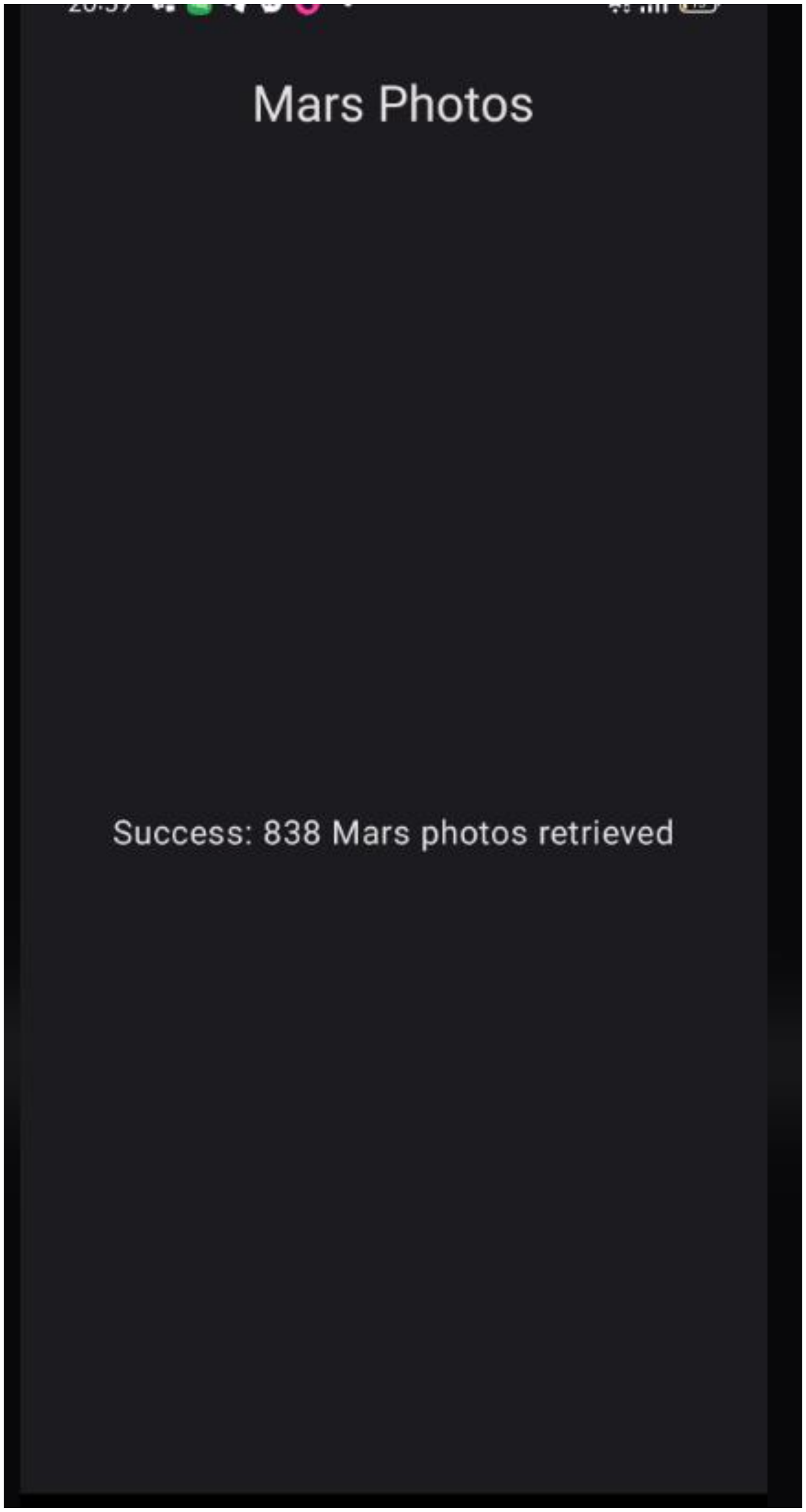
Marviewmodel.kt update

```
fun getMarsPhotos() {
    viewModelScope.launch {
        marsUiState = MarsUiState.Loading
        marsUiState = try {
            val listResult = MarsApi.retrofitService.getPhotos()
            MarsUiState.Success(
                "Success: ${listResult.size} Mars photos retrieved"
            )
        } catch (e: IOException) {
            MarsUiState.Error
        } catch (e: HttpException) {
            MarsUiState.Error
        }
    }
}
```

marsapiservice.kt update

```
package com.example.marsphotos.network

import retrofit2.http.GET
import retrofit2.Retrofit
import
com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterFactory
import kotlinx.serialization.json.Json
import okhttp3.MediaType
import okhttp3.MediaType.Companion.toMediaType

private const val BASE_URL =
    "https://android-kotlin-fun-mars-server.appspot.com"

private val retrofit = Retrofit.Builder()

.addConverterFactory(Json.asConverterFactory("application/json".toMediaType()
))
    .baseUrl(BASE_URL)
    .build()

interface MarsApiService {
    @GET("photos")
    suspend fun getPhotos(): List<MarsPhoto>
}

object MarsApi {
    val retrofitService : MarsApiService by lazy {
        retrofit.create(MarsApiService::class.java)
    }
}
```

Result


Mars Photos

Success: 838 Mars photos retrieved

Quiz p1

1. With concurrent programming, code might execute in an order different from how it was written.

- [✓ True] ✓ Correct!
- False

2. Fill-in-the-blanks

Enter one or more words to complete the sentence.

The `onBackPressedDispatcher` composable is used to respond to the Back button, with or without a NavHost.

Incorrect.

3. Which of the following statements are true about coroutine contexts?

Choose as many answers as you see fit.

- [ ] `Dispatchers.Default` is the best choice for long running tasks involving reading and writing large amounts of data.
- [✓] `Dispatchers.Main` can be used for updating the UI but not for long-running tasks. ✓ Correct!
- [✓] A `Job` controls the lifecycle of a coroutine. ✓ Correct!
- [✓] `Dispatchers.IO` is optimized for network I/O, among other background tasks. ✓ Correct!

4. `launch()` and `async()` are extension functions of a ___, which keeps track of any coroutines it creates.

- ● `CoroutineScope` ✓ Correct!
- ○ `Job`
- ○ `Dispatcher`
- ○ `CoroutineContext`

