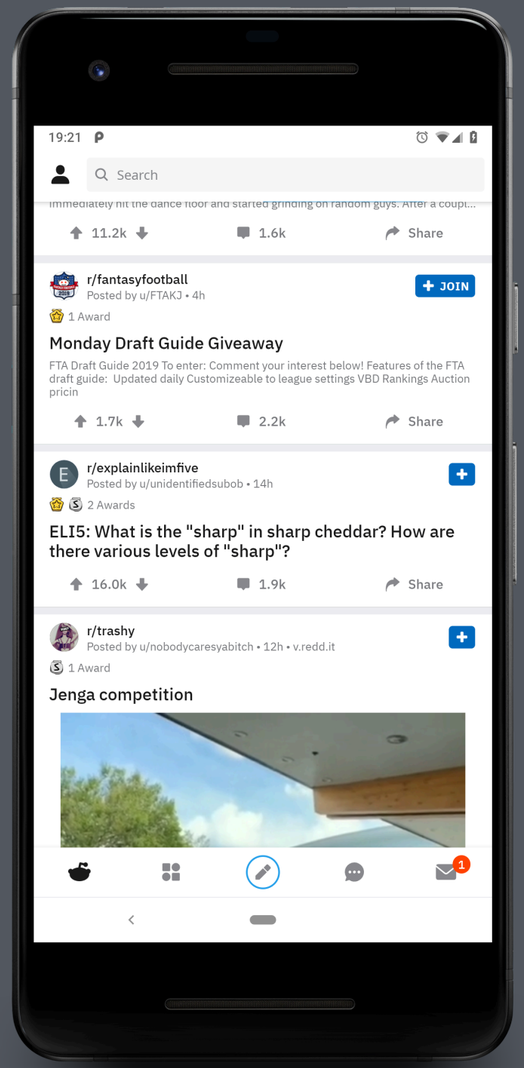
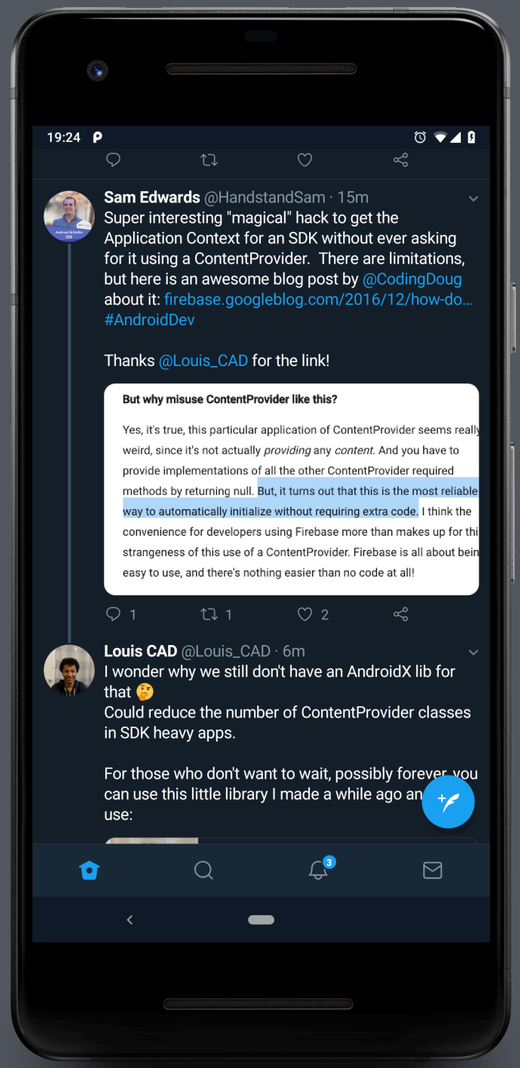
IMMOBILIEN SCOUT 24

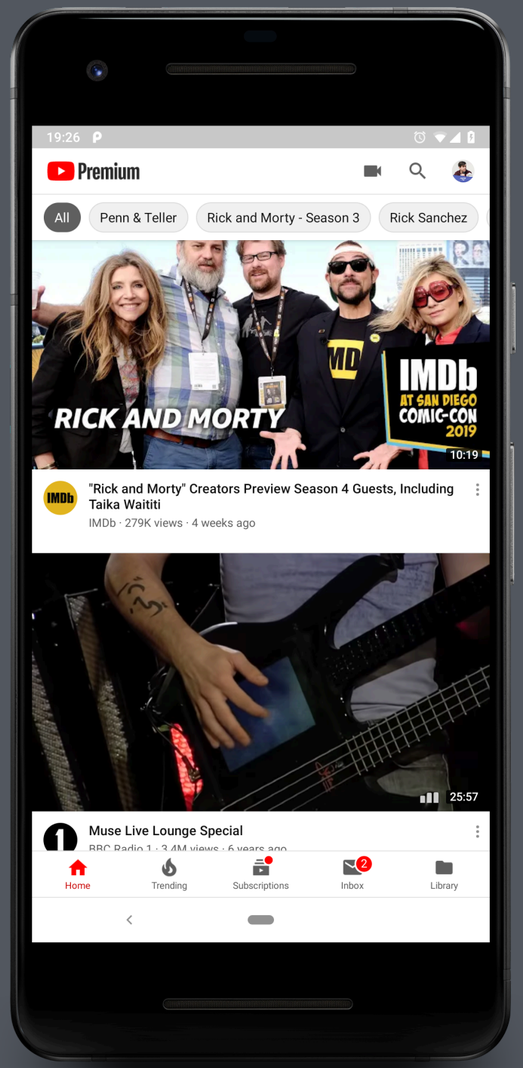
Implementing the Paging Library

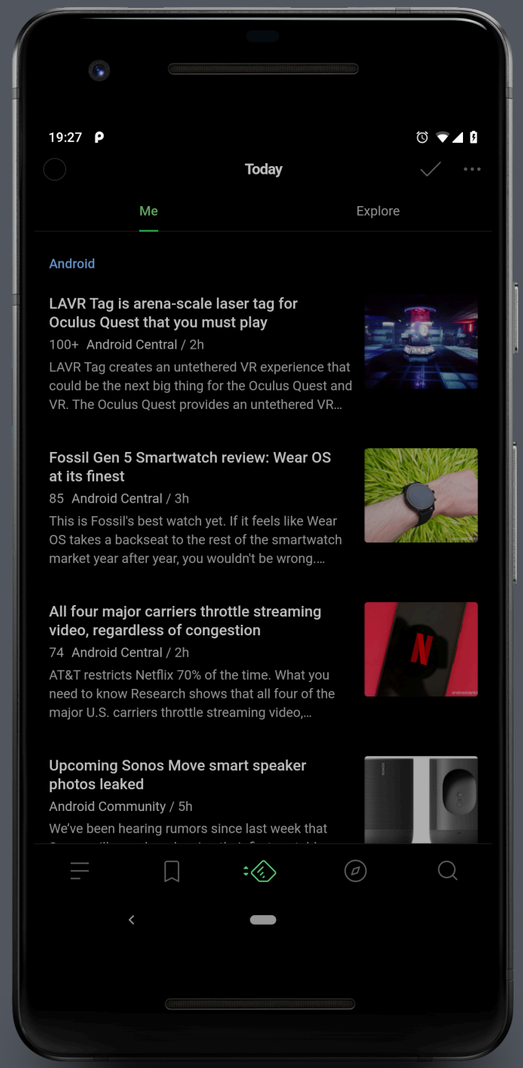
Droidcon Lisbon 🇵🇹

@askashdavies


19:21
Search
11.2k
1.6k
Share
r/fantasyfootball
Posted by u/FTAKJ • 4h
JOIN
1 Award
Monday Draft Guide Giveaway
FTA Draft Guide 2019 To enter: Comment your interest below! Features of the FTA draft guide: Updated daily Customizeable to league settings VBD Rankings Auction pricin
1.7k
2.2k
Share
r/explainlikeimfive
Posted by u/unidentifiedsubob • 14h
2 Awards
ELI5: What is the "sharp" in sharp cheddar? How are there various levels of "sharp"?
16.0k
1.9k
Share
r/trashy
Posted by u/nobodycaresyabitch • 12h • v.redd.it
1 Award
Jenga competition

