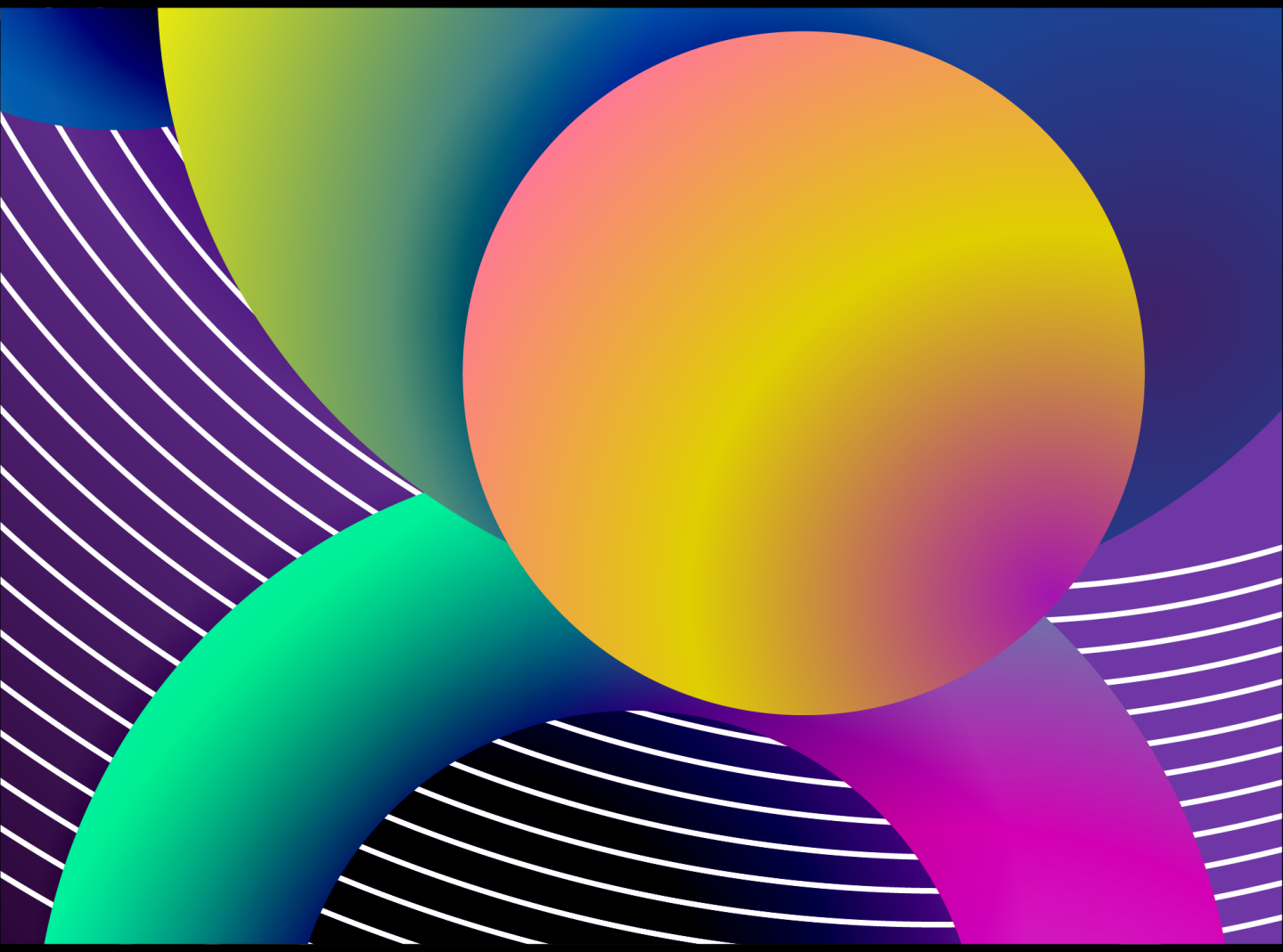
Jetpack Compose 1.8 Essentials



Payload publishing

Jetpack Compose 1.8 Essentials

Jetpack Compose 1.8 Essentials

ISBN-13: 978-1-965764-18-3

Rev: 1.0

https://www.payloadbooks.com

Chapter 63

63. Gemini AI Image Generation

This chapter will complete the GeminiDemo project by designing the final UI screen and extending the view model to add text-to-image generation support using the Gemini Imagen model. The steps in this chapter will demonstrate how to create and configure an Imagen model instance and use it to generate images based on text descriptions.

63.1 Opening the GeminiDemo project

If you are reading the AI chapters sequentially, launch Android Studio and open the GeminiDemo project from the previous chapter. Alternatively, open the *GeminiDemo_step3* project from the source code samples download. If you have not already done so, you can download the sample code using the following link:

https://www.payloadbooks.com/product/compose18/

Before proceeding, follow the steps in the *"Preparing the Gemini Firebase AI Logic Project"* chapter to add your *google-services.json* file to the project.