5. Which of the following statements are true about structured concurrency and its best practices?

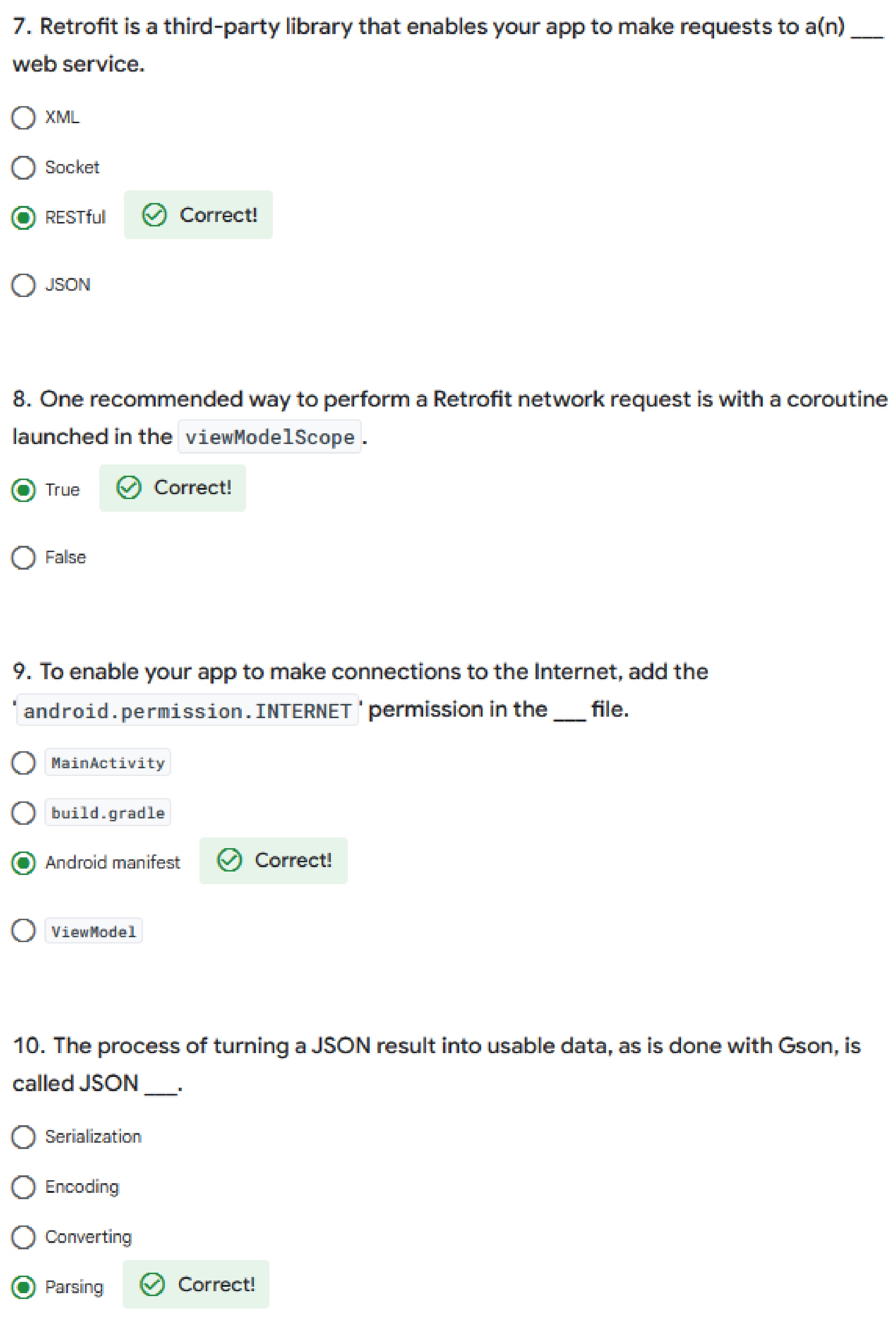
Choose as many answers as you see fit.

- [✓] If a coroutine is canceled, child coroutines should also be canceled. ✓ Correct!
- [ ] A parent scope can complete before one or more of its children are completed.
- [ ] A failure should propagate downward without canceling the parent coroutine.
- [✓] Coroutines must be launched from a coroutine scope. ✓ Correct!

6. Which of the following statements are true about web services?

Choose as many answers as you see fit.

- [✓] GET, POST, and DELETE are all examples of HTTP operations. ✓ Correct!
- [✓] A URL is a type of URI but not all URIs are URLs. ✓ Correct!
- [ ] RESTful services always provide a formatted XML response.
- [ ] Retrofit is a third-party library for handling JSON from a web service.


7. Retrofit is a third-party library that enables your app to make requests to a(n) ___ web service.

- XML
- Socket
- [✓ RESTful] Correct!
- JSON

8. One recommended way to perform a Retrofit network request is with a coroutine launched in the `viewModelScope`.

- [✓ True] Correct!
- False

9. To enable your app to make connections to the Internet, add the '`android.permission.INTERNET`' permission in the ___ file.

- `MainActivity`
- `build.gradle`
- [✓ Android manifest] Correct!
- `ViewModel`

10. The process of turning a JSON result into usable data, as is done with Gson, is called JSON ___.

- Serialization
- Encoding
- Converting
- [✓ Parsing] Correct!

Add repository and Manual DI

Create Data layer

Create repository

`MarsPhotosRepository`.kt

```
package com.example.marsphotos.data

import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.network.MarsApi

interface MarsPhotosRepository {
    suspend fun getMarsPhotos(): List<MarsPhoto>
}
class NetworkMarsPhotosRepository() : MarsPhotosRepository {
    override suspend fun getMarsPhotos(): List<MarsPhoto> {
        return MarsApi.retrofitService.getPhotos()
    }
}
```

re-update `getMarsPhotos() at ui/screens/MarsViewModel.kt`

```
fun getMarsPhotos() {
        viewModelScope.launch {
            marsUiState = MarsUiState.Loading
            marsUiState = try {
                val marsPhotosRepository = NetworkMarsPhotosRepository()
                val listResult = marsPhotosRepository.getMarsPhotos()
                MarsUiState.Success(
                    "Success: ${listResult.size} Mars photos retrieved"
                )
            } catch (e: IOException) {
                MarsUiState.Error
            } catch (e: HttpException) {
                MarsUiState.Error
            }
        }
```

Dependency injection

Create an Application Container

AppContainer.kt

```
package com.example.marsphotos.data

import com.example.marsphotos.network.MarsApiService
import retrofit2.Retrofit
import
com.jakewharton.retrofit2.converter.kotlinx.serialization.asConverterFactory
import kotlinx.serialization.json.Json
import okhttp3.MediaType.Companion.toMediaType


interface AppContainer {
    val marsPhotosRepository: MarsPhotosRepository
}


class DefaultAppContainer : AppContainer {
    private val baseUrl = "https://android-kotlin-fun-mars-
server.appspot.com/"
```

```
    private val retrofit: Retrofit = Retrofit.Builder()
.addConverterFactory(Json.asConverterFactory("application/json".toMediaType()
))
        .baseUrl(baseUrl)
        .build()


    private val retrofitService: MarsApiService by lazy {
        retrofit.create(MarsApiService::class.java)
    }


    override val marsPhotosRepository: MarsPhotosRepository by lazy {
        NetworkMarsPhotosRepository(retrofitService)
    }
}
```

re-update`MarsPhotosRepository.kt`

```
package com.example.marsphotos.data

import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.network.MarsApiService

interface MarsPhotosRepository {
    suspend fun getMarsPhotos(): List<MarsPhoto>
}
class NetworkMarsPhotosRepository(
    private val marsApiService: MarsApiService
) : MarsPhotosRepository {
    override suspend fun getMarsPhotos(): List<MarsPhoto> =
marsApiService.getPhotos()
}
```

and delete this code at MarsApiService.kt

```
object MarsApi {
    val retrofitService: MarsApiService by lazy {
        retrofit.create(MarsApiService::class.java)
    }
```

