

Create an Android Game From Scratch (or port your existing game)

Create An Android Game from Scratch (or Port your Existing Game) 5 Steps to Android Game Development - An Overview

INTRODUCTION



Author: James Cho

Almost six months ago, I began this

[Android game development tutorial](#)

series hoping to teach the absolute beginner how to create a fully functional Android game. As of the

[previous lesson](#)

(Unit 4, Day 6), we have studied Java, created our first 2D Java-based Platformer, discussed the power of Android, and laid the framework for a 2D Android game.

If you are new to this tutorial, feel free to look through the steps below. If you feel like you know enough about Java and Game Development to proceed, then by all means do so by clicking "Next" at the end of the page!

Who is this guide for?

1. People who have been

[previously following this tutorial](#)

and have gained basic knowledge in Java and Game development. This tutorial was written for you.

2. People who want to bring an existing Java game to Android. Following these steps will allow you to port most 2D Java games to Android.

3. Aspiring game developers testing the waters of Android Game Development. This guide will show you the necessary steps.

4. Anyone who is interested in Android or Games! If you are an absolute beginner, we recommend starting with [Unit 1](#).

The following is a detailed **overview** of Android Game Development. We elaborate on the game development framework developed in [Day 5 and Day 6](#).

5 Steps to Android Game Development

1. Download The Android Game Framework

SOURCE CODE



kiloboltandroidframework.zip

File Size: 411 kb

File Type: zip

[Download File](#)

2. CHANGE THE NAME OF THE PACKAGES

I. Right click on each of the two packages:

1. **com.kilobolt.framework**

2.

com.kilobolt.framework.implementation
Select Refactor >> Rename.

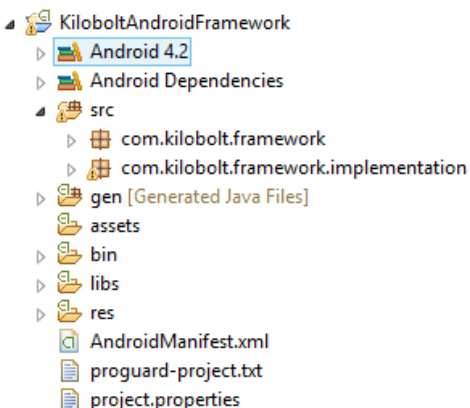
Change

kilobolt
to a name of your choice.

II. You will also want to create a new package in the **src** folder.

Call it:

com.yourcompany.yourgame.
You can also change the name of the Project for organization.



To recap what we have discussed in Day 5, our Android game's architecture will consist of:

1. The interface (com.kilobolt.framework)
2. The implementation of the interface (com.kilobolt.framework.implementation)
3. The game code (com.yourcompany.yourgame)

All of the classes that go into making your game will go into the 3rd package; however, **you cannot directly port a Java game to Android**. For example, Java's Swing class is not supported in Android, so you will not be able to use its methods (you would have to use methods that we implemented in package #2). This means you will be rewriting some of the code.

3. Develop the Game

In the newly created package, we can create our game. With a few changes, we can bring in most of the code from Units 2 and 3. If you are creating a game from scratch, you would just follow the following structure to create the fitting classes.

How will this game be structured?

Our game will have just one Activity (recall Activities are windows) that will display a SurfaceView (that we create using the **AndroidFastRenderView** class). This SurfaceView will paint objects (created using classes like *Robot* or *Platform*) as they update.

How do we go about doing this? Assuming that we are creating a game called *Sample*:

1. Create a *SampleGame* class in the 3rd package. Extend *AndroidGame*.

Since *AndroidGame* implements the Game class from the **framework** you typically would have to implement all the methods from the Game class inside *AndroidGame*. However, since *AndroidGame* is an abstract class, you can choose not to implement certain methods. However, you would be required to implement these methods in any subclass that extends *AndroidGame* (in this case *SampleGame*).

So our SampleGame might look something like this:

1. SampleGame class (sample code)

```
package com.kilobolt.samplegame;
import com.kilobolt.framework.Screen;

import com.kilobolt.framework.implementation.AndroidGame;
public class SampleGame extends AndroidGame {

    @Override

    public
    Screen getInitScreen

    () {
        return new
```

LoadingScreen

```
(this);  
}  
}
```

We would make SampleGame the main Activity of our game (we will do so in the AndroidManifest in step 4). That way, when we start the game, the SampleGame class will be instantiated, and the methods from the Activity Lifecycle will be called (starting with the

onCreate

). These methods are all implemented in the AndroidGame superclass that SampleGame extends.

You will find that we set most of the screen layouts in the **onCreate** method of the AndroidGame class. You can change the screen resolution of the game in there by editing these four numbers:

```
int framebufferWidth = isPortrait ? 800: 1280;
```

```
int framebufferHeight = isPortrait ?
```

```
1280
```

```
:
```

```
800
```

```
;
```

As it is, this would create a 1280x800 canvas for our game. This canvas will shrink or stretch to fit any device.

To add the back button functionality, add the following code:

```
@Override  
public void onBackPressed() {  
getCurrentScreen().backButton();  
}
```

Notice that the getInitScreen() has the following statement:

```
return new LoadingScreen(this);
```

This would transition into the **LoadingScreen** class, which we will define soon:

2. Assets class (sample code)

```
packagecom.kilobolt.samplegame;
import com.kilobolt.framework.Image;
```

```
importcom.kilobolt.framework.Sound;
public class Assets {
```

```
    public static Image menu;
```

```
    publicstatic
    Sound click
```

```
    ;
}
This
```

Assets

class is used to create a variable for each resource that we will use in the game. Notice that these variables

menu
and

click
are not initialized. We will initialize them in the LoadingScreen below:

3. LoadingScreen class (sample code)

```
packagecom.kilobolt.samplegame;
import com.kilobolt.framework.Game;
```

```
importcom.kilobolt.framework.Graphics;
importcom.kilobolt.framework.Screen;
importcom.kilobolt.framework.Graphics.ImageFormat;
public class LoadingScreen extends Screen {
```

```
    public
    LoadingScreen
```

```
    (
    Game game
```

```
    ){
    super(
    game
```

```
    );
}
    @Override
```

```
    publicvoid
```

update

(float
deltaTime

){

[Graphics](#)

g

=

game.

getGraphics();
Assets.

menu=

g.

newImage("menu.jpg"
, ImageFormat.

RGB565);
Assets.

click=
game.

getAudio()
.

createSound("explode.ogg");
game.setScreen(new MainMenuScreen(game));

}

@Override

publicvoid
paint

(float
deltaTime

){

}

@Override

publicvoid
pause

()

}

```
@Override
```

```
public void  
resume
```

```
() {  
}
```

```
@Override
```

```
public void  
dispose
```

```
() {  
}
```

```
@Override
```

```
public void  
backButton
```

```
() {  
}
```

```
}  
All
```

screen

classes have three important classes. The `update()` method, and the `paint()` method, and the `backButton()` method (which is called when the user presses the back button in the game).

In the `update()` method, you load all the resources that you will use in the game (i.e. all the resources that we have created in the **Assets** class). We would not need anything in our **paint()** method, unless you would like to have an image while the game loads these resources (make sure you load this in another class).

The files used in this example: `menu.jpg`, `explode.ogg` must be placed in the **assets** folder of our project.

When all the Assets are loaded, we call the statement: **`game.setScreen(new MainMenuScreen(game));`**

This opens the *MainMenuScreen*, which we define below:

4. MainMenuScreen class (sample code)

```
package com.kilobolt.samplegame;  
import java.util.List;
```

```
import com.kilobolt.framework.Game;
```

```
import com.kilobolt.framework.Graphics;  
import com.kilobolt.framework.Screen;  
import com.kilobolt.framework.Input.TouchEvent;  
public class MainMenuScreen extends Screen {
```

```
    public  
    MainMenuScreen
```

```
    (  
    Game game
```

```
    ){  
    super(  
    game
```

```
    );  
    }  
    @Override
```

```
    public void  
    update
```

```
    (float  
    deltaTime
```

```
    ){  
    Graphics  
    g
```

```
    =  
    game.
```

```
    getGraphics();  
    List
```

```
    <  
    TouchEvent
```

```
    >  
    touchEvents
```

```
    =  
    game.
```

```
    getInput()  
    .
```

```
    getTouchEvents();  
    int len = touchEvents.size();
```



```

for(int
i

=0;
i

<
len

;
i

++){
    TouchEvent event

=
touchEvents.

get(
i

);
if(
event.

type==
TouchEvent.

TOUCH_UP){
    if (inBounds(event, 0, 0, 250, 250)) {

//START GAME

        game.

setScreen(new
GameScreen

(
game

));
    }

    }

}

private boolean inBounds(TouchEvent event, int x, int y, int width,

int
height

```

```
) {  
    if(  
        event.
```

```
    x >  
    x
```

```
    &&  
    event.
```

```
    x <  
    x
```

```
    +  
    width
```

```
    -1 &&  
    event.
```

```
    y >  
    y
```

```
    &&  
    event.
```

```
    y <  
    y
```

```
    +  
    height
```

```
    -1)  
    return true;  
    else  
    return false;  
    }  
    @Override
```

```
public void  
    paint
```

```
(float  
    deltaTime
```

```
) {  
    Graphics  
    g
```

```
    =  
    game.
```

```
    getGraphics();  
    g.
```

```
drawImage(  
Assets.
```

```
menu
```

```
,
```

```
0
```

```
,
```

```
0);
```

```
}
```

```
@Override
```

```
publicvoid
```

```
pause
```

```
() {
```

```
}
```

```
@Override
```

```
publicvoid
```

```
resume
```

```
() {
```

```
}
```

```
@Override
```

```
publicvoid
```

```
dispose
```

```
() {
```

```
}
```

```
@Override
```

```
publicvoid
```

```
backButton
```

```
() {
```

```
//Display "Exit Game?" Box
```

```
}
```

```
}
```

Notice that here we also have the three methods: update, paint, and backButton.

In addition, we have added an

inBounds

method that is used to create rectangles with coordinates (x, y, x2, y2).

We use this to create regions in the screen that we can touch to interact with the game (as we do here in the update method). In our case, when the user touches and releases inside the square with side length 250 with a corner at (0, 0), we would call the: `game.setScreen(new GameScreen(game));`

This is the screen on which we will run our game. Think of this as the **StartingClass** from Units 2 and 3.

Using the same techniques that we have used in the MenuScreen above, along with the game development techniques from the previous lessons, you should now be able to create your game. Begin experimenting using the sample below. If you ever get stuck, return to **Kilobolt.com** as I will be continuing Unit 4 by porting our game from Units 2 and 3 to Android. You can see a fully working example [here](#).

In addition, I will be demonstrating how to create a High Scores screen, Splash screen, and demonstrating how to Restart the game once the player dies.

Here's a sample GameScreen.

4. GameScreen class (sample code)

Use this as a starting point for your game!

```
package com.kilobolt.samplegame;
import java.util.List;

import android.graphics.Color;

import android.graphics.Paint;
import com.kilobolt.framework.Game;

import com.kilobolt.framework.Graphics;
import com.kilobolt.framework.Image;
import com.kilobolt.framework.Screen;
import com.kilobolt.framework.Input.TouchEvent;
public class GameScreen extends Screen {

    enum
    GameState

    {
        Ready, Running, Paused, GameOver
    }

    GameState state = GameState.Ready;

    // Variable Setup

    // You would create game objects here.
```

```
int livesLeft = 1;
```

Paint

```
paint
```

```
;
```

```
public GameScreen(Game game) {
```

```
super(
```

```
game
```

```
);
```

```
// Initialize game objects here
```

```
// Defining a paint object
```

```
paint
```

```
=new Paint();
```

```
paint.
```

```
setTextSize(30);
```

```
paint.
```

```
setTextAlign(Paint
```

```
.
```

```
Align
```

```
.
```

```
CENTER);
```

```
paint.
```

```
setAntiAlias(true);
```

```
paint.
```

```
setColor(Color
```

```
.
```

```
WHITE);
```

```
}
```

```
@Override
```

```
public void
```

```
update
```

```
(float
```

```
deltaTime
```

```
){
```

```
List
```

```
<
TouchEvent
```

```
>
touchEvents
```

```
=
game.
```

```
getInput()
.
```

```
getTouchEvents();
    // We have four separate update methods in this example.
```

```
// Depending on the state of the game, we call different update methods.
// Refer to Unit 3's code. We did a similar thing without separating the
// update methods.
```

```
    if (state == GameState.Ready)
```

```
        updateReady
```

```
(
touchEvents
```

```
);
if(
state
```

```
==
GameState.
```

```
Running)
    updateRunning
```

```
(
touchEvents, deltaTime
```

```
);
if(
state
```

```
==
GameState.
```

```
Paused)
    updatePaused
```

```
(
touchEvents
```

```
);
if(
state
```

```

==
GameState.

GameOver)
    updateGameOver

(
touchEvents

);
}
    private void updateReady(List<TouchEvent> touchEvents) {

        // This example starts with a "Ready" screen.

// When the user touches the screen, the game begins.
// state now becomes GameState.Running.
// Now the updateRunning() method will be called!
        if (touchEvents.size() > 0)

            state

=
GameState.

Running;
}
    private void updateRunning(List<TouchEvent> touchEvents, float deltaTime) {

        //This is identical to the update() method from our Unit 2/3 game.

        // 1. All touch input is handled here:

int
len

=
touchEvents.

size();
for(int
i

=0;
i

<
len

;
i

++){