63.2 Initializing the Imagen model

Before generating images, we need to create an Imagen model instance by calling the *imagenModel()* method, passing the model name and configuration options, and defining the number of images required and the image's aspect ratio and format. Safety settings may also be included to control whether images containing people are allowed and how strictly Imagen should enforce objectionable content filters. Edit the *GeminiViewModel.kt* file and add the following code to create the Imagen model:

```
.
.
class GeminiViewModel: ViewModel() {

    private val _stateFlow = MutableStateFlow("")
    val stateFlow: StateFlow<String> = _stateFlow.asStateFlow()

    private val generativeModel =
        Firebase.ai(backend = GenerativeBackend.googleAI())
            .generativeModel(modelName = "gemini-2.5-flash")
.
.
    @OptIn(PublicPreviewAPI::class)
    val imagenModel = Firebase.ai(
        backend = GenerativeBackend.googleAI()
    ).imagenModel(
        modelName = "imagen-3.0-generate-002",
        generationConfig = ImagenGenerationConfig(
            numberOfImages = 1,
            aspectRatio = ImagenAspectRatio.LANDSCAPE_4x3,
```

```
            imageFormat = ImagenImageFormat.jpeg(),
        ),
        safetySettings = ImagenSafetySettings(
            safetyFilterLevel = ImagenSafetyFilterLevel.BLOCK_LOW_AND_ABOVE,
            personFilterLevel = ImagenPersonFilterLevel.ALLOW_ADULT
        )
    )
}
```

63.3 Sending the image generation request

With the Imagen model initialized, the next requirement is a method to send the image generation prompt to the model. This method will be called from the image generation screen (*ImageGenScreen.kt*) and passed the description of the required image. The generated image will be assigned to a StateFlow instance and collected for display on the image generation screen. With these requirements in mind, make the following additions to complete the GeminiViewModel class:

```
.
.
class GeminiViewModel : ViewModel() {

    private val _stateFlow = MutableStateFlow("")
    val stateFlow: StateFlow<String> = _stateFlow.asStateFlow()
    private val _generatedImage = MutableStateFlow<Bitmap?>(null)
    val generatedImage: StateFlow<Bitmap?> = _generatedImage.asStateFlow()
.
.
    @OptIn(PublicPreviewAPI::class)
    fun generateImage(prompt: String) {

        viewModelScope.launch(Dispatchers.IO) {

            try {
                _stateFlow.value = "Generating image..."
                val imageResponse = imagenModel.generateImages(
                    prompt = prompt
                )

                val image = imageResponse.images.first()
                _generatedImage.value = image.asBitmap()
                _stateFlow.value = "Image complete"
            } catch (e: Exception) {
                _stateFlow.value = e.localizedMessage ?: ""
            }
        }
    }
}
```

The above function calls the *generateImages()* Imagen model method from within a coroutine and passes it the image description text:

```
viewModelScope.launch(Dispatchers.IO) {

    try {
        _stateFlow.value = "Generating image..."
        val imageResponse = imagenModel.generateImages(
            prompt = prompt
        )
```

If the request succeeds, the generated image is extracted from the image response object, converted to a bitmap, and emitted via the *generatedImage* state flow:

```
val image = imageResponse.images.first()
_generatedImage.value = image.asBitmap()
_stateFlow.value = "Image complete"
```

Alternatively, if the generation request fails, the catch statement displays the error message via the stateFlow instance:

```
} catch (e: Exception) {
    _stateFlow.value = e.localizedMessage ?: ""
}
```

63.4 Designing the screen layout

The image generation screen will consist of Image, TextField, and Button composables. We will also include a Text component to display the generation status, including error messages. Open the *ImageGenScreen.kt* file located in the screens folder and edit it as follows:

```
.
.
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.graphics.asImageBitmap
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
```

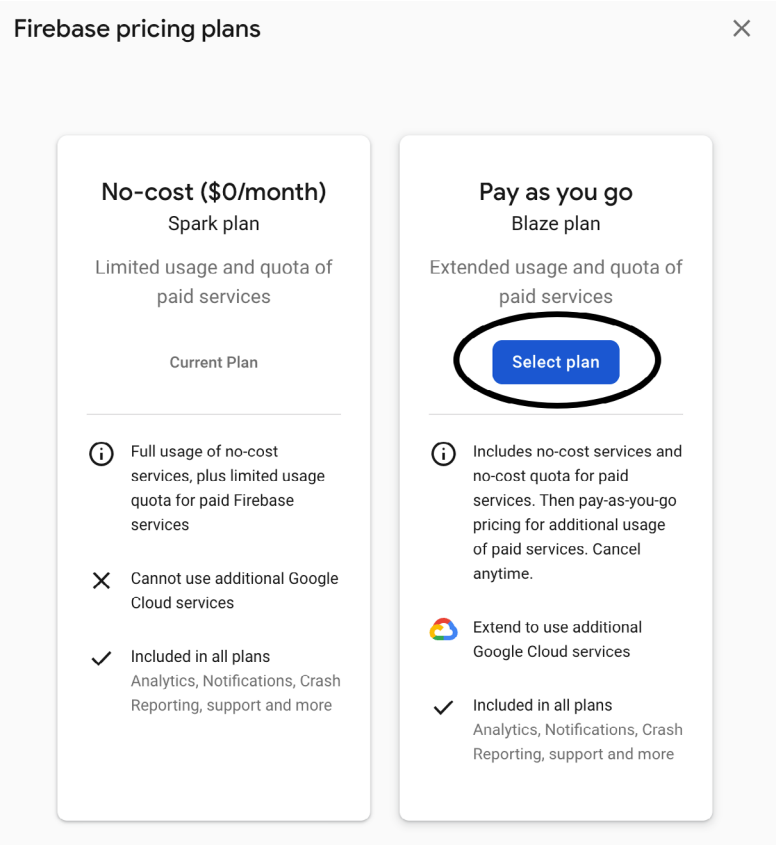


Figure 63-2

On the screen shown in Figure 63-3, follow the steps to set up a billing account and complete the upgrade:

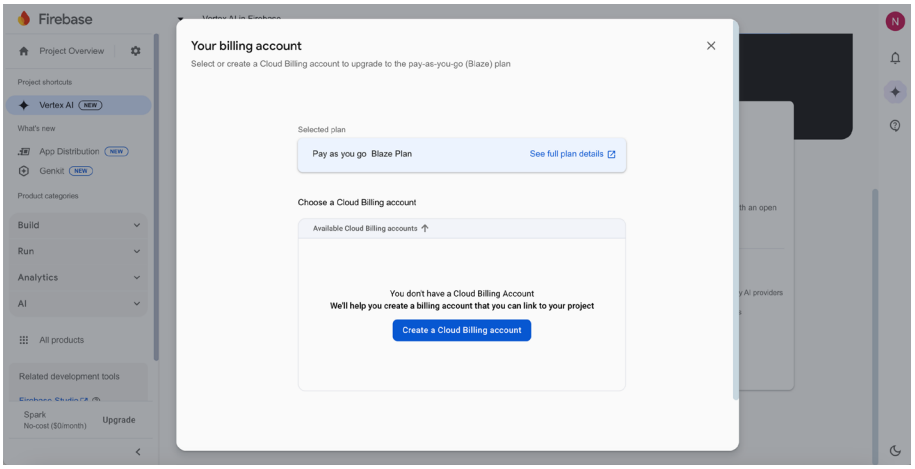


Figure 63-3

Firebase AI Logic prices are very low (cents rather than dollars or the equivalent in your preferred currency), and the costs incurred should be minimal.

63.6 Testing image generation

Run the app, select Image Gen in the bottom navigation bar, and enter a description before clicking the Generate Image button. If the request violates the model's safety settings or Google's mandatory content filters, the error will be reported on the Text component. If the image is generated successfully, it will appear on the screen as shown in Figure 63-4:

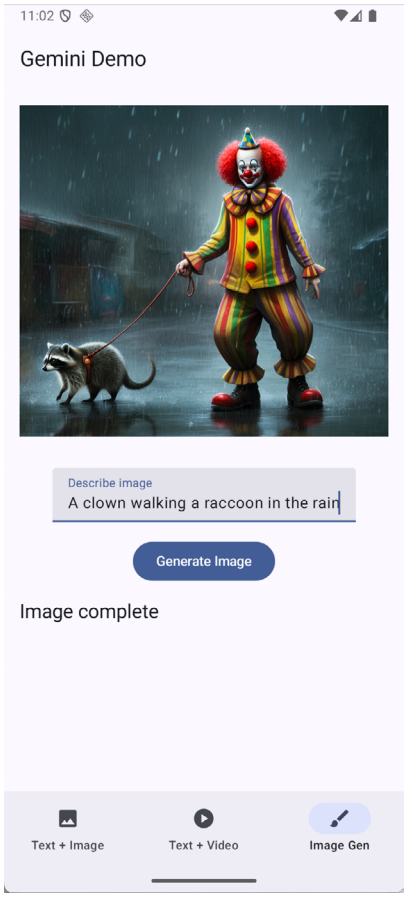


Figure 63-4

63.7 Take the knowledge test

Click the link below or scan the QR code to test your knowledge and understanding of the Gemini AI Imagen model:

https://www.answertopia.com/lhvy

63.8 Summary

The Imagen model generates images from text descriptions. Several options are available for configuring the Imagen model, including the aspect ratio, format, and the number of images to be generated. The Imagen model includes built-in rules that prevent the generation of inappropriate images. Additional optional filters may also be applied, including whether images can include adults or children. Once the model has been initialized and

configured, the *generateImage()* method is called and passed the image description. The generated images are returned as ImagenInlineImage instances packaged in an ImagenGenerationResponse object from which they can be accessed and converted to bitmap format.