Attach application container to the app

Create MarsPhotosApplication.kt

```
package com.example.marsphotos

import android.app.Application
import com.example.marsphotos.data.AppContainer
import com.example.marsphotos.data.DefaultAppContainer

class MarsPhotosApplication : Application() {
    lateinit var container: AppContainer
    override fun onCreate() {
        super.onCreate()
        container = DefaultAppContainer()
    }
}
```

add android:name=".MarsPhotosApplication" at AndroidManifest.xml

Add repository to ViewModel

Update MarsViewModel.kt

```
package com.example.marsphotos.ui.screens

import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import
androidx.lifecycle.ViewModelProvider.AndroidViewModelFactory.Companion.APPLIC
ATION_KEY
import androidx.lifecycle.viewModelScope
import androidx.lifecycle.viewmodel.initializer
import androidx.lifecycle.viewmodel.viewModelFactory
import com.example.marsphotos.MarsPhotosApplication
import com.example.marsphotos.data.MarsPhotosRepository
import com.example.marsphotos.model.MarsPhoto
import kotlinx.coroutines.launch
import retrofit2.HttpException
import java.io.IOException

/**
 * UI state for the Home screen
 */
sealed interface MarsUiState {
    data class Success(val photos: String) : MarsUiState
    object Error : MarsUiState
    object Loading : MarsUiState
}

class MarsViewModel(private val marsPhotosRepository: MarsPhotosRepository) :
ViewModel() {
    /** The mutable State that stores the status of the most recent request
*/
    var marsUiState: MarsUiState by mutableStateOf(MarsUiState.Loading)
        private set

    /**
     * Call getMarsPhotos() on init so we can display status immediately.
     */
    init {
        getMarsPhotos()
    }

    /**
     * Gets Mars photos information from the Mars API Retrofit service and
updates the
     * [MarsPhoto] [List] [MutableList].
     */
    fun getMarsPhotos() {
        viewModelScope.launch {
            marsUiState = MarsUiState.Loading
            marsUiState = try {
                val listResult = marsPhotosRepository.getMarsPhotos()
                MarsUiState.Success(
                    "Success: ${listResult.size} Mars photos retrieved"
                )
            } catch (e: IOException) {
                MarsUiState.Error
            } catch (e: HttpException) {
                MarsUiState.Error
            }
        }
    }

    /**
     * Factory for [MarsViewModel] that takes [MarsPhotosRepository] as a
dependency
     */
    companion object {
        val Factory: ViewModelProvider.Factory = viewModelFactory {
            initializer {
                val application = (this[APPLICATION_KEY] as
MarsPhotosApplication)
                val marsPhotosRepository =
application.container.marsPhotosRepository
                MarsViewModel(marsPhotosRepository = marsPhotosRepository)
            }
        }
    }
}
```

update viewmodel() at marsphotosapp.kt

```
val marsViewModel: MarsViewModel =
    viewModel(factory = MarsViewModel.Factory)
```

Get setup for local tests

Add the local test dependencies

```
testImplementation("junit:junit:4.13.2")
testImplementation("org.jetbrains.kotlinx:kotlinx-coroutines-test:1.7.1")
```

Create the local test directory

Create fake data and dependencies for tests

Create new object

```
package com.example.marsphotos.fake

import com.example.marsphotos.model.MarsPhoto

object FakeDataSource {

    const val idOne = "img1"
    const val idTwo = "img2"
    const val imgOne = "url.1"
    const val imgTwo = "url.2"
    val photosList = listOf(
        MarsPhoto(
            id = idOne,
            imgSrc = imgOne
        ),
        MarsPhoto(
            id = idTwo,
            imgSrc = imgTwo
        )
    )
}
```
And class
```
package com.example.marsphotos.fake

import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.network.MarsApiService

class FakeMarsApiService : MarsApiService {
    override suspend fun getPhotos(): List<MarsPhoto> {
        return FakeDataSource.photosList
    }
}
```

Write a repository test

```
package com.example.marsphotos.fake

import com.example.marsphotos.data.NetworkMarsPhotosRepository

class NetworkMarsRepositoryTest {
    @Test
    fun networkMarsPhotosRepository_getMarsPhotos_verifyPhotoList(){
        val repository = NetworkMarsPhotosRepository(
            marsApiService = FakeMarsApiService()
        )
        assertEquals(FakeDataSource.photosList, repository.getMarsPhotos())
    }

}
```

Test coroutines

```
package com.example.marsphotos

import com.example.marsphotos.data.NetworkMarsPhotosRepository
import com.example.marsphotos.fake.FakeDataSource
import com.example.marsphotos.fake.FakeMarsApiService
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Test
```

```
class NetworkMarsRepositoryTest {

    @Test
    fun networkMarsPhotosRepository_getMarsPhotos_verifyPhotoList() =
        runTest {
            val repository = NetworkMarsPhotosRepository(
                marsApiService = FakeMarsApiService()
            )
            assertEquals(FakeDataSource.photosList,
repository.getMarsPhotos())
        }
}
```

Write a ViewModel test

Create the fake repository

```
package com.example.marsphotos.fake

import com.example.marsphotos.data.MarsPhotosRepository
import com.example.marsphotos.model.MarsPhoto

class FakeNetworkMarsPhotosRepository : MarsPhotosRepository{
    override suspend fun getMarsPhotos(): List<MarsPhoto> {
        return FakeDataSource.photosList
    }
}
```

Create a test dispatcher

```
package com.example.marsphotos

import com.example.marsphotos.fake.FakeDataSource
import com.example.marsphotos.fake.FakeNetworkMarsPhotosRepository
import com.example.marsphotos.rules.TestDispatcherRule
import com.example.marsphotos.ui.screens.MarsUiState
import com.example.marsphotos.ui.screens.MarsViewModel
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Rule
import org.junit.Test

class MarsViewModelTest {

    @get:Rule
    val testDispatcher = TestDispatcherRule()

    @Test
    fun marsViewModel_getMarsPhotos_verifyMarsUiStateSuccess() =
        runTest {
            val marsViewModel = MarsViewModel(
                marsPhotosRepository = FakeNetworkMarsPhotosRepository()
            )
            assertEquals(
                MarsUiState.Success("Success:
${FakeDataSource.photosList.size} Mars " +
                        "photos retrieved"),
                marsViewModel.marsUiState
            )
        }
}
```

Load and display images from the internet

Display a downloaded image

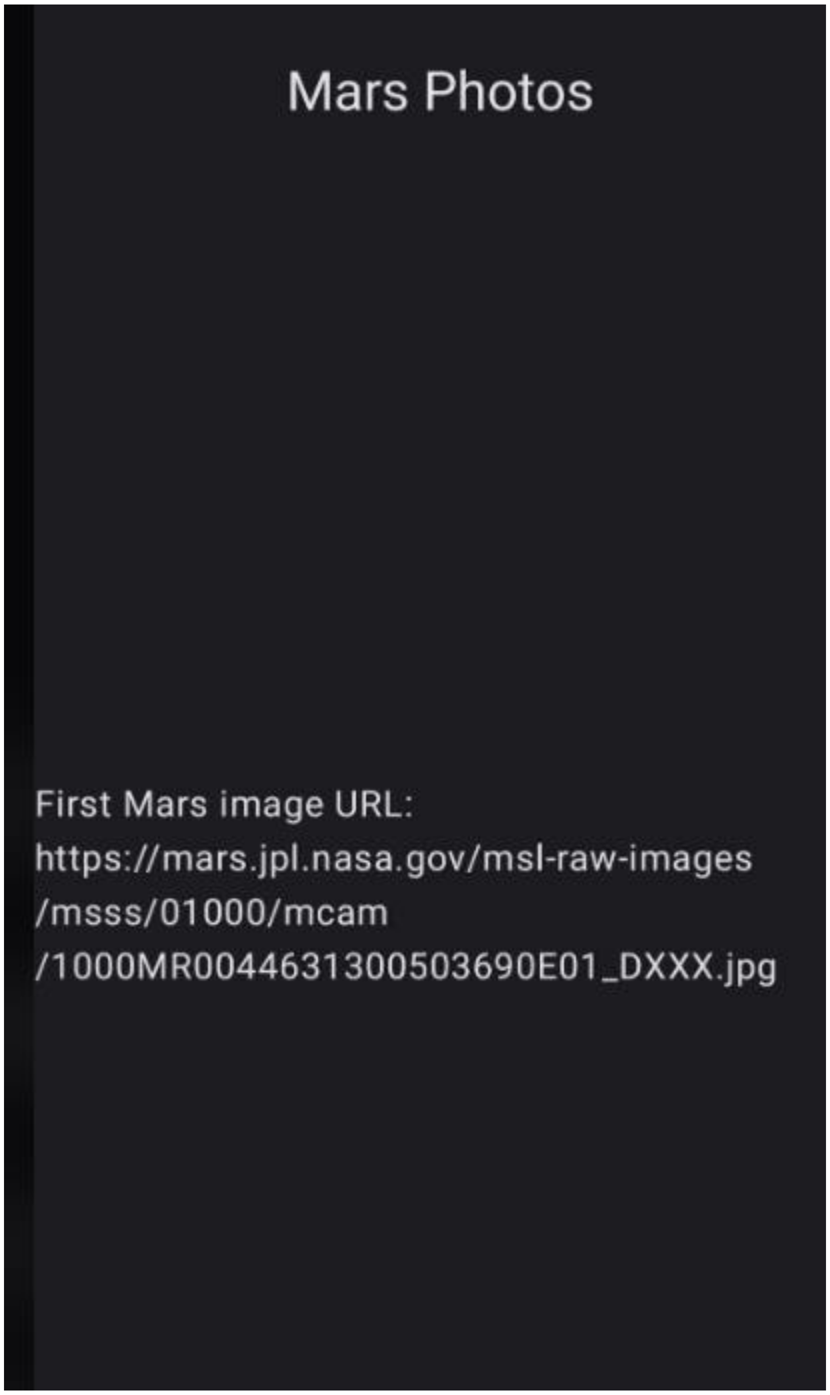
Add Coil dependency

Update at gradle

```
// Coil
implementation("io.coil-kt:coil-compose:2.4.0")
```

Update MarsViewModel.kt at getMarsPhotos()

```
fun getMarsPhotos() {
    viewModelScope.launch {
        marsUiState = MarsUiState.Loading
        marsUiState = try {
            val result = marsPhotosRepository.getMarsPhotos()[0]
            MarsUiState.Success("First Mars image URL: ${result.imgSrc}")
        } catch (e: IOException) {
            MarsUiState.Error
        } catch (e: HttpException) {
            MarsUiState.Error
        }
    }
}
```



First Mars image URL: https://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631300503690E01_DXXX.jpg

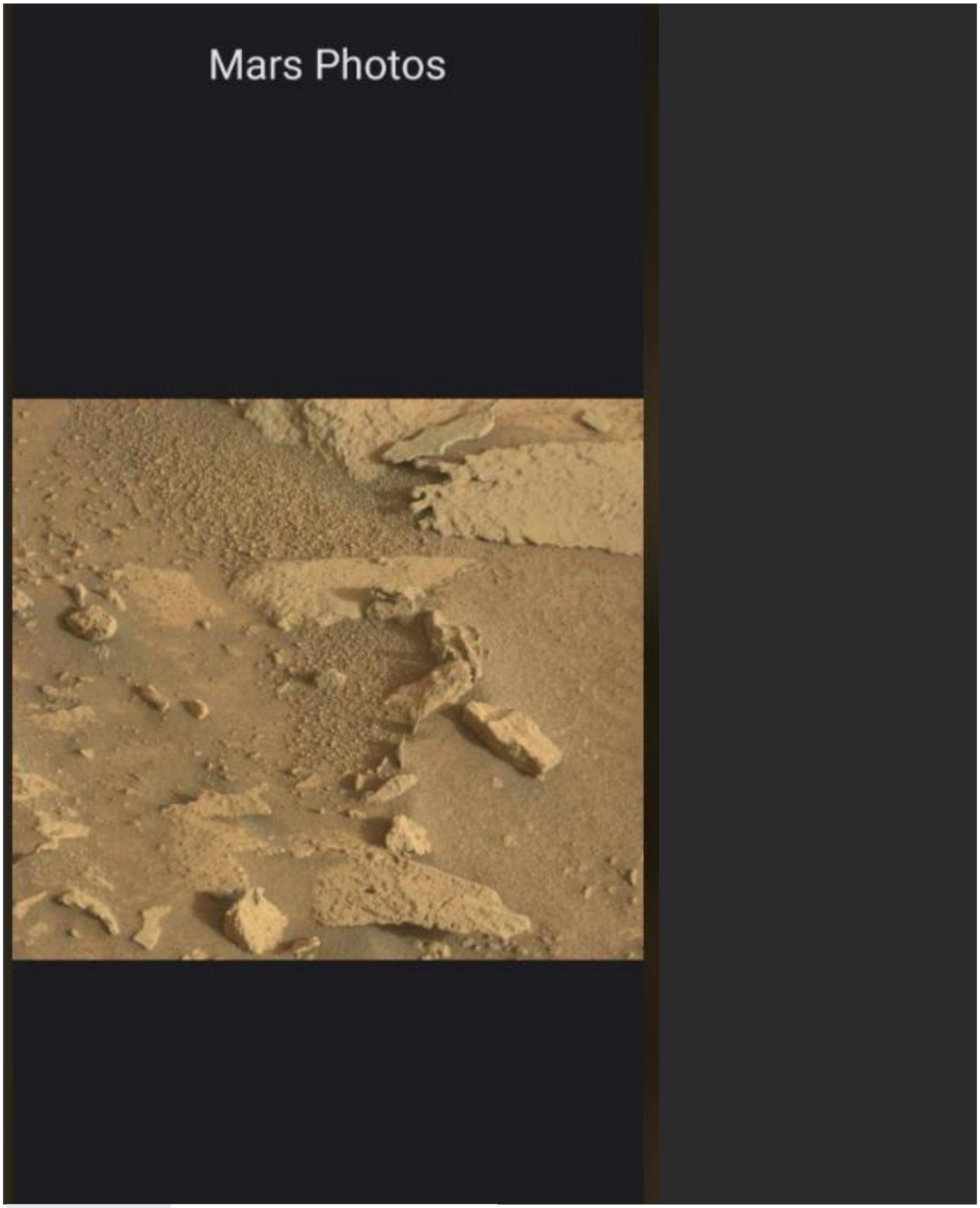
Add AsyncImage **composable**

Update HomeScreen.kt(it will have an error and we will fix later)

```
package com.example.marsphotos.ui.screens

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.marsphotos.R
import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.ui.theme.MarsPhotosTheme
import coil.compose.AsyncImage
import coil.request.ImageRequest
import androidx.compose.ui.platform.LocalContext

@Composable
fun HomeScreen(
    marsUiState: MarsUiState,
    modifier: Modifier = Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier = modifier)
        is MarsUiState.Success -> MarsPhotoCard(photo = marsUiState.photos,
modifier = modifier)
        else -> ErrorScreen(modifier = modifier)
    }
}

/**
 * The home screen displaying the loading message.
 */
@Composable
fun LoadingScreen(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier.size(200.dp),
        painter = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.loading)
    )
}

/**
 * The home screen displaying error message with re-attempt button.
 */
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error),
contentDescription = ""
        )
        Text(text = stringResource(R.string.loading_failed), modifier =
Modifier.padding(16.dp))
    }
}

/**
 * ResultScreen displaying number of photos retrieved.
 */
@Composable
fun ResultScreen(photos: String, modifier: Modifier = Modifier) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
    ) {
        Text(text = photos)
    }
}

@Preview(showBackground = true)
@Composable
fun LoadingScreenPreview() {
    MarsPhotosTheme {
        LoadingScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun ErrorScreenPreview() {
    MarsPhotosTheme {
        ErrorScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun PhotosGridScreenPreview() {
    MarsPhotosTheme {
        val mockData = List(10) { MarsPhoto("$it", "") }
        ResultScreen(stringResource(R.string.placeholder_success))
    }
}

@Composable
fun MarsPhotoCard(photo: MarsPhoto, modifier: Modifier = Modifier) {
    AsyncImage(
        model = ImageRequest.Builder(context = LocalContext.current)
            .data(photo.imgSrc)
            .crossfade(true)
            .build(),
        contentDescription = stringResource(R.string.mars_photo),
        modifier = Modifier.fillMaxWidth()
    )
}
```