```

```

        TouchEvent event

    =
    touchEvents.

    get(
    i

    );
        if (event.type == TouchEvent.TOUCH_DOWN) {

            if (event.x < 640) {

// Move left.
            }

            else if (event.x > 640) {

// Move right.
            }

            }

            if (event.type == TouchEvent.TOUCH_UP) {

                if (event.x < 640) {

// Stop moving left.
                }

                else if (event.x > 640) {

// Stop moving right. }
                }
            }

            }

            // 2. Check miscellaneous events like death:

            if (livesLeft == 0) {

                state

            =

            GameState.

            GameOver;
            }

            // 3. Call individual update() methods here.

// This is where all the game updates happen.
// For example, robot.update();
    }

```



```
private void updatePaused(List<TouchEvent> touchEvents) {
```

```
int  
len
```

```
=  
touchEvents.
```

```
size();  
for(int  
i
```

```
=0;  
i
```

```
<  
len
```

```
;  
i
```

```
++){  
    TouchEvent event
```

```
=  
touchEvents.
```

```
get(  
i
```

```
);  
if(  
event.
```

```
type==  
TouchEvent.
```

```
TOUCH_UP){  
    }
```

```
}  
}
```

```
private void updateGameOver(List<TouchEvent> touchEvents) {
```

```
int  
len
```

```
=  
touchEvents.
```

```
size();  
for(int
```

```
i

=0;
i

<
len

;
i

++){
    TouchEvent event

=
touchEvents.

get(
i

);
if(
event.

type==
TouchEvent.

TOUCH_UP){
    if(
event.

x>300&&
event.

x<980&&
event.

y>100
&&
event.

y<500){
        nullify

    };
        game.

setScreen(new
MainMenuScreen

(
game

));
return;
```

```
}  
}  
}  
}
```

```
@Override
```

```
public void  
paint
```

```
(float  
deltaTime
```

```
){  
Graphics  
g
```

```
=  
game.
```

```
getGraphics();  
    // First draw the game elements.
```

```
    // Example:
```

```
// g.drawImage(Assets.background, 0, 0);  
// g.drawImage(Assets.character, characterX, characterY);  
    // Secondly, draw the UI above the game elements.
```

```
if(  
state
```

```
==  
GameState.
```

```
Ready)  
    drawReadyUI
```

```
();  
if(  
state
```

```
==  
GameState.
```

```
Running)  
    drawRunningUI
```

```
();  
if(  
state
```

```

==
GameState.

Paused)
    drawPausedUI

());
if(
state

==
GameState.

GameOver)
    drawGameOverUI

());
}

private void nullify() {

    // Set all variables to null. You will be recreating them in the

// constructor.
    paint

=null;
    // Call garbage collector to clean up memory.

```

System

```

.

gc();
}
private void drawReadyUI() {

```

Graphics

```

g

=
game.

getGraphics();
    g.drawARGB(155, 0, 0, 0);

    g.

drawString("Tap each side of the screen to move in that direction."
,

```

```
,  
  
300  
, paint  
  
);  
}  
  
private void drawRunningUI() {
```

Graphics

```
g  
  
=  
game.  
  
getGraphics();  
}  
  
private void drawPausedUI() {
```

Graphics

```
g  
  
=  
game.  
  
getGraphics();  
// Darken the entire screen so you can display the Paused screen.  
g.  
  
drawARGB(155  
,  
  
0  
,  
  
0  
,  
  
0);  
}  
  
private void drawGameOverUI() {
```

Graphics

```
g  
  
=  
game.  
  
getGraphics();
```

```
        g.  
drawRect(0  
,  
0  
,  
1281  
,  
801  
,  
Color  
.  
BLACK);  
        g.  
drawString("GAME OVER."  
,  
640  
,  
300  
, paint  
);  
    }  
  
    @Override  
  
    public void  
    pause  
  
    () {  
        if (  
            state  
  
            ==  
            GameState.  
            Running)  
                state  
  
            =  
            GameState.  
            Paused;  
        }  
  
        @Override
```

```
public void  
resume  
  
{  
}  
  
@Override
```

```
public void  
dispose  
  
{  
}  
  
@Override
```

```
public void  
backButton  
  
{  
    pause  
  
};  
}  
}
```

4. Edit the AndroidManifest

With the game finished, we must edit our

AndroidManifest
file.

Open up **AndroidManifest.xml**.

- We set our Main Activity (the class that will open when the application starts).
- Set the version of the game.

The **versionCode** just represents the release number (1st release, 2nd release). So each time that you patch the game and upload it to the Play Store, you would increase this by 1.

The **versionName** is the game version that is visible to users on the Play Store. You can make this whatever you'd like.

- You can change the icon here by placing an image into the **drawable** folder. Right now, it is the ic_launcher.png.
- Change the label to change the name of the Game.
- Set the Main Activity by changing android:name as below. You can have a

separate "label" for each activity in your app. We only have one activity, so I make it the same as that of our application.

We must also set **permissions** that our game might require, as Android prevents applications from doing things that might harm the user's experience unless it has been granted permission.

As we need Wake lock (preventing our game from going to sleep), storage permission (for saving data), and vibrations (for a better experience), we must add the following below the "manifest" element:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-
permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE" />
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.kilobolt.androidgame"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-permission android:name="android.permission.WAKE_LOCK" />
```

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE" />
  <uses-sdk
```

```
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
</application>
```

```
    android:icon="@drawable/ic_launcher"
    android:label="SampleGame" >
<activity
    android:name=".SampleGame"
    android:configChanges="keyboard|keyboardHidden|orientation"
    android:label="SampleGame"
    android:screenOrientation="landscape" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />

</intent-filter>
</activity>
</application>
</manifest>
```

Those are all the changes you need!

5. Export/Publish the Game

Export refers to packaging your game into an APK file (an Android application package file). You can do this by following the steps below:

1. Right click on your Android project.
2. Select Export >> Android >> Export Android Application
3. Each time that you export the game, you must

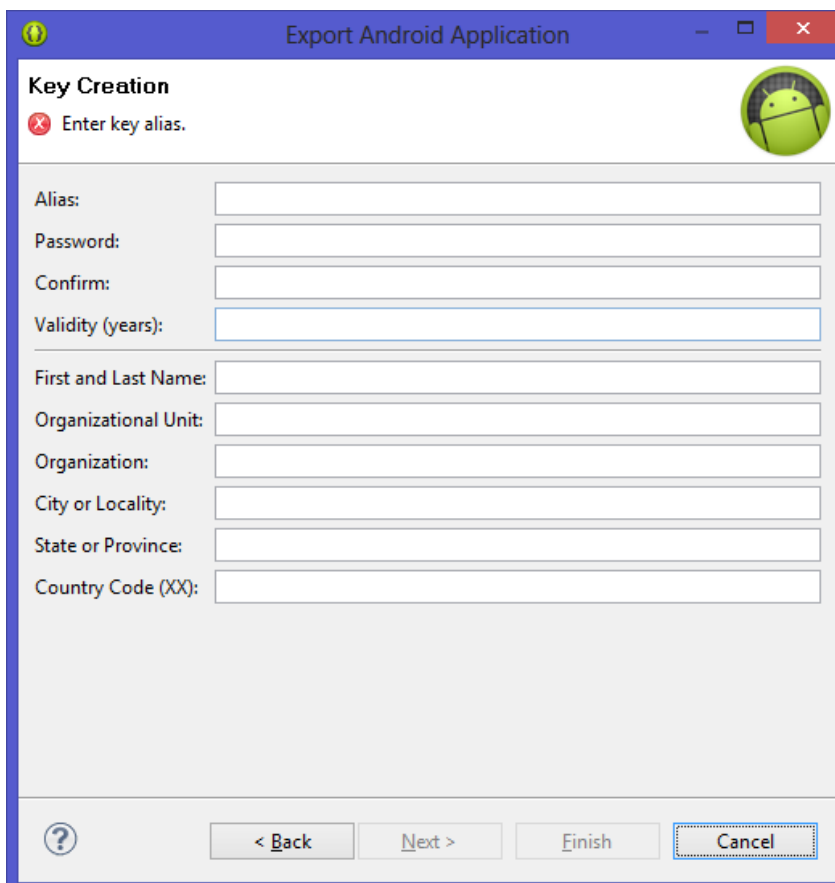
sign it

(literally put your information on it).

- If you have never published the current game before, you must create a new keystore file. Make sure you store this somewhere safe and make copies.

You must use the same keystore each time that you publish the same application (so if you want to update your application, keep your keystore safe!

Creating a Keystore



Alias

is the name of the keystore file. You can call this keystore1, myKeystore, etc.

Password: can be anything you choose. At least 6 characters. You must remember this password!

Validity: This is how long your signature will be valid. A good number is 30 years.

You can choose to fill out the remaining information as required.

Loading a keystore

- You can also load a keystore by browsing to the keystore file and entering the password you set.

After you sign your application, export your APK to a location of your choice. You will now be uploading this to your Developer account.

Publishing a Game

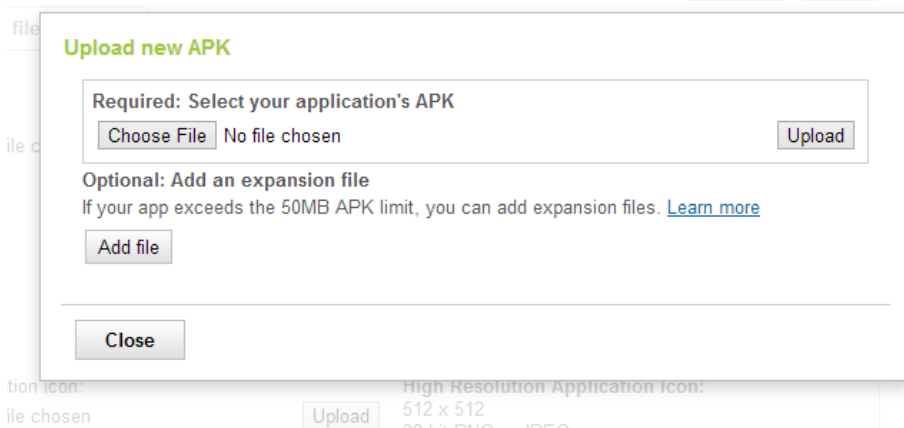
To publish the game, you must first register for an account. There's a one time registration fee of \$25. After that, you can publish as many apps as you would like.

Registration:

<https://play.google.com/apps/publish/signup>

After you have registered, the interface makes it extremely easy to upload your app.

1. Press **Upload Application**.



Locate your new application. It will begin uploading.

2. **Upload assets**, such as promotional graphics, icon graphics, etc. These images will be used throughout the Play Store.

3. **Fill out the listing details**, including the name of the application and the description.

3. **Select Publishing options**. Decide which countries you want to target, how much you want to charge, etc.

4. Finally, **Consent** by acknowledging Android Guidelines and legal compliance, and scroll up.

5. Hit the **Publish** button.

Congratulations! You have published your first game.

Final Words

I have given you a quick overview of how to create an Android game from beginning to the end. Now the choice is yours.

If you would like to follow along as I create a fully working example, move on to the next lesson. If you feel like you can handle it on your own, I wish you much success!

Thank you so much for following this tutorial thus far. If you want to say "thank you," you can support us by:

[1. Sending us a Donation!](#)

[2. Liking us on Facebook!](#)

3. Linking to us on various forums and websites!

These things will help us bring you (and everyone else) more high quality content!

As always, thank you for reading.

Feel free to email me questions at

`jamescho7@kilobolt.com`

.