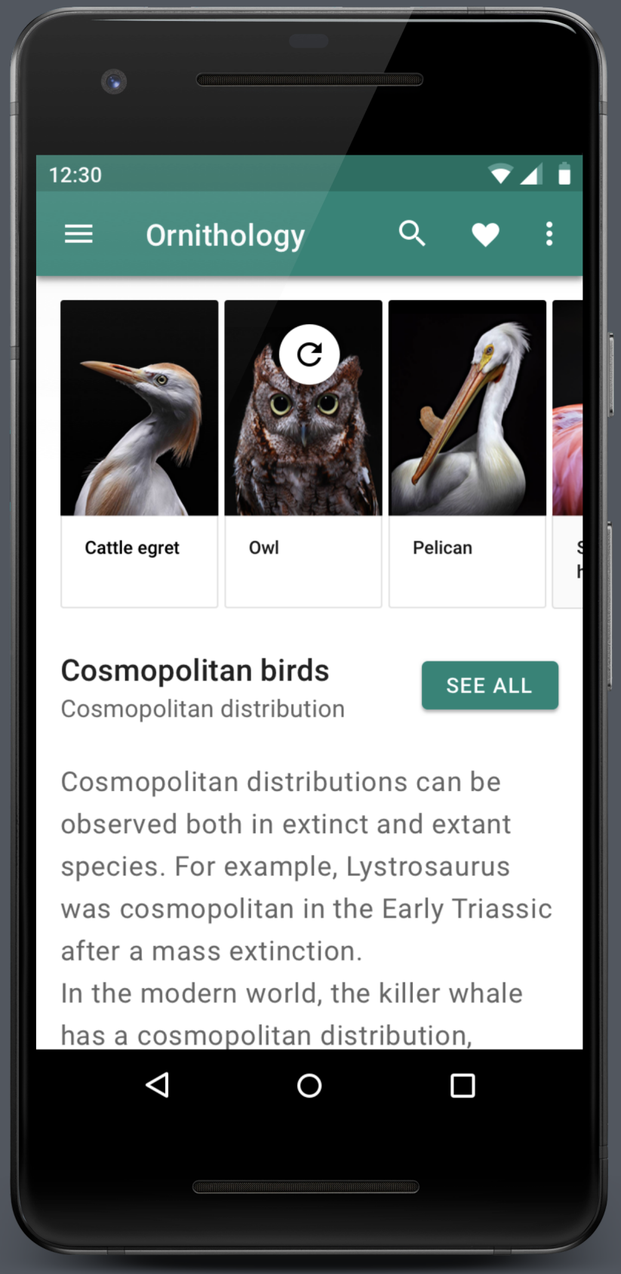

19:24
Sam Edwards @HandstandSam · 15m
Super interesting "magical" hack to get the Application Context for an SDK without ever asking for it using a ContentProvider. There are limitations, but here is an awesome blog post by @CodingDoug about it: firebase.googleblog.com/2016/12/how-do... #AndroidDev
Thanks @Louis_CAD for the link!
But why misuse ContentProvider like this?
Yes, it's true, this particular application of ContentProvider seems really weird, since it's not actually providing any content. And you have to provide implementations of all the other ContentProvider required methods by returning null. But, it turns out that this is the most reliable way to automatically initialize without requiring extra code. I think the convenience for developers using Firebase more than makes up for this strangeness of this use of a ContentProvider. Firebase is all about being easy to use, and there's nothing easier than no code at all!
1
1
2
Louis CAD @Louis_CAD · 6m
I wonder why we still don't have an AndroidX lib for that
Could reduce the number of ContentProvider classes in SDK heavy apps.
For those who don't want to wait, possibly forever, you can use this little library I made a while ago and use:


19:26
Premium
All
Penn & Teller
Rick and Morty - Season 3
Rick Sanchez
RICK AND MORTY
IMDb
AT SAN DIEGO COMIC-CON 2019
10:19
"Rick and Morty" Creators Preview Season 4 Guests, Including Taika Waititi
IMDb · 279K views · 4 weeks ago
25:57
Muse Live Lounge Special
Home
Trending
Subscriptions
Inbox
Library


19:27
Today
Me
Explore
Android
LAVR Tag is arena-scale laser tag for Oculus Quest that you must play
100+ Android Central / 2h
LAVR Tag creates an untethered VR experience that could be the next big thing for the Oculus Quest and VR. The Oculus Quest provides an untethered VR...
Fossil Gen 5 Smartwatch review: Wear OS at its finest
85 Android Central / 3h
This is Fossil's best watch yet. If it feels like Wear OS takes a backseat to the rest of the smartwatch market year after year, you wouldn't be wrong....
All four major carriers throttle streaming video, regardless of congestion
74 Android Central / 2h
AT&T restricts Netflix 70% of the time. What you need to know Research shows that all four of the major U.S. carriers throttle streaming video,...
Upcoming Sonos Move smart speaker photos leaked
Android Community / 5h
We've been hearing rumors since last week that

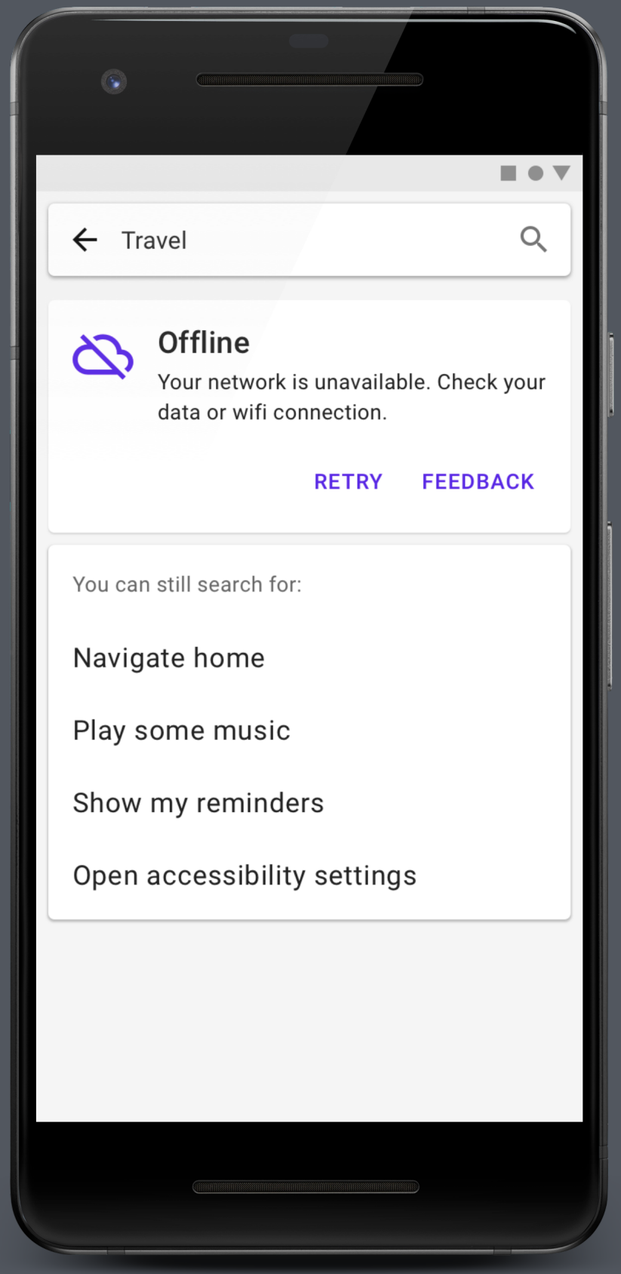
Challenges 🤔

Up-To-Date

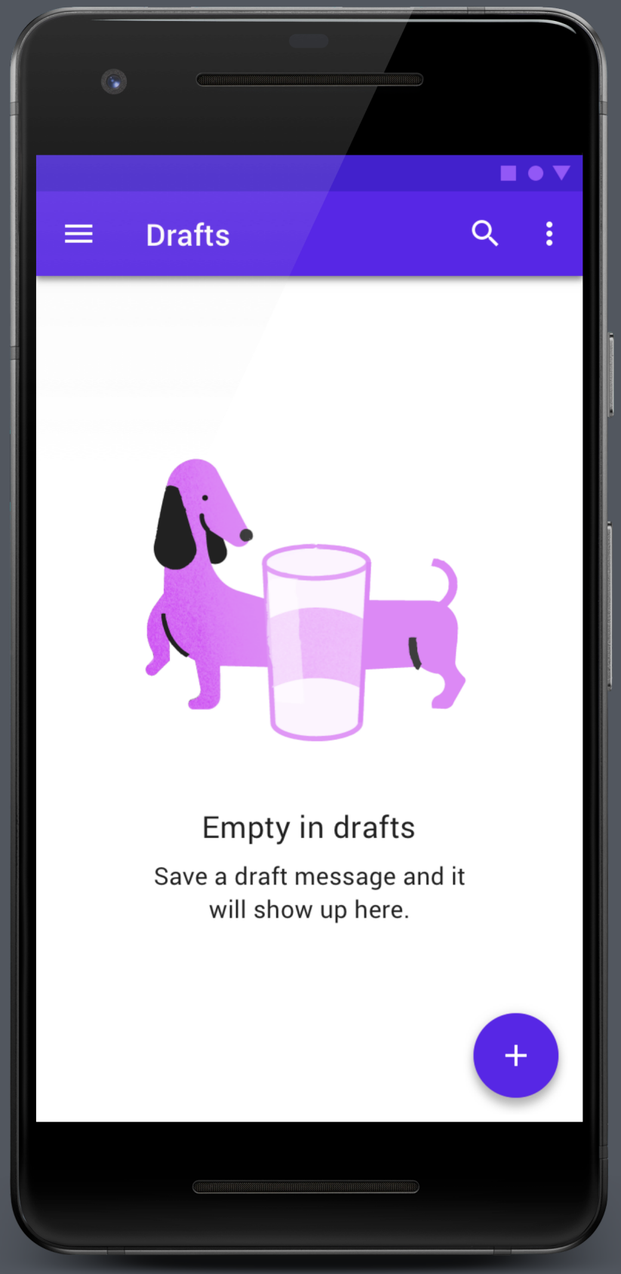

12:30
Ornithology
Cattle egret
Owl
Pelican
Cosmopolitan birds
Cosmopolitan distribution
SEE ALL
Cosmopolitan distributions can be observed both in extinct and extant species. For example, Lystrosaurus was cosmopolitan in the Early Triassic after a mass extinction.
In the modern world, the killer whale has a cosmopolitan distribution,

Large Data-Sets

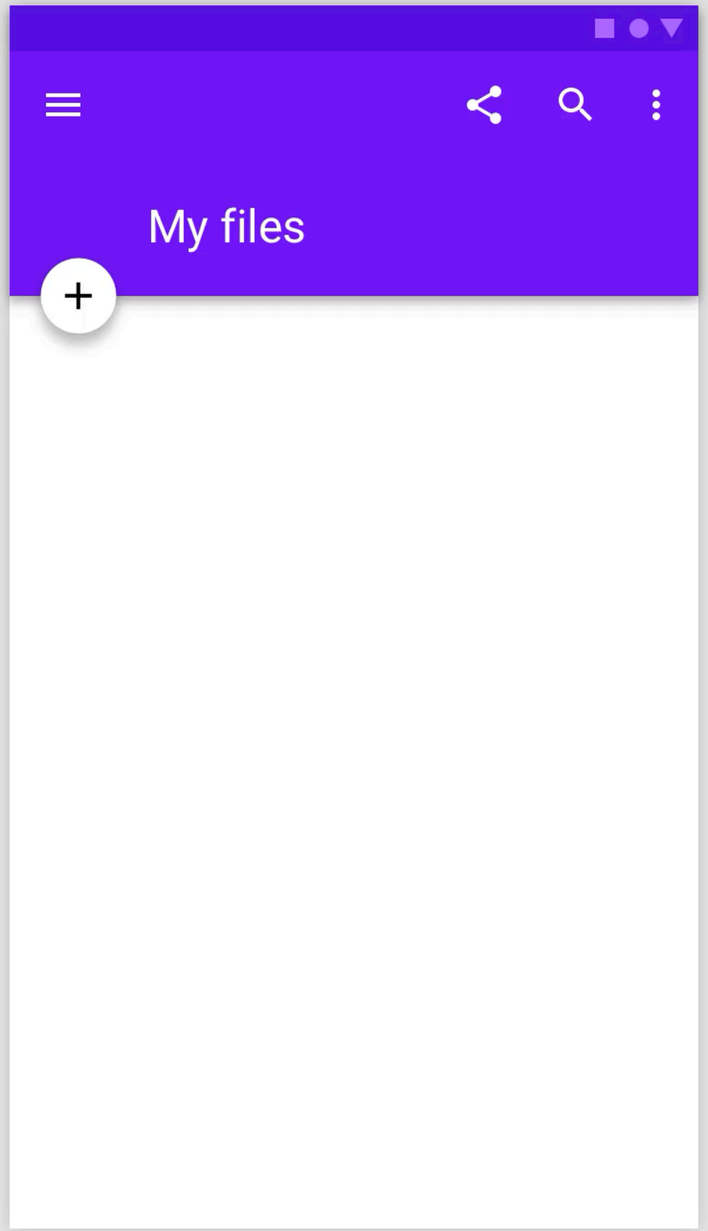
Offline


Travel
Offline
Your network is unavailable. Check your data or wifi connection.
RETRY
FEEDBACK
You can still search for:
Navigate home
Play some music
Show my reminders
Open accessibility settings

State


Drafts
Empty in drafts
Save a draft message and it will show up here.

Progress



ListView

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

ArrayAdapter

```
// extends BaseAdapter>

list.adapter = ArrayAdapter<String>(
    context,
    R.layout.simple_list_item_1,
    arrayOf("Kotlin", "Java" /* ... */)
)
```

BaseAdapter

```
class ListAdapter : BaseAdapter() {

  override fun getView(position: Int, convertView: View, container: ViewGroup) {
    val view = convertView ?: layoutInflater.inflate(
        R.layout.simple_list_item_1,
        container,
        false
    )

    convertView
        .findViewById(R.id.text1)
        .text = getItem(position)

    return convertView
  }
}
```

BaseAdapter

```
class ListAdapter() : BaseAdapter() {

  var items: List<String> = emptyList()
    set(value) {
      field = value
      notifyDataSetChanged()
    }

  override fun getCount(): Int = items.size

  override fun getItem(position: Int): String = items[position]

  override fun getView(position: Int, convertView: View, container: ViewGroup) {
    /* ... */
  }
}
```

Pagination 

Paging 

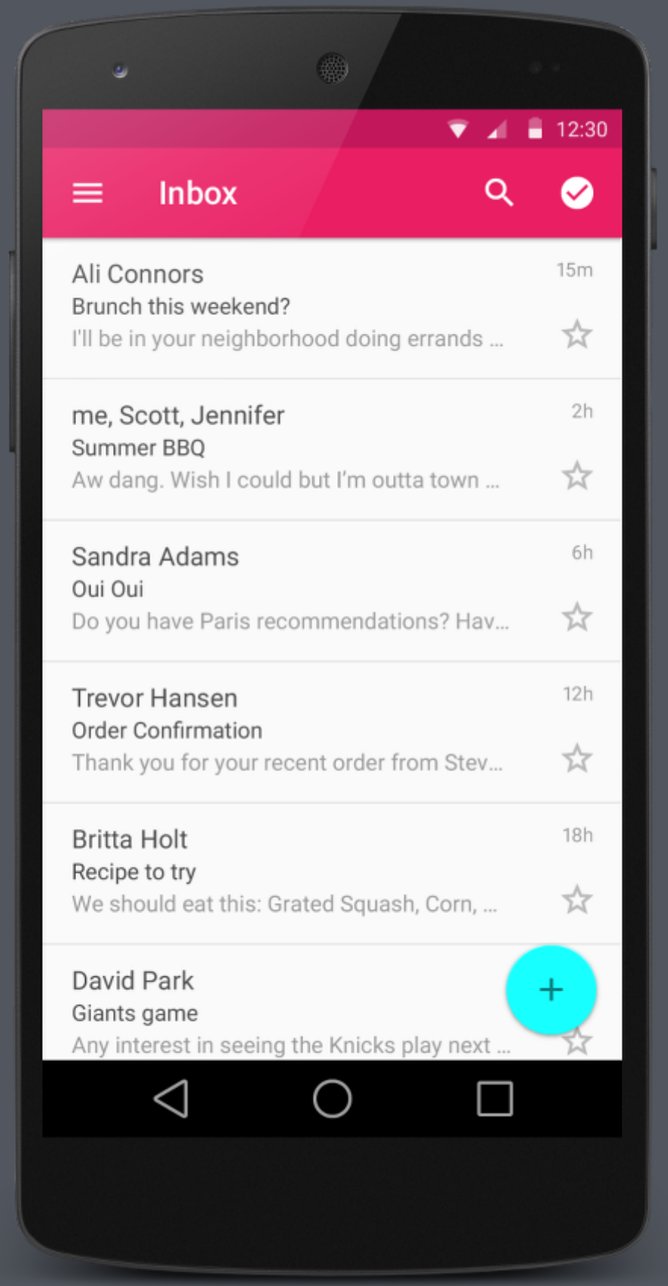
OnScrollListener

@askashdavies

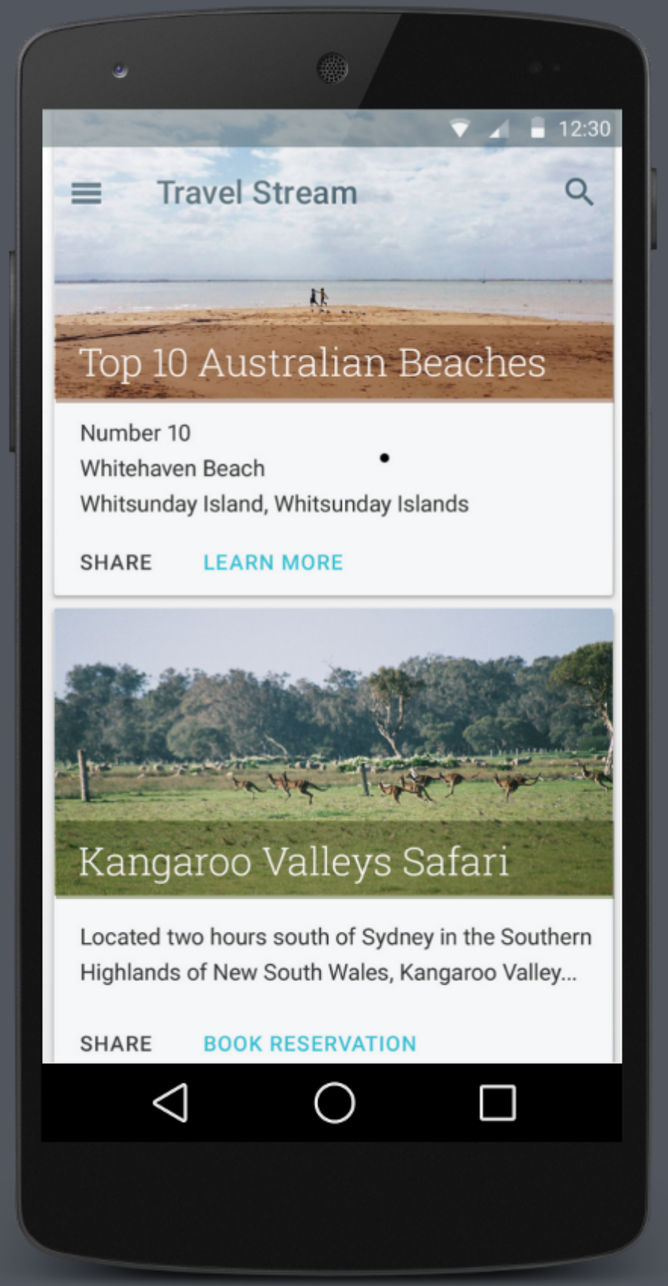
BaseAdapter / ListView

- Manages list of it's own data
- Manages view inflation and configuration
- Notify entire data set of change
- Not capable of diffing items

RecyclerView



RecyclerView



Paging 

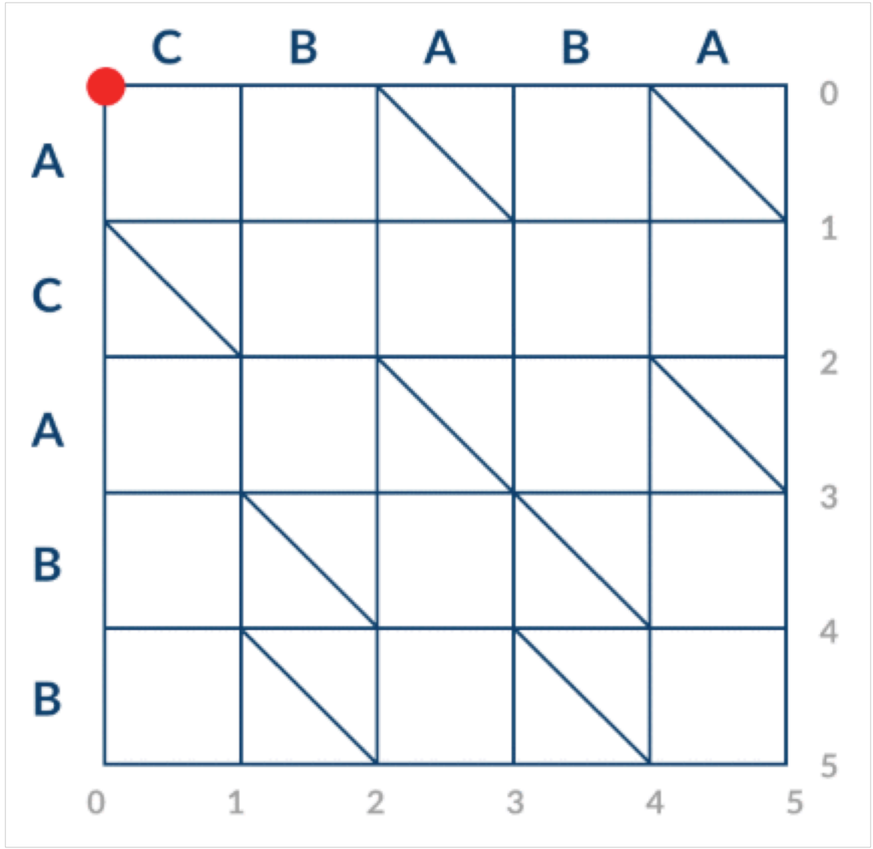
`AsyncListUtil`

Diffing

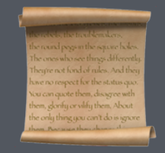
`DiffUtil` / `AsyncListDiffer`

DiffUtil

Myers Diff Algorithm



medium.com/skyrise/the-myers-diff-algorithm-and-kotlin-observable-properties-69dfb18541b

ListAdapter 

ListAdapter 📜

Immutability 💪

ListAdapter

`submitList(...)`

Migration

`ListAdapter<T>`

RecyclerView.Adapter

```
class UserAdapter : RecyclerView.Adapter<UserViewHolder>() {

  private var items: List<User> = emptyList()

  override fun getItemCount() = items.size

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  fun updateList(items: List<User>) {
    val result: DiffResult = DiffUtil.calculate(DiffCallback(this.items, items))
    result.dispatchUpdatesTo(this)
  }

  class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

RecyclerView.Adapter

```
class UserAdapter : RecyclerView.Adapter<UserViewHolder>() {

  private var items: List<User> = emptyList()

  override fun getItemCount() = items.size

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  fun updateList(items: List<User>) {
    val result: DiffResult = DiffUtil.calculate(UserComparator(this.items, items))
    result.dispatchUpdatesTo(this)
  }

  class UserViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

DiffUtil.Callback

```
class UserComparator(
    private val oldItems: List<User>,
    private val newItems: List<User>
) : DiffUtil.Callback() {

  override fun getOldListSize(): Int = oldItems.size

  override fun getNewListSize(): Int = newItems.size

  override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
    return oldItems[oldItemPosition].id == newItems[newItemPosition].id
  }

  override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
    return oldItems[oldItemPosition] == newItems[newItemPosition]
  }
}
```

DiffUtil.Callback

```
class UserComparator(
    private val oldItems: List<User>,
    private val newItems: List<User>
) : DiffUtil.Callback() {

  override fun getOldListSize(): Int = oldItems.size

  override fun getNewListSize(): Int = newItems.size

  override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
    return oldItems[oldItemPosition].id == newItems[newItemPosition].id
  }

  override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
    return oldItems[oldItemPosition] == newItems[newItemPosition]
  }
}
```

DiffUtil.ItemCallback<User>

```
object UserComparator : DiffUtil.ItemCallback<User>() {

  override fun areItemsTheSame(oldItem: User, newItem: User): Boolean {
    return oldItem.id == newItem.id
  }

  override fun areContentsTheSame(oldItem: User, newItem: User): Boolean {
    return oldItem == newItem
  }
}
```

DiffUtil.ItemCallback<User>

```
object UserComparator : DiffUtil.ItemCallback<User>() {

  override fun areItemsTheSame(oldItem: User, newItem: User): Boolean {
    return oldItem.id == newItem.id
  }

  override fun areContentsTheSame(oldItem: User, newItem: User): Boolean {
    return oldItem == newItem
  }
}
```

RecyclerView.Adapter

```
class UserAdapter : RecyclerView.Adapter<UserViewHolder>() {

  private var items: List<User> = emptyList()

  override fun getItemCount() = items.size

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  fun updateList(items: List<User>) {
    /* ... */
  }

  class UserViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

ListAdapter

```
class UserAdapter : ListAdapter<User, UserViewHolder>(UserComparator) {

  private var items: List<User> = emptyList()

  override fun getItemCount() = items.size

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  fun updateList(items: List<User>) {
    /* ... */
  }

  class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

ListAdapter

```
class UserAdapter : ListAdapter<User, UserViewHolder>(UserComparator) {

  private var items: List<User> = emptyList()

  override fun getItemCount() = items.size

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  fun updateList(items: List<User>) {
    /* ... */
  }

  class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

ListAdapter

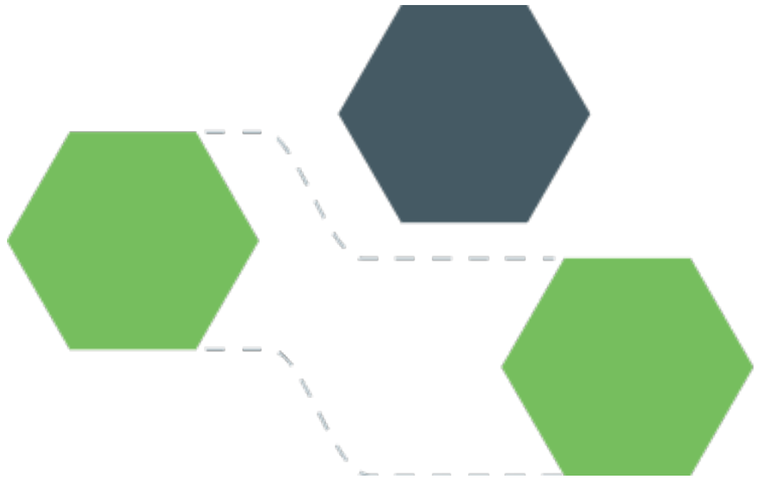
```
class UserAdapter : ListAdapter<User, UserViewHolder>(UserComparator) {

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(items[position])
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    /* ... */
  }

  class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {

    fun bind(item: User) {
      /* ... */
    }
  }
}
```

@askashdavies

ListAdapter 💪

Android JetPack

Foundation Components



Android JetPack

Architecture Components

Android JetPack

Behaviour Components

Android JetPack

UI Components

Android JetPack

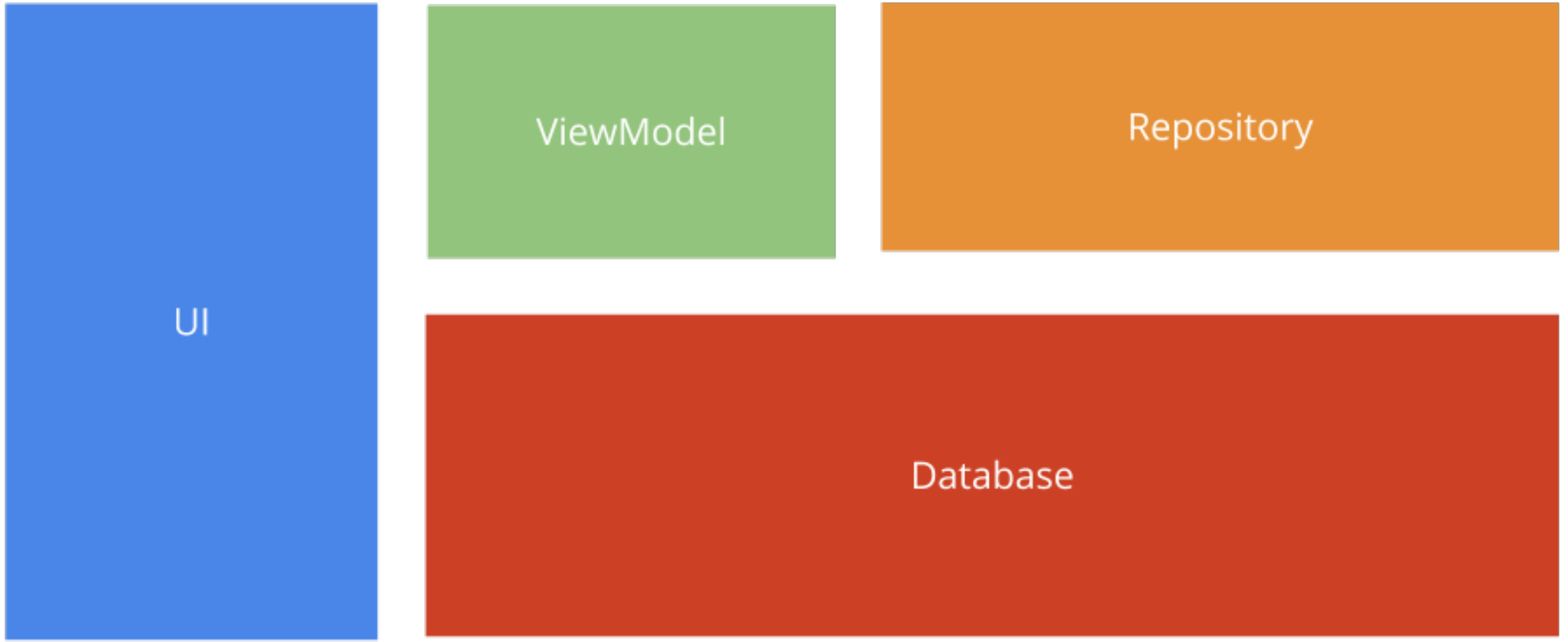
Paging Library

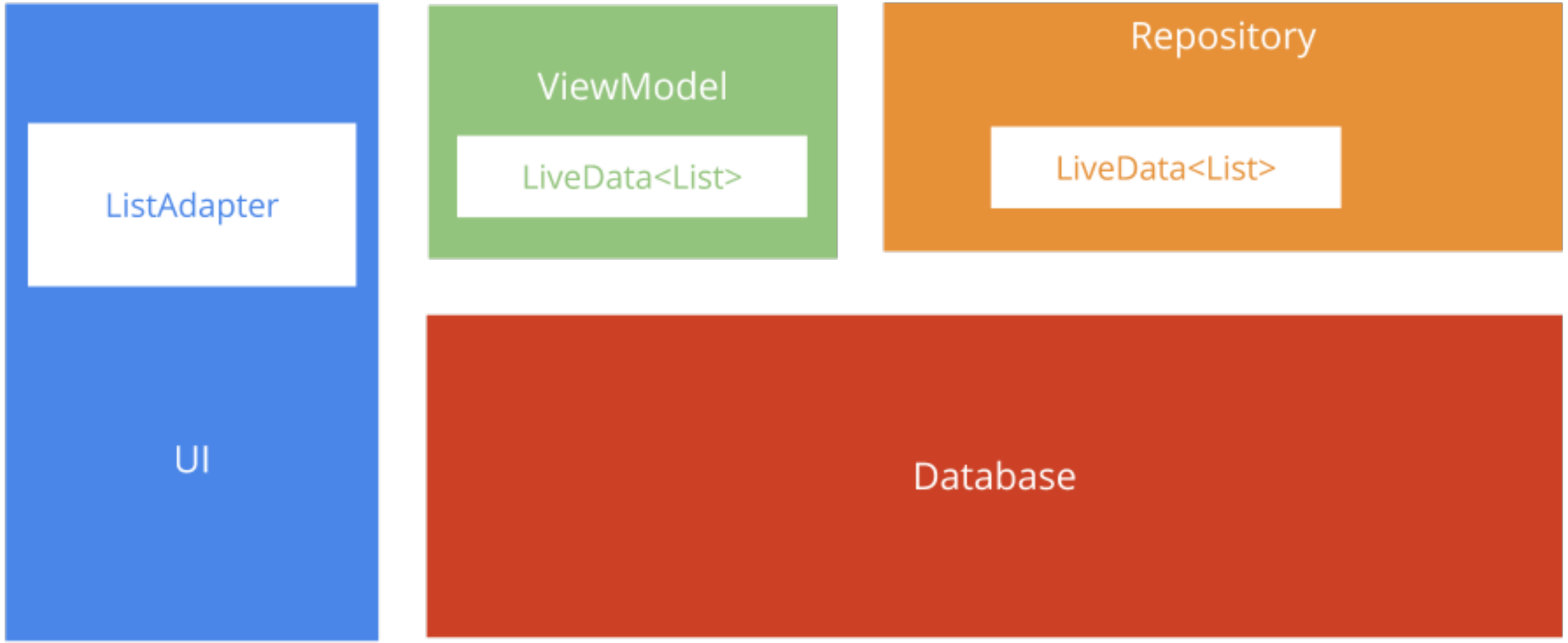
@askashdavies

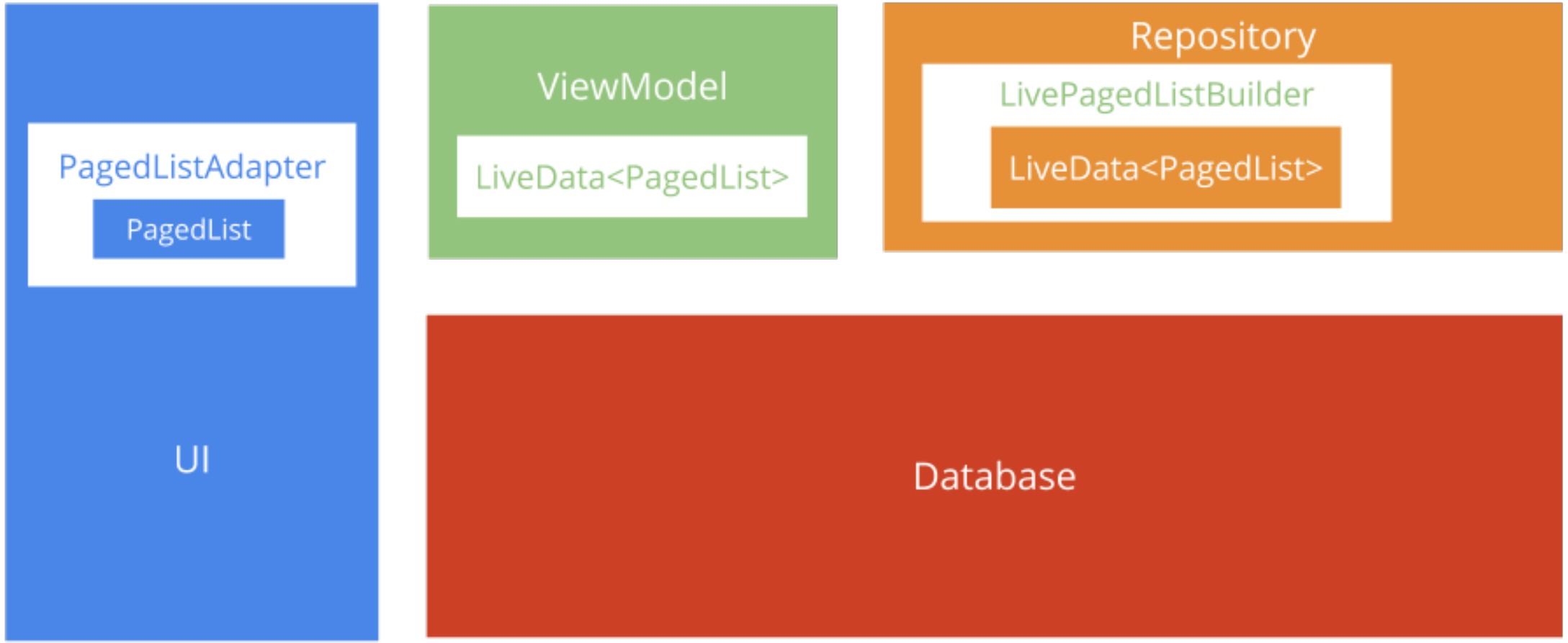
Android JetPack 🚀

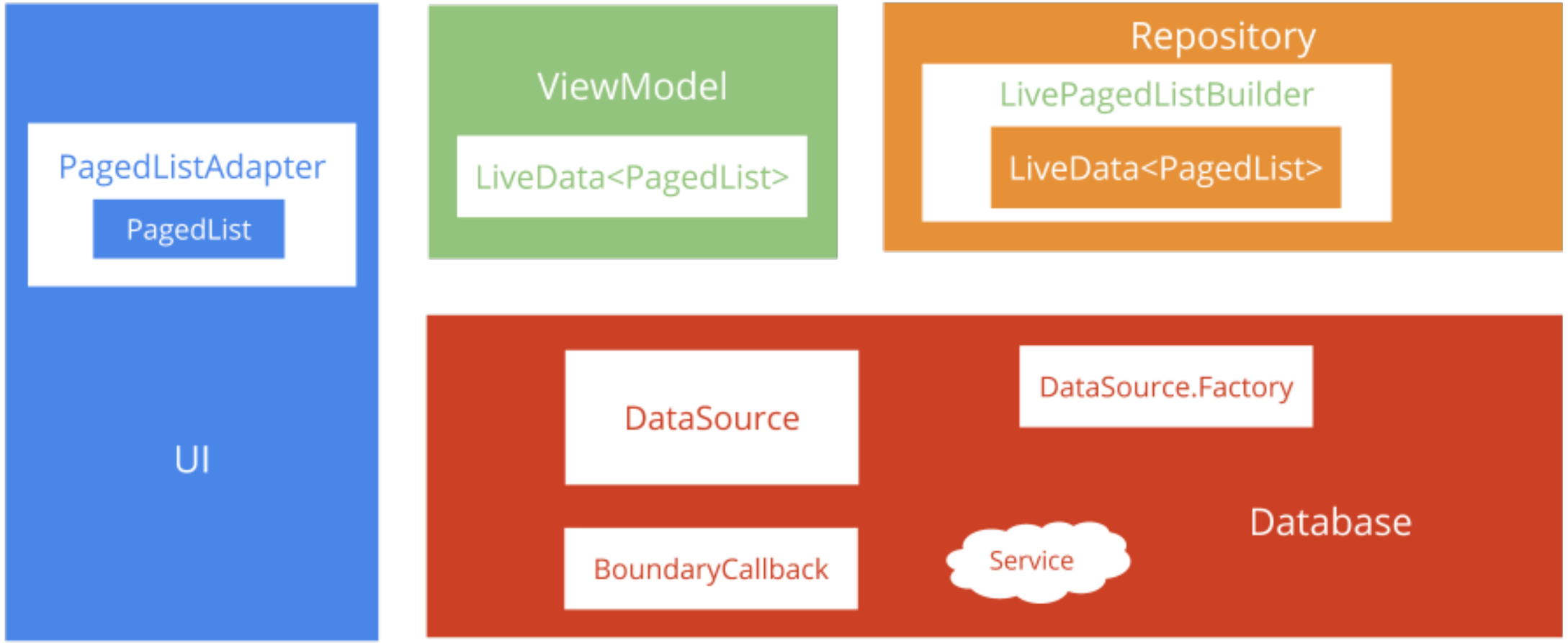
Paging Library

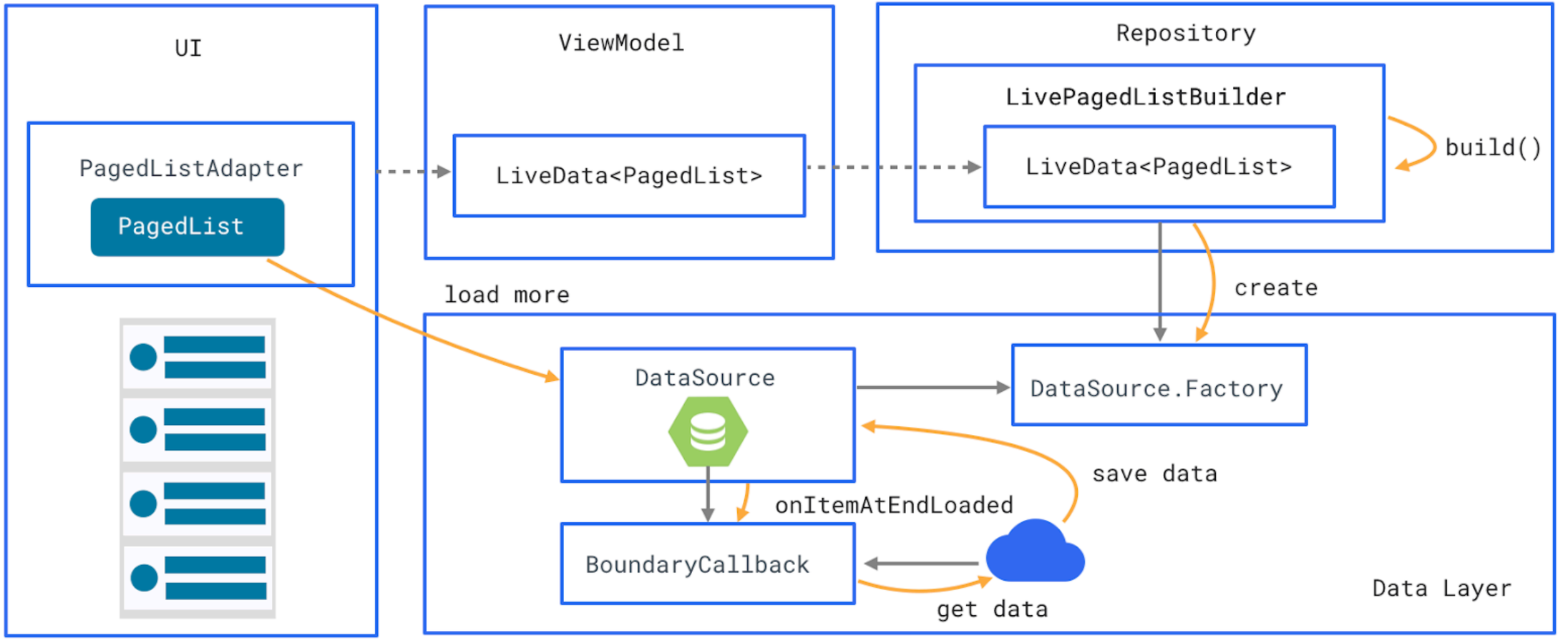
- `PagedListAdapter` ⚙️
- `PagedList` 📄
- `DataSource` / `DataSource.Factory` ⛲
- `BoundaryCallback` 🏁


UI
ViewModel
Repository
Database

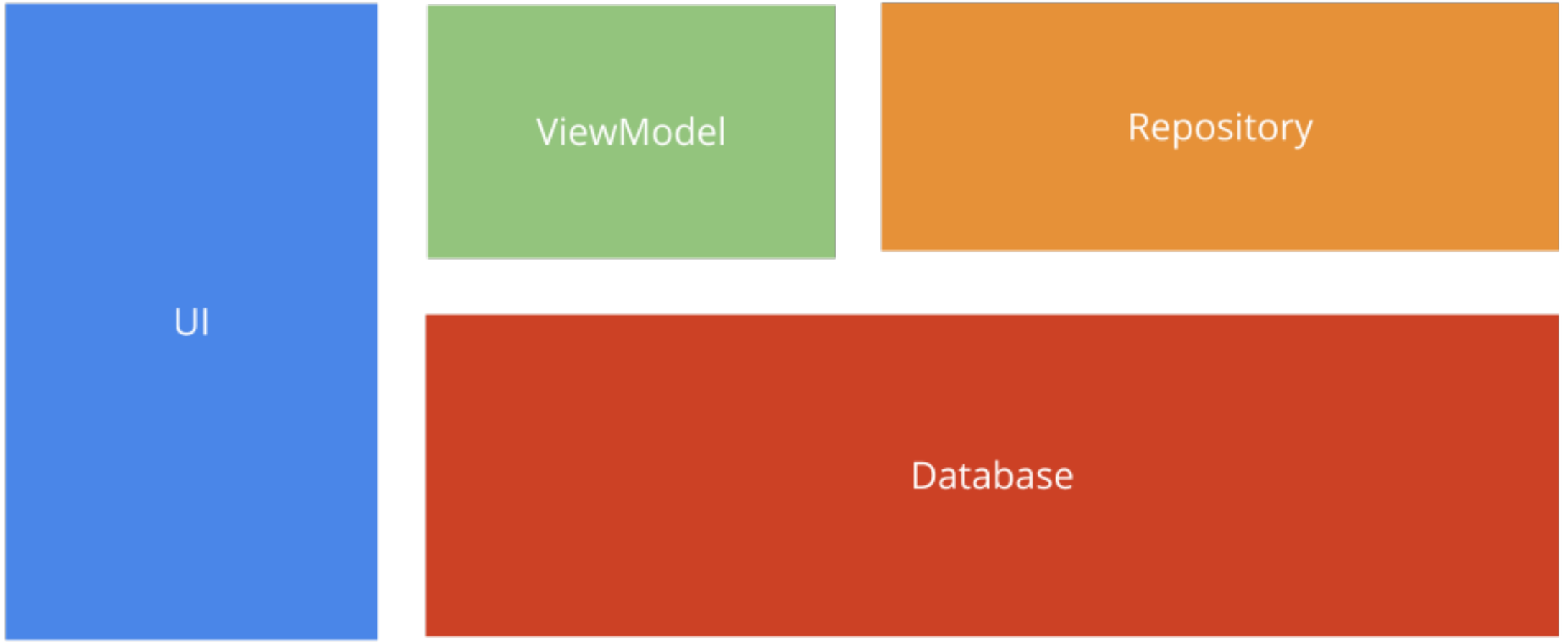

ListAdapter
UI
ViewModel
LiveData<List>
Repository
LiveData<List>
Database

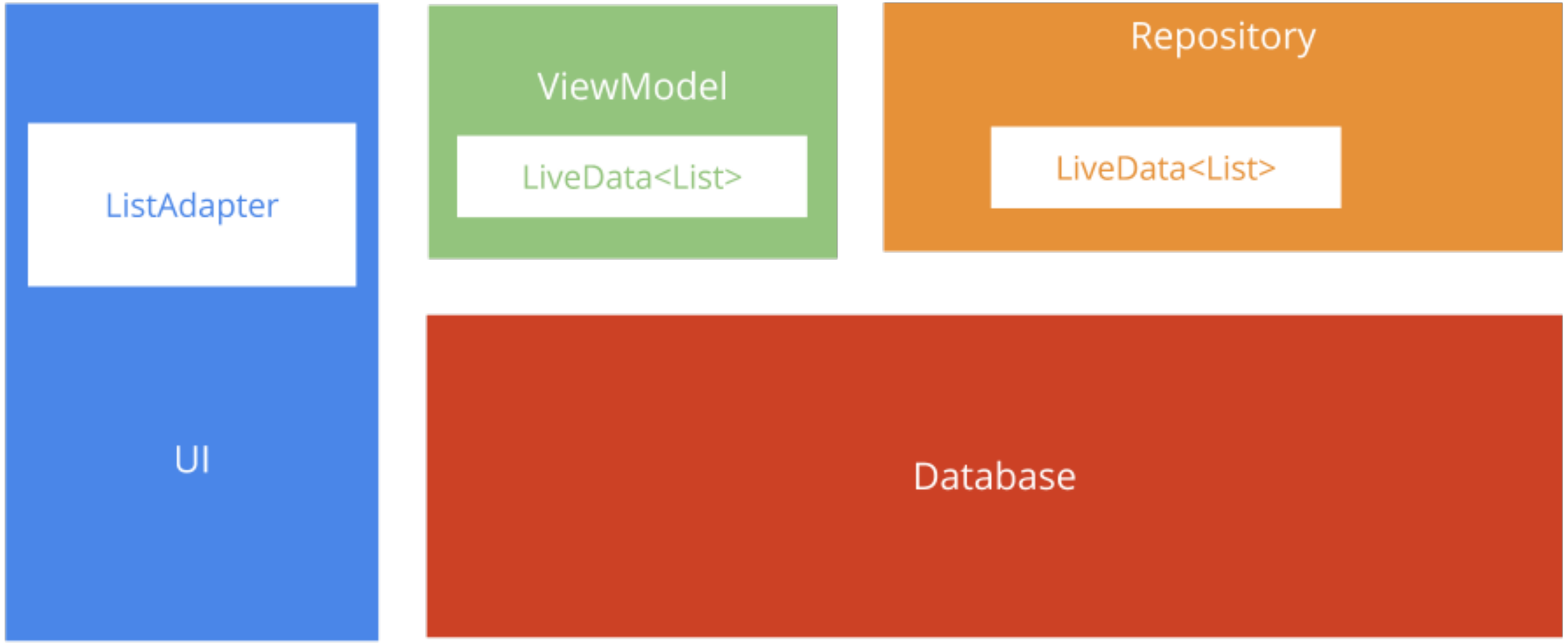

PagedListAdapter
PagedList
UI
ViewModel
LiveData<PagedList>
Repository
LivePagedListBuilder
LiveData<PagedList>
Database

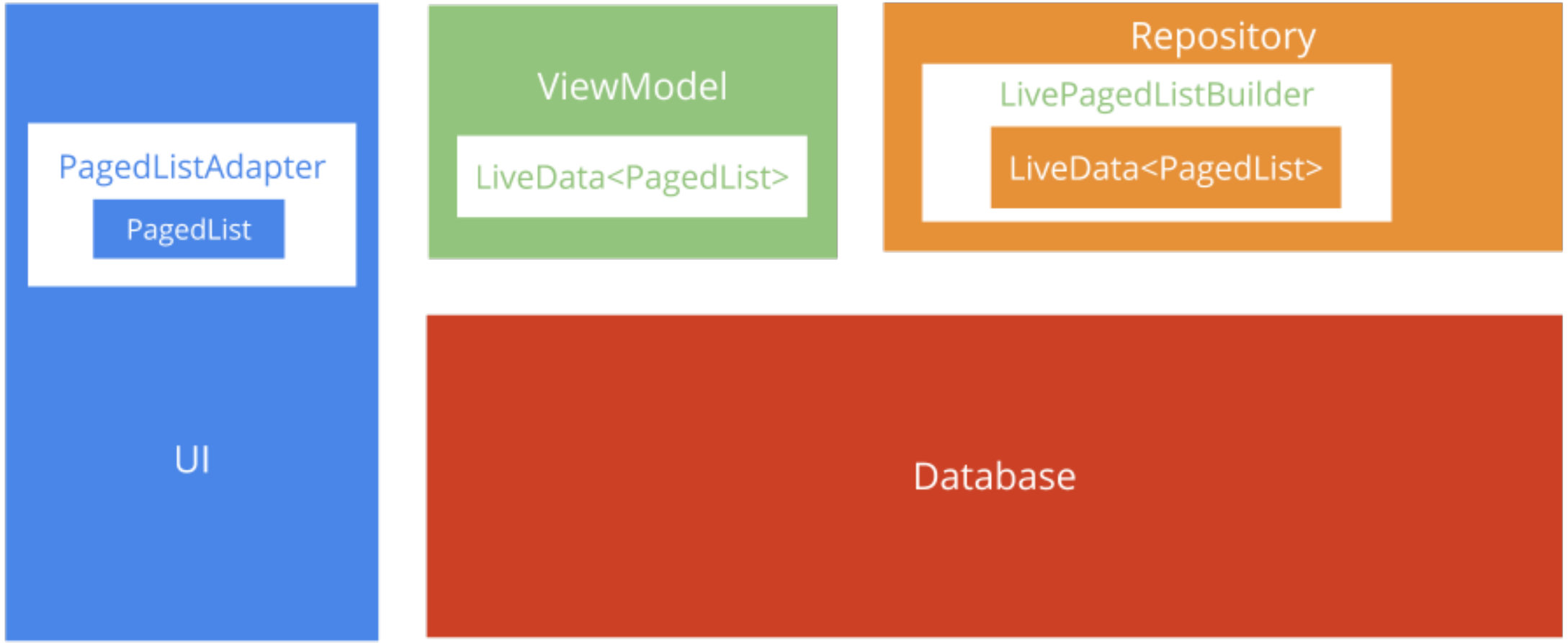

PagedListAdapter
PagedList
UI
ViewModel
LiveData<PagedList>
Repository
LivePagedListBuilder
LiveData<PagedList>
DataSource
DataSource.Factory
BoundaryCallback
Service
Database


UI
PagedListAdapter
PagedList
ViewModel
LiveData<PagedList>
Repository
LivePagedListBuilder
LiveData<PagedList>
build()
load more
create
DataSource
DataSource.Factory
save data
onItemAtEndLoaded
BoundaryCallback
get data
Data Layer

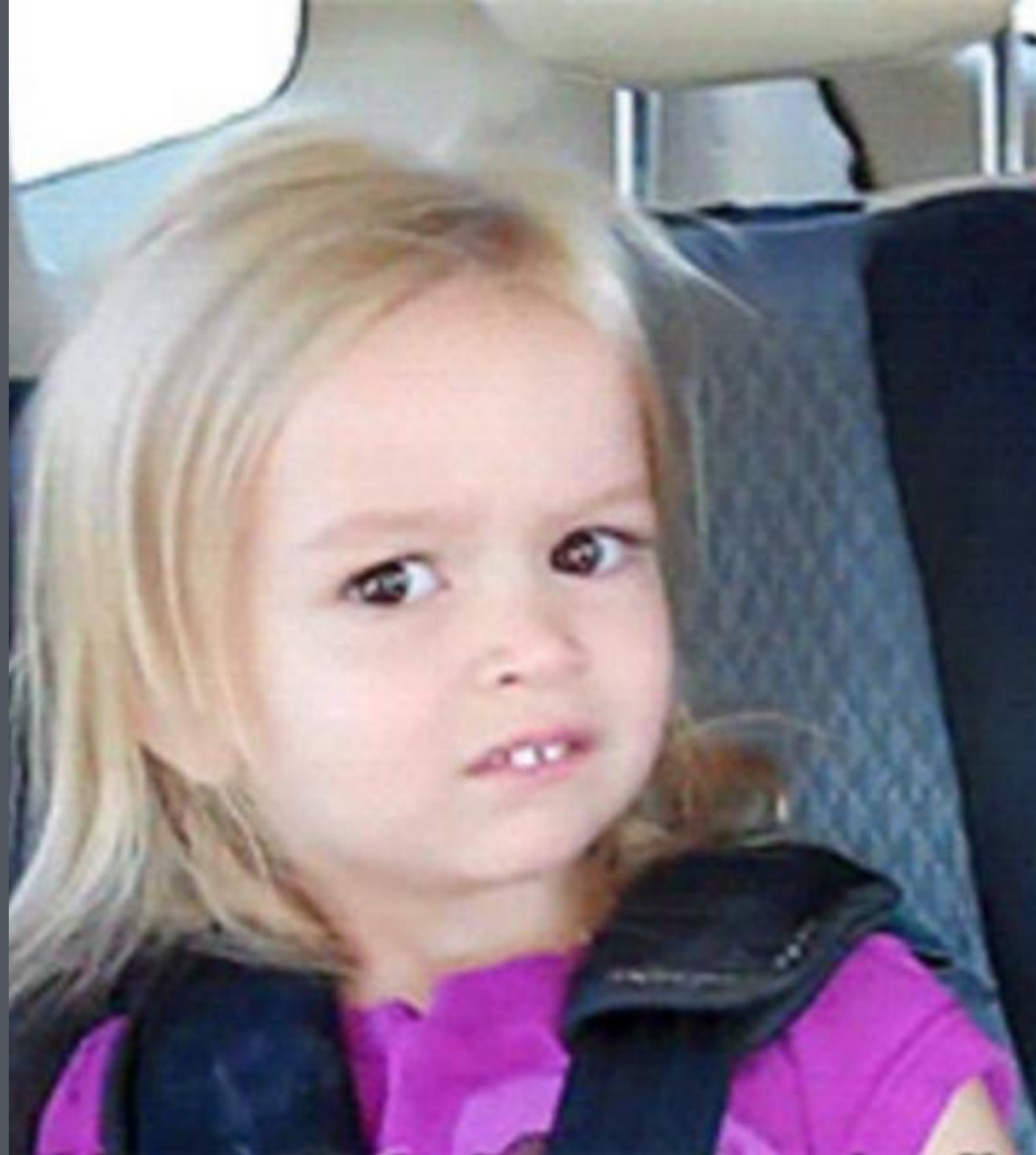
@askