
	<b>JURUSAN TEKNIK</b> <b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	:	Praktik Pemrograman Mobile 2
	<b>Dosen Pengampu</b>	:	Sigit Kariagil Bimonugroho, S.Kom., M.T.
	<b>Sub CPMK</b>	:	Arsitektur Aplikasi (Lapisan UI)



## LAPORAN PRAKTIKUM

IDENTITAS MAHASISWA	
<b>Nama Mahasiswa</b>	: Firza Aji Karya
<b>NPM</b>	: 223307007
<b>Kelas</b>	: 3A

DASAR TEORI
<p>Arsitektur aplikasi adalah seperangkat aturan desain. Mirip dengan cetak biru sebuah rumah, arsitektur memberikan struktur bagi aplikasi Anda. Arsitektur aplikasi yang baik dapat membuat kode Anda andal, fleksibel, skalabel, dan mudah dikelola selama bertahun-tahun.</p> <p>Mempelajari cara menggunakan ViewModel, salah satu komponen Arsitektur untuk menyimpan data aplikasi Anda. Data yang disimpan tidak akan hilang jika framework menghancurkan dan membuat ulang aktivitas dan fragmen selama perubahan konfigurasi atau peristiwa lainnya.</p> <p>LiveData adalah class holder data yang dapat diamati dan peka terhadap siklus proses. Beberapa karakteristik LiveData:</p> <ul style="list-style-type: none"> <li>- LiveData menyimpan data; LiveData adalah wrapper yang dapat digunakan dengan semua jenis data.</li> <li>- LiveData dapat diamati, yang berarti bahwa observer akan diberi tahu saat data yang dimiliki objek LiveData berubah.</li> </ul>

ALAT DAN BAHAN
<ol style="list-style-type: none"> <li>1. Laptop</li> <li>2. Android Studio</li> <li>3. GitHub</li> </ol>

KODE PROGRAM DAN HASIL
<p>4. Menambahkan LiveData ke kata acak saat ini</p> <ul style="list-style-type: none"> <li>• Di GameViewModel, ubah jenis variabel <code>_currentScrambledWord</code> menjadi <code>MutableLiveData&lt;String&gt;</code>. LiveData dan <code>MutableLiveData</code> adalah class generik, jadi Anda perlu menentukan jenis data yang disimpannya.</li> <li>• Ubah jenis variabel <code>_currentScrambledWord</code> menjadi <code>val</code> karena nilai objek LiveData/MutableLiveData</li> <li>• Ubah jenis kolom cadangan <code>currentScrambledWord</code> ke <code>LiveData&lt;String&gt;</code>, karena tidak dapat diubah.</li> </ul> <pre>private val _currentScrambledWord = MutableLiveData&lt;String&gt;() val currentScrambledWord: LiveData&lt;String&gt;     get() = _currentScrambledWord</pre>

	<b>JURUSAN TEKNIK</b>		
	<b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	: Praktik Pemrograman Mobile 2	
	<b>Dosen Pengampu</b>	: Sigit Kariagil Bimonugroho,S.Kom., M.T.	
	<b>Sub CPMK</b>	: Arsitektur Aplikasi (Lapisan UI)	

- Untuk mengakses data dalam objek `LiveData`, gunakan properti `value`. Di `GameViewModel` dalam metode `getNextWord()`, dalam blok `else`, ubah referensi `_currentScrambledWord` menjadi `_currentScrambledWord.value`.

```

    } else {
        _currentScrambledWord.value = String(tempWord)
        ++_currentWordCount
        wordsList.add(currentWord)
    }

```

#### 5. Melampirkan observer ke objek LiveData

- Di `GameFragment`, hapus metode `updateNextWordOnScreen()` dan semua panggilan ke dalamnya. Anda tidak memerlukan metode ini, karena Anda akan melampirkan observer ke `LiveData`.
- Di `onSubmitWord()`, ubah blok `if-else` kosong sebagai berikut. Metode yang lengkap akan terlihat seperti ini.

```

private fun onSubmitWord() {
    val playerWord = binding.textInputEditText.text.toString()

    if (viewModel.isUserWordCorrect(playerWord)) {
        setErrorTextField(false)
        if (!viewModel.nextWord()) {
            showFinalScoreDialog()
        }
    } else {
        setErrorTextField(true)
    }
}

```

- Teruskan `viewLifecycleOwner` sebagai parameter pertama ke metode `observe()`. `viewLifecycleOwner` merepresentasikan siklus proses Tampilan Fragment. Parameter ini membantu `LiveData` mengetahui siklus proses `GameFragment` dan memberi tahu observer hanya jika `GameFragment` dalam status aktif
- Tambahkan lambda sebagai parameter kedua dengan `newWord` sebagai parameter fungsi.
- Dalam isi fungsi ekspresi lambda, tetapkan `newWord` ke tampilan teks kata acak.



**JURUSAN TEKNIK  
PROGRAM STUDI D-III TEKNOLOGI INFORMASI**

**Mata Kuliah** : Praktik Pemrograman Mobile 2  
**Dosen Pengampu** : Sigit Kariagil Bimonugroho, S.Kom., M.T.  
**Sub CPMK** : Arsitektur Aplikasi (Lapisan UI)



```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
    super.onCreateView(view, savedInstanceState)  
  
    // Setup a click listener for the Submit and Skip buttons.  
    binding.submit.setOnClickListener { onSubmitWord() }  
    binding.skip.setOnClickListener { onSkipWord() }  
    // Update the UI  
    // Observe the currentScrambledWord LiveData.  
    viewModel.currentScrambledWord.observe(viewLifecycleOwner,  
        { newWord ->  
            binding.textViewUnscrambledWord.text = newWord  
        })  
}
```

6. Memasang observer ke skor dan jumlah kata

- Dalam `GameViewModel`, ubah jenis variabel class `_score` dan `_currentWordCount` menjadi `val`.
- Ubah jenis data variabel `_score` dan `_currentWordCount` menjadi `MutableLiveData`, lalu inisialisasikan ke 0.
- Ubah jenis kolom cadangan ke `LiveData<Int>`.

```
class GameViewModel : ViewModel() {  
    private val _score = MutableLiveData( value: 0)  
    val score: LiveData<Int>  
        get() = _score  
  
    private val _currentWordCount = MutableLiveData( value: 0)  
    val currentWordCount: LiveData<Int>  
        get() = _currentWordCount  
}
```

- Di `GameViewModel` pada awal metode `reinitializeData()`, ubah referensi `_score` dan `_currentWordCount` masing-masing menjadi `_score.value` dan `_currentWordCount.value`.

```
fun reinitializeData() {  
    _score.value = 0  
    _currentWordCount.value = 0  
    wordsList.clear()  
    getNextWord()  
}
```



**JURUSAN TEKNIK  
PROGRAM STUDI D-III TEKNOLOGI INFORMASI**

**Mata Kuliah** : Praktik Pemrograman Mobile 2  
**Dosen Pengampu** : Sigit Kariagil Bimonugroho, S.Kom., M.T.  
**Sub CPMK** : Arsitektur Aplikasi (Lapisan UI)



- Di GameViewModel, dalam metode nextWord(), ubah referensi `currentWordCount` menjadi `_currentWordCount`.

```
fun nextWord(): Boolean {  
    return if (_currentWordCount.value!! < MAX_NO_OF_WORDS) {  
        getNextWord()  
        true  
    } else false  
}
```

- Di GameViewModel, di dalam metode increaseScore() dan getNextWord(), ubah referensi `_score` dan `_currentWordCount` menjadi `_score.value` dan `_currentWordCount.value`.

```
private fun increaseScore() {  
    _score.value = (_score.value)?.plus(SCORE_INCREASE)  
}
```

- gunakan fungsi Kotlin `inc()` untuk meningkatkan nilai satu per satu dengan keamanan null.

```
private fun getNextWord() {  
    currentWord = allWordsList.random()  
    val tempWord = currentWord.toCharArray()  
    tempWord.shuffle()  
    while (String(tempWord).equals(currentWord, ignoreCase: false)) {  
        tempWord.shuffle()  
    }  
    if (wordsList.contains(currentWord)) {  
        getNextWord()  
    } else {  
        _currentScrambledWord.value = String(tempWord)  
        _currentWordCount.value = (_currentWordCount.value)?.inc()  
        wordsList.add(currentWord)  
    }  
}
```

- Di GameFragment, akses nilai `score` menggunakan properti `value`. Dalam metode `showFinalScoreDialog()`, ubah `viewModel.score` menjadi `viewModel.score.value`.



**JURUSAN TEKNIK  
PROGRAM STUDI D-III TEKNOLOGI INFORMASI**

**Mata Kuliah** : Praktik Pemrograman Mobile 2  
**Dosen Pengampu** : Sigit Kariagil Bimonugroho, S.Kom., M.T.  
**Sub CPMK** : Arsitektur Aplikasi (Lapisan UI)





```
private fun showFinalScoreDialog() {  
    MaterialAlertDialogBuilder(requireContext())  
        .setTitle(getString(R.string.congratulations))  
        .setMessage(getString(R.string.you_scored, viewModel.score.value))  
        .setCancelable(false)  
        .setNegativeButton("Exit") { _, _ ->  
            exitGame()  
        }  
        .setPositiveButton("Play Again") { _, _ ->  
            restartGame()  
        }  
        .show()  
}
```

- Di `GameFragment` dalam metode `onViewCreated()`, hapus kode yang memperbarui tampilan teks skor dan jumlah kata.
- Pada `GameFragment` di akhir metode `onViewCreated()`, lampirkan observer untuk `score`. Teruskan `viewLifecycleOwner` sebagai parameter pertama ke observer dan ekspresi lambda untuk parameter kedua.
- Di akhir metode `onViewCreated()`, lampirkan observer untuk `currentWordCount` `LiveData`. Teruskan `viewLifecycleOwner` sebagai parameter pertama ke observer dan ekspresi lambda untuk parameter kedua. Di dalam ekspresi lambda, teruskan jumlah kata baru sebagai parameter dan di isi fungsi, tetapkan jumlah kata baru dengan `MAX_NO_OF_WORDS`

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
    super.onViewCreated(view, savedInstanceState)  
  
    // Setup a click listener for the Submit and Skip buttons.  
    binding.submit.setOnClickListener { onSubmitWord() }  
    binding.skip.setOnClickListener { onSkipWord() }  
    // Update the UI  
    // Observe the currentScrambledWord LiveData.  
    viewModel.currentScrambledWord.observe(viewLifecycleOwner, { newWord ->  
        binding.textViewUnscrambledWord.text = newWord  
    })  
    viewModel.score.observe(viewLifecycleOwner, { newScore ->  
        binding.score.text = getString(R.string.score, newScore)  
    })  
    viewModel.currentWordCount.observe(viewLifecycleOwner, { newWordCount ->  
        binding.wordCount.text =  
            getString(R.string.word_count, newWordCount, MAX_NO_OF_WORDS)  
    })  
}
```

## 7. Menggunakan LiveData dengan data binding

	<b>JURUSAN TEKNIK</b> <b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	:	Praktik Pemrograman Mobile 2
	<b>Dosen Pengampu</b>	:	Sigit Kariagil Bimonugroho, S.Kom., M.T.
	<b>Sub CPMK</b>	:	Arsitektur Aplikasi (Lapisan UI)

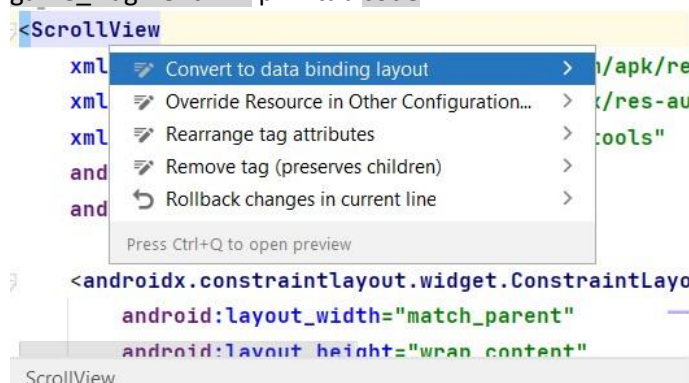
- Di file `build.gradle(Module)`, aktifkan properti `dataBinding` pada bagian `buildFeatures`.

```

buildFeatures {
    dataBinding true
}

```

- Mengkonversikan file tata letak menjadi tata letak data binding dengan cara buka `game_fragment.xml` pilih tab `code`



- Di `GameFragment`, pada awal metode `onCreateView()`, ubah pembuatan instance variabel `binding` untuk menggunakan data binding.

```



override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    // Inflate the layout XML file and return a binding object instance
    binding = DataBindingUtil.inflate(inflater, R.layout.game_fragment, container, attachToParent: false)
}

```

## 8. Menambahkan variable data binding

- Di `game_fragment.xml`, di dalam tag `<data>`, tambahkan tag turunan yang disebut `<variable>`, deklarasikan properti yang disebut `gameViewModel` dan jenis `GameViewModel`. Hal tersebut digunakan untuk mengikat data di `ViewModel` ke tata letak
- Di bawah deklarasi `gameViewModel`, tambahkan variabel lain di dalam tag `<data>` jenis `Integer`, dan beri nama `maxNoOfWords`. Anda akan menggunakan ini untuk mengikat ke variabel di `ViewModel` guna menyimpan jumlah kata per game.



	<b>JURUSAN TEKNIK</b> <b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	:	Praktik Pemrograman Mobile 2
	<b>Dosen Pengampu</b>	:	Sigit Kariagil Bimonugroho, S.Kom., M.T.
	<b>Sub CPMK</b>	:	Arsitektur Aplikasi (Lapisan UI)

```

<data>
    <variable
        name="gameViewModel"
        type="com.example.android.unscramble.ui.game.GameViewModel" />
    <variable
        name="maxNoOfWords"
        type="int" />

```

```

</data>

```

- Pada GameFragment di awal metode `onViewCreated()`, inialisasi variabel tata letak `gameViewModel` dan `maxNoOfWords`.

```

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.gameViewModel = viewModel
    binding.maxNoOfWords = MAX_NO_OF_WORDS
    binding.lifecycleOwner = viewLifecycleOwner

    // Setup a click listener for the Submit and Skip buttons.
    binding.submit.setOnClickListener { onSubmitWord() }
    binding.skip.setOnClickListener { onSkipWord() }
}

```

#### 9. Menggunakan ekspresi binding

- Di `game_fragment.xml`, tambahkan atribut `text` ke tampilan teks `textView_unscrambled_word`. Gunakan variabel tata letak baru, `gameViewModel` dan tetapkan `@{gameViewModel.currentScrambledWord}` ke atribut `text`.

```

<TextView
    android:id="@+id/textView_unscrambled_word"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{gameViewModel.currentScrambledWord}"

```

- Perbarui atribut `text` untuk tampilan teks `score` dengan ekspresi binding berikut. Gunakan resource string `score` dan teruskan `gameViewModel.score` sebagai parameter resource.



```

<TextView
    android:id="@+id/score"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{@string/score(gameViewModel.score)}"

```

#### 10. Menguji aplikasi Unscramble dengan Talkback diaktifkan

- Dalam `GameViewModel`, konversikan kata acak `String` menjadi string `Spannable`. String `spannable` adalah string yang berisi beberapa informasi tambahan.

	<b>JURUSAN TEKNIK</b> <b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	:	Praktik Pemrograman Mobile 2
	<b>Dosen Pengampu</b>	:	Sigit Kariagil Bimonugroho, S.Kom., M.T.
	<b>Sub CPMK</b>	:	Arsitektur Aplikasi (Lapisan UI)

```

val currentScrambledWord: LiveData<Spannable> = Transformations.map(_currentScrambledWord) {
    if (it == null) {
        SpannableString( source: "") ^map
    } else {
        val scrambledWord = it.toString()
        val spannable: Spannable = SpannableString(scrambledWord)
        spannable.setSpan(
            TtsSpan.VerbatimBuilder(scrambledWord).build(),
            0,
            scrambledWord.length,
            Spannable.SPAN_INCLUSIVE_INCLUSIVE
        )
        spannable ^map
    }
}

```

#### 11. Menghapus kode yang tidak digunakan

- Dalam `GameFragment`, hapus metode `getNextScrambledWord()` dan `onDetach()`.

```

private fun getNextScrambledWord(): String {
    val tempWord = allWordsList.random().toCharArray()
    tempWord.shuffle()
    return String(tempWord)
}

override fun onDetach() {
    super.onDetach()
    Log.d( tag: "GameFragment", msg: "GameFragment destroyed!")
}

```



- Dalam `GameViewModel`, hapus metode `onCleared()`.

```


override fun onCleared() {
    super.onCleared()
    Log.d( tag: "GameFragment", msg: "GameViewModel destroyed!")
}

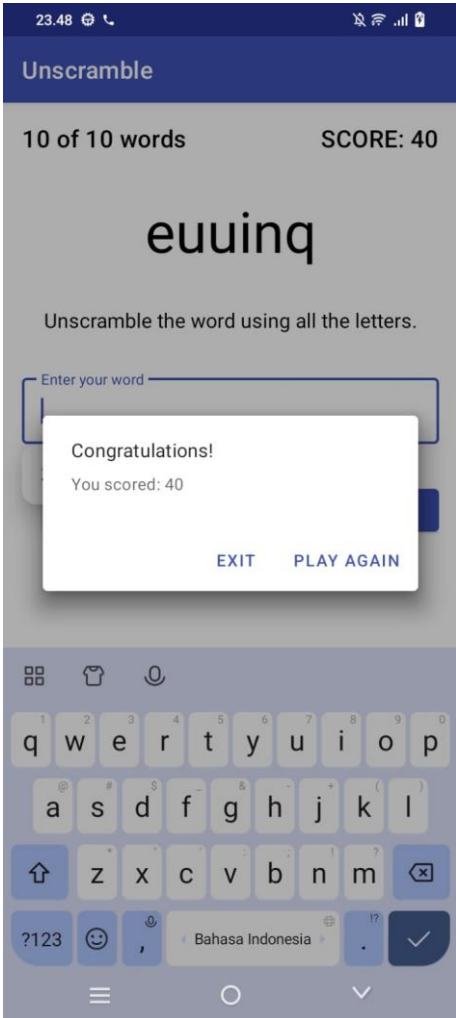
```



	<b>JURUSAN TEKNIK</b> <b>PROGRAM STUDI D-III TEKNOLOGI INFORMASI</b>		
	<b>Mata Kuliah</b>	:	Praktik Pemrograman Mobile 2
	<b>Dosen Pengampu</b>	:	Sigit Kariagil Bimonugroho, S.Kom., M.T.
	<b>Sub CPMK</b>	:	Arsitektur Aplikasi (Lapisan UI)

**Hasil Akhir :**





KESIMPULAN
<p>Pada praktikum ini kita dapat mengetahui apa yang dimaksud dengan LiveData, kita dapat mengerti bagaimana cara menambahkan LiveData ke kata acak saat ini, Melampirkan observer ke objek LiveData, Menggunakan LiveData dengan data binding, Menambahkan variabel data binding dsb.</p>