Update at MarsViewModel.kt and result when run

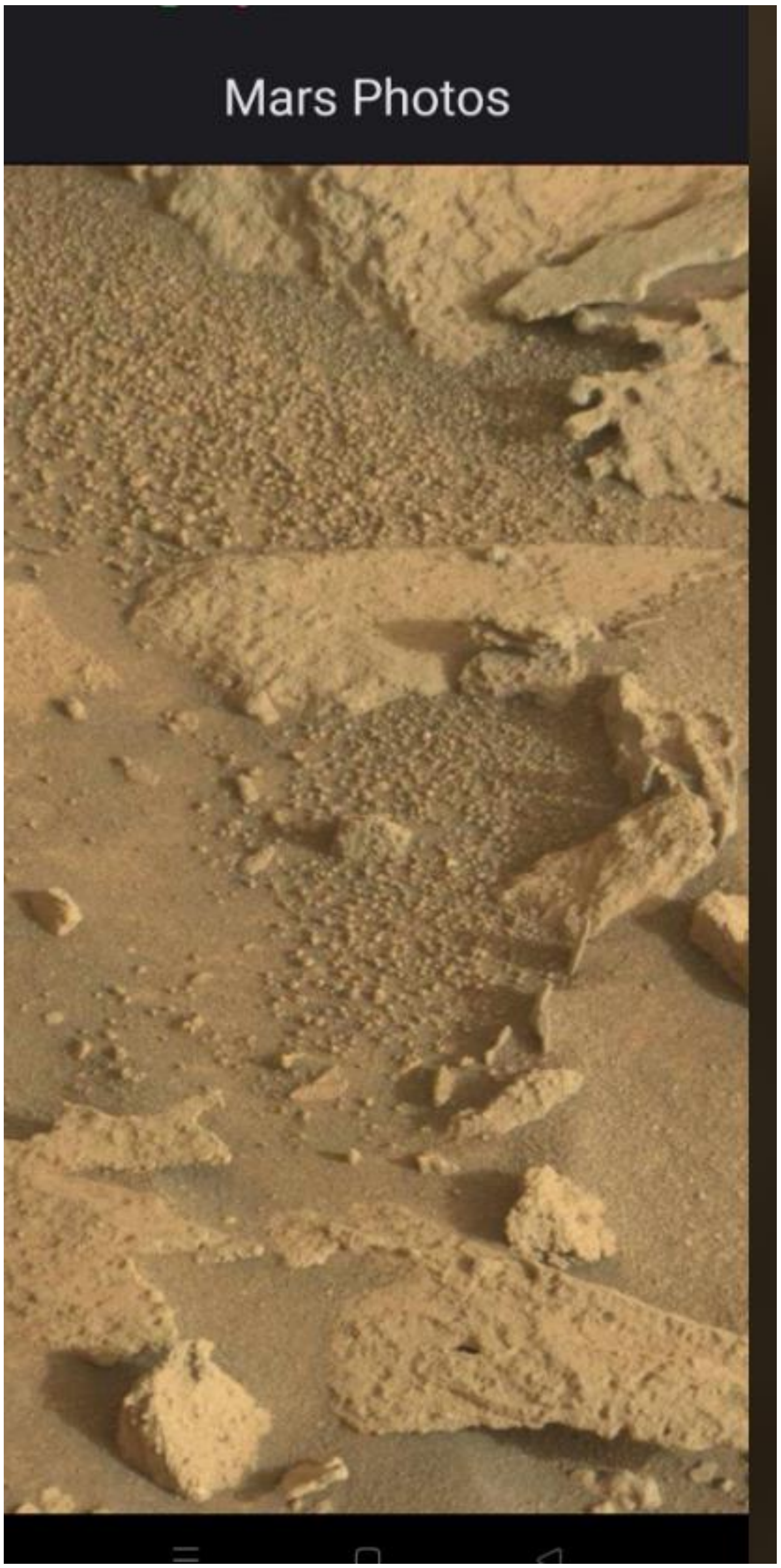
```
sealed interface MarsUiState {
    data class Success(val photos: MarsPhoto) : MarsUiState
fun getMarsPhotos() {
    viewModelScope.launch {
        marsUiState = MarsUiState.Loading
        marsUiState = try {
            MarsUiState.Success(marsPhotosRepository.getMarsPhotos()[0])
        } catch (e: IOException) {
            MarsUiState.Error
        } catch (e: HttpException) {
            MarsUiState.Error
        }
    }
}
```



HomeScreen.kt To fill available space on screen

```
@Composable
fun MarsPhotoCard(photo: MarsPhoto, modifier: Modifier = Modifier) {
    AsyncImage(
        model = ImageRequest.Builder(context = LocalContext.current)
```

```
            .data(photo.imgSrc)
            .crossfade(true)
            .build(),
        contentDescription = stringResource(R.string.mars_photo),
        contentScale = ContentScale.Crop,
        modifier = Modifier.fillMaxWidth()
    )
}
```



Now the space is fully covered

Add loading and error images

Update homescreen.kt to use loading and error images

```
@Composable
fun MarsPhotoCard(photo: MarsPhoto, modifier: Modifier = Modifier) {
    AsyncImage(
        model = ImageRequest.Builder(context = LocalContext.current)
            .data(photo.imgSrc)
            .crossfade(true)
            .build(),
        error = painterResource(R.drawable.ic_broken_image),
        placeholder = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.mars_photo),
        contentScale = ContentScale.Crop,
        modifier = Modifier.fillMaxWidth()
    )
}
```

Display a grid of images with a LazyVerticalGrid

Add LazyVerticalGrid

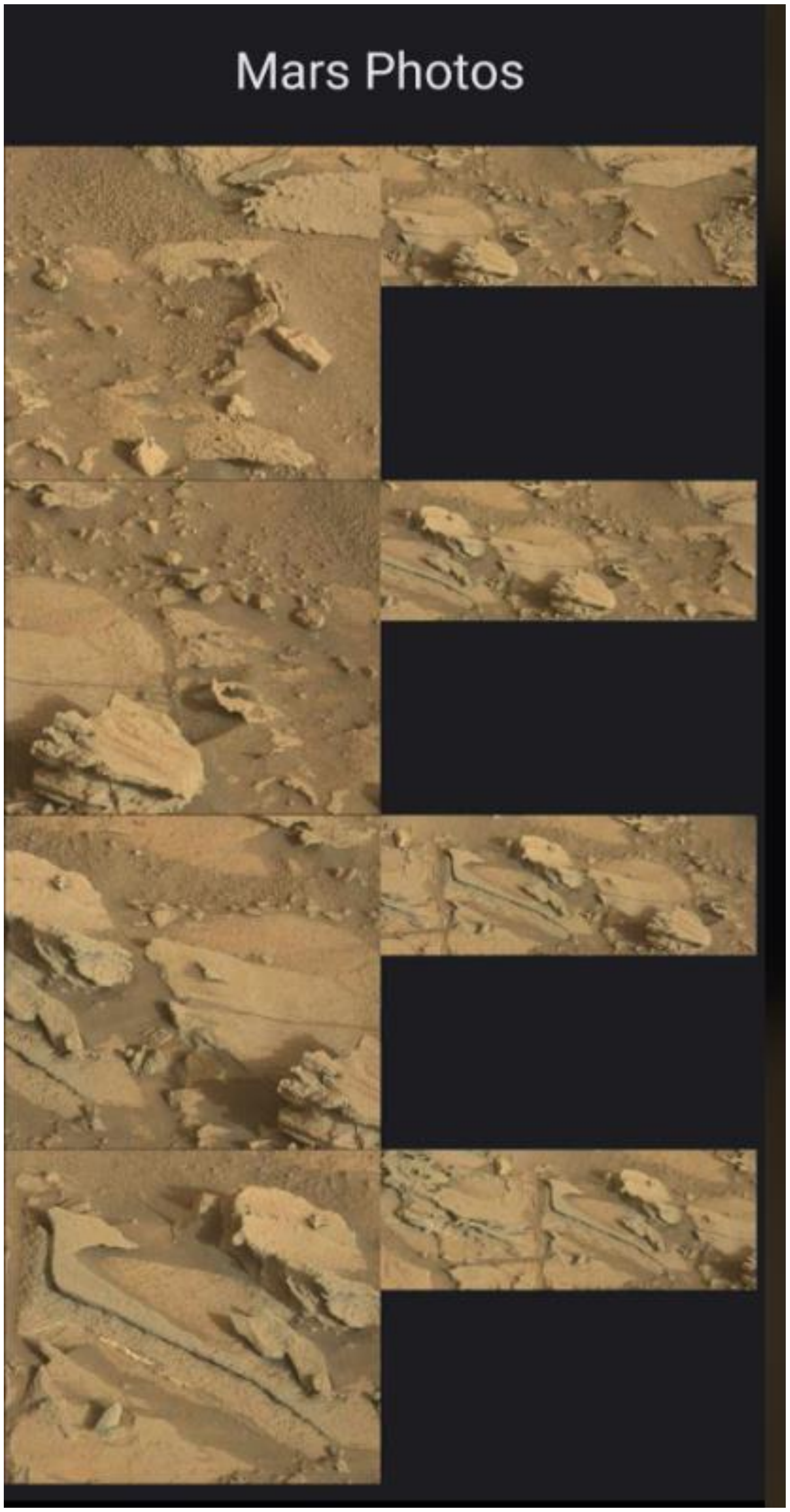
Reupdate Homescreen with method and fixed code

```

package com.example.marsphotos.ui.screens

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.marsphotos.R
import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.ui.theme.MarsPhotosTheme
import coil.compose.AsyncImage
import coil.request.ImageRequest
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.layout.ContentScale
import androidx.compose.foundation.lazy.grid.items

@Composable
fun HomeScreen(
    marsUiState: MarsUiState,
    modifier: Modifier = Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier = modifier)
//          is MarsUiState.Success -> MarsPhotoCard(photo = marsUiState.photos,
modifier = modifier)
        is MarsUiState.Success -> PhotosGridScreen(marsUiState.photos,
modifier)
        else -> ErrorScreen(modifier = modifier)
    }
}

/**
 * The home screen displaying the loading message.
 */
@Composable
fun LoadingScreen(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier.size(200.dp),
        painter = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.loading)
    )
}

/**
 * The home screen displaying error message with re-attempt button.
 */
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error),
contentDescription = ""
        )
        Text(text = stringResource(R.string.loading_failed), modifier =
Modifier.padding(16.dp))
    }
}

/**
 * ResultScreen displaying number of photos retrieved.
 */
@Composable
fun ResultScreen(photos: String, modifier: Modifier = Modifier) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
    ) {
        Text(text = photos)
    }
}

@Preview(showBackground = true)
```

```kotlin
@Composable
fun LoadingScreenPreview() {
    MarsPhotosTheme {
        LoadingScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun ErrorScreenPreview() {
    MarsPhotosTheme {
        ErrorScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun PhotosGridScreenPreview() {
    MarsPhotosTheme {
        val mockData = List(10) { MarsPhoto("$it", "") }
        ResultScreen(stringResource(R.string.placeholder_success))
    }
}

@Composable
fun MarsPhotoCard(photo: MarsPhoto, modifier: Modifier = Modifier) {
    AsyncImage(
        model = ImageRequest.Builder(context = LocalContext.current)
            .data(photo.imgSrc)
            .crossfade(true)
            .build(),
        error = painterResource(R.drawable.ic_broken_image),
        placeholder = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.mars_photo),
        contentScale = ContentScale.Crop,
        modifier = Modifier.fillMaxWidth()
    )
}
@Composable
fun PhotosGridScreen(photos: List<MarsPhoto>, modifier: Modifier = Modifier)
{
    LazyVerticalGrid(
        columns = GridCells.Adaptive(150.dp),
        modifier = modifier.fillMaxWidth(),
        contentPadding = PaddingValues(4.dp)
    ) {
        items(items = photos, key = { photo -> photo.id }) {
                photo -> MarsPhotoCard(photo)
        }
    }
}
```

fixed at marsViewmodel.kt

```kotlin
sealed interface MarsUiState {
    data class Success(val photos: List<MarsPhoto>) : MarsUiState
fun getMarsPhotos() {
    viewModelScope.launch {
        marsUiState = MarsUiState.Loading
        marsUiState = try {
            MarsUiState.Success(marsPhotosRepository.getMarsPhotos())
        }catch (e: IOException) {
            MarsUiState.Error
        } catch (e: HttpException) {
            MarsUiState.Error
        }
    }
}
```

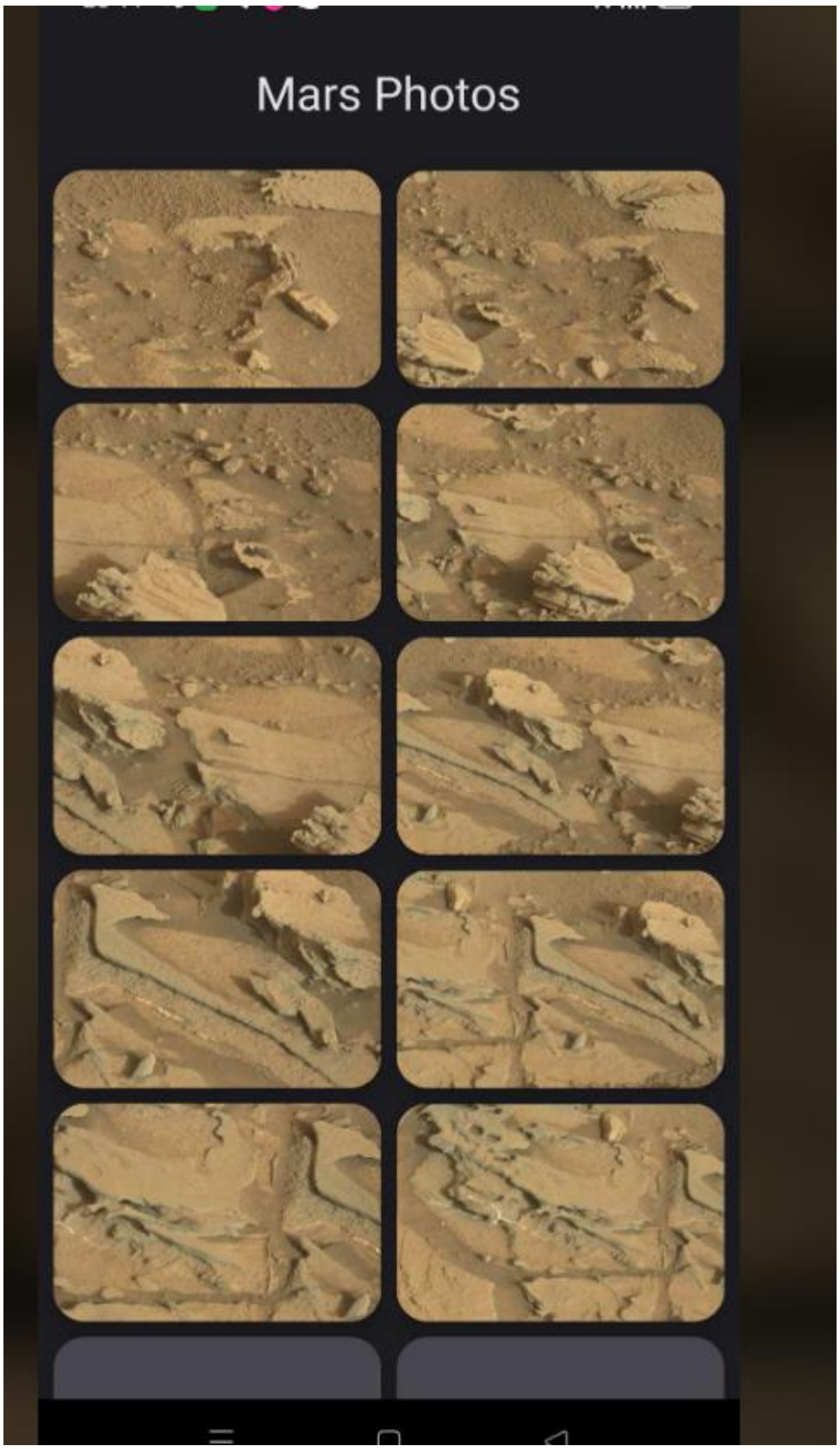


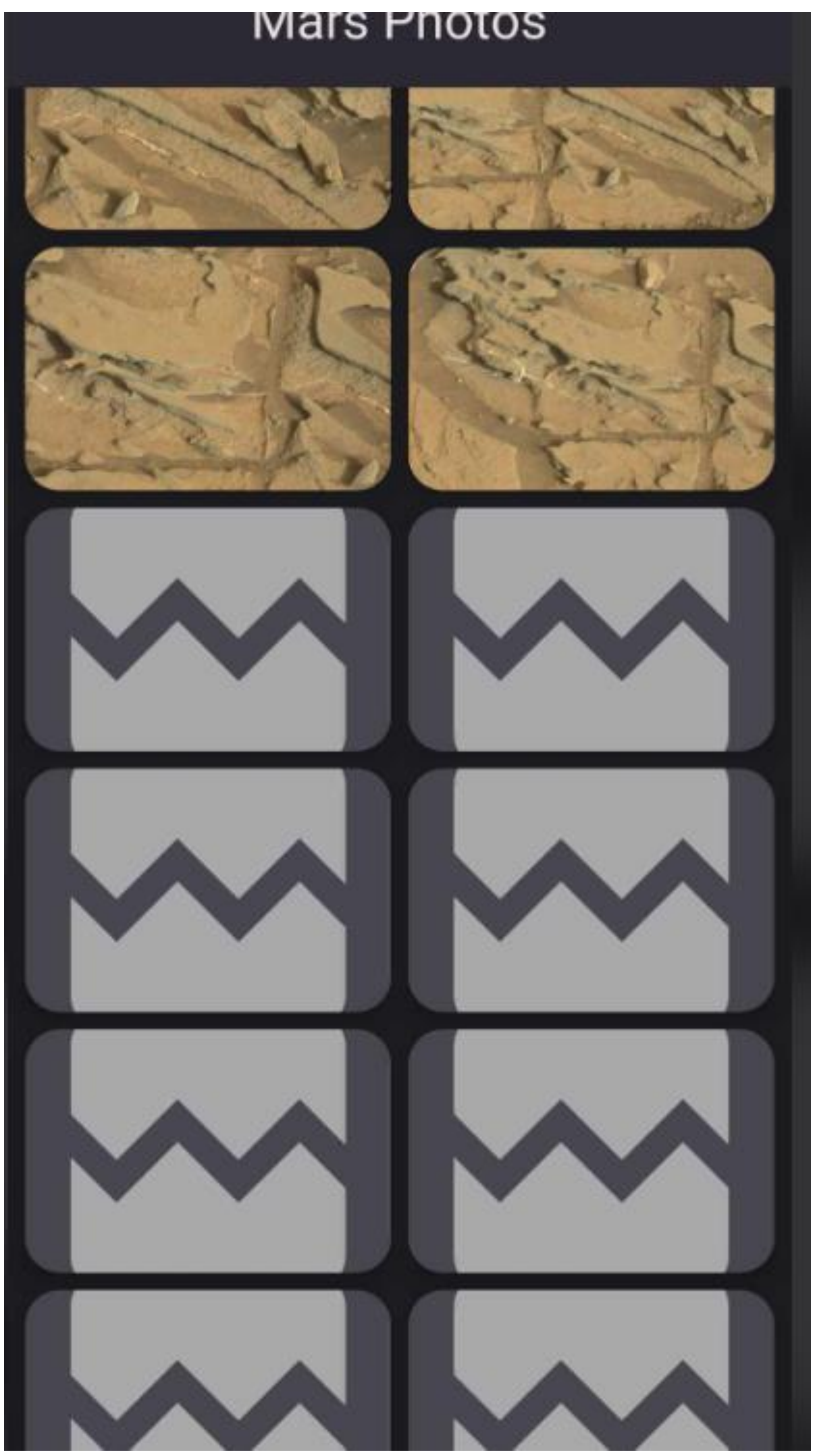
Add card composable

Update homescreen.kt

```
package com.example.marsphotos.ui.screens

import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.grid.GridCells
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.marsphotos.R
import com.example.marsphotos.model.MarsPhoto
import com.example.marsphotos.ui.theme.MarsPhotosTheme
import coil.compose.AsyncImage
import coil.request.ImageRequest
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.layout.ContentScale
import androidx.compose.foundation.lazy.grid.items
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.MaterialTheme


@Composable
fun HomeScreen(
    marsUiState: MarsUiState,
    modifier: Modifier = Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier = modifier)
//        is MarsUiState.Success -> MarsPhotoCard(photo = marsUiState.photos,
modifier = modifier)
        is MarsUiState.Success -> PhotosGridScreen(marsUiState.photos,
modifier)
        else -> ErrorScreen(modifier = modifier)
    }
}

/**
 * The home screen displaying the loading message.
 */
@Composable
fun LoadingScreen(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier.size(200.dp),
        painter = painterResource(R.drawable.loading_img),
        contentDescription = stringResource(R.string.loading)
    )
}

/**
 * The home screen displaying error message with re-attempt button.
 */
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error),
contentDescription = ""
        )
        Text(text = stringResource(R.string.loading_failed), modifier =
Modifier.padding(16.dp))
    }
}

/**
 * ResultScreen displaying number of photos retrieved.
 */
@Composable
fun ResultScreen(photos: String, modifier: Modifier = Modifier) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
    ) {
        Text(text = photos)
    }
}

@Preview(showBackground = true)
@Composable
fun LoadingScreenPreview() {
    MarsPhotosTheme {
        LoadingScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun ErrorScreenPreview() {
    MarsPhotosTheme {
        ErrorScreen()
    }
}

@Preview(showBackground = true)
@Composable
```

```
fun PhotosGridScreenPreview() {
    MarsPhotosTheme {
        val mockData = List(10) { MarsPhoto("$it", "") }
        PhotosGridScreen(mockData)
    }
}

@Composable
fun MarsPhotoCard(photo: MarsPhoto, modifier: Modifier = Modifier) {
    Card(
        modifier = modifier,
        shape = MaterialTheme.shapes.medium,
        elevation = CardDefaults.cardElevation(defaultElevation = 8.dp)
    ) {
        AsyncImage(
            model = ImageRequest.Builder(context =
LocalContext.current).data(photo.imgSrc)
                .crossfade(true).build(),
            error = painterResource(R.drawable.ic_broken_image),
            placeholder = painterResource(R.drawable.loading_img),
            contentDescription = stringResource(R.string.mars_photo),
            contentScale = ContentScale.Crop,
            modifier = Modifier.fillMaxWidth()
        )
    }
}
@Composable
fun PhotosGridScreen(photos: List<MarsPhoto>, modifier: Modifier = Modifier)
{
    LazyVerticalGrid(
        columns = GridCells.Adaptive(150.dp),
        modifier = modifier,
        contentPadding = PaddingValues(4.dp)
    ) {
        items(items = photos, key = { photo -> photo.id }) { photo ->
            MarsPhotoCard(
                photo,
                modifier = modifier
                    .padding(4.dp)
                    .fillMaxWidth()
                    .aspectRatio(1.5f)
            )
        }
    }
}
```


Mars Photos



Result and when turn on/off airplane mode

[Add retry action](#)

Update error screen and home screen

```
@Composable
fun HomeScreen(
    marsUiState: MarsUiState, retryAction: () -> Unit, modifier: Modifier =
Modifier
) {
    when (marsUiState) {
        is MarsUiState.Loading -> LoadingScreen(modifier =
modifier.fillMaxSize())
        is MarsUiState.Success -> PhotosGridScreen(
            marsUiState.photos, modifier = modifier.fillMaxWidth()
        )

        is MarsUiState.Error -> ErrorScreen(retryAction, modifier =
modifier.fillMaxSize())
    }
}
@Composable
fun ErrorScreen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
            painter = painterResource(id = R.drawable.ic_connection_error),
contentDescription = ""
        )
        Text(text = stringResource(R.string.loading_failed), modifier =
Modifier.padding(16.dp))
    }
}
```

Update homescreen at marsphotoapp.kt

```
HomeScreen(
    marsUiState = marsViewModel.marsUiState,
    retryAction = marsViewModel::getMarsPhotos
)
```

Update the ViewModel test

```
/*
 * Copyright (C) 2023 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

```kotlin
 */
package com.example.marsphotos

import com.example.marsphotos.fake.FakeDataSource
import com.example.marsphotos.fake.FakeNetworkMarsPhotosRepository
import com.example.marsphotos.rules.TestDispatcherRule
import com.example.marsphotos.ui.screens.MarsUiState
import com.example.marsphotos.ui.screens.MarsViewModel
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Rule
import org.junit.Test

class MarsViewModelTest {

    @get:Rule
    val testDispatcher = TestDispatcherRule()

    @Test
    fun marsViewModel_getMarsPhotos_verifyMarsUiStateSuccess() =
        runTest {
            val marsViewModel = MarsViewModel(
                marsPhotosRepository = FakeNetworkMarsPhotosRepository()
            )
            assertEquals(
                MarsUiState.Success(FakeDataSource.photosList),
                marsViewModel.marsUiState
            )
        }
}
```

1. Which of the following is not a common HTTP operation/method:

- ○ GET
- ○ POST
- ○ DELETE
- [✓ SET] ✓ Correct!

2. The response from a REST web service is commonly formatted in one of the common data transfer formats like XML or JSON.

- [✓ True] ✓ Correct!
- ○ False

3. Which of the following is not true for the Retrofit library:

- ○ It is a client library.
- ○ It enables your app to make requests to a REST web service.
- [✓ It converts Kotlin objects to JSON objects.] ✓ Correct!
- ○ It is a third-party library.

4. Which of the following applies to a Singleton pattern:

- ○ `object` declarations are used to declare singleton objects in Kotlin.
- ○ Ensures that one, and only one, instance of an object is created
- ○ Has one global point of access to that object.
- [✓ All of the above] ✓ Correct!

5. Each JSON object contains the following:

- ○ A set of key-value pairs separated by a colon.
- ● A set of key-value pairs separated by a comma. ✓ Correct!
- ○ A set of key-value pairs separated by a semi colon.
- ○ None of the above

6. Following Android's recommended app architecture guidelines, an app should have which of the following:

- ○ A UI Layer
- ○ A Domain Layer
- ● A Data Layer ✓ Correct!
- ○ A Business Layer

7. The advantages of using Dependency Injection (DI) in your app include which of the following:

Choose as many answers as you see fit.

- ☑ Helps with the reusability of code ✓ Correct!
- ☑ Makes refactoring easier ✓ Correct!
- ☑ Helps with testing ✓ Correct!
- ☐ Makes your app run faster

8. If your app has more than one type of data source, they should all be stored in the same repository for ease of use.

- ○ True
- [✓ False] ✓ Correct!

9. Which of the following is used to replace the `Main` dispatcher with a `TestDispatcher` in a local unit test:

- ○ `runTest`
- ○ `runBlocking`
- ○ `Distpatchers.resetMain()`
- [✓ `Dispatchers.setMain()`] ✓ Correct!

10. The `runTest()` function can be used to test `suspend` functions.

- [✓ True] ✓ Correct!
- ○ False

1. What are the two key things Retrofit needs to build a web services API?

- ○ The base URI for the web service, and a GET query.
- [✓ The base URI for the web service, and a converter factory.] ✓ Correct!
- ○ A network connection to the web service, and an authorization token.
- ○ A converter factory, and a parser for the response.

2. What is the purpose of the Moshi library?

- ○ To get data back from a web service.
- ○ To interact with Retrofit to make a web service request.
- [✓ To parse a JSON response from a web service into Kotlin data objects.] ✓ Correct!
- ○ To rename Kotlin objects to match the keys in the JSON response.

3. Which Glide method do you use to indicate the `ImageView` that will contain the loaded image?

- [✓ `into()`] ✓ Correct!
- ○ `with()`
- ○ `imageview()`
- ○ `apply()`

4. How do you specify a placeholder image to show when Glide is loading?

- ○ Use the `into()` method with a drawable.
- ● Use `RequestOptions()` and call the `placeholder()` method with a drawable. ✓ Correct!
- ○ Assign the `Glide.placeholder` property to a drawable.
- ○ Use `RequestOptions()` and call the `loadingImage()` method with a drawable.

5. How do you add a query option to a REST web service call in Retrofit?

- ○ Append the query to the end of the request URL.
- ● Add a parameter for the query to the function that makes the request, and annotate that parameter with `@Query`. ✓ Correct!
- ○ Use the Query class to build a request.
- ○ Use the `addQuery()` method in the Retrofit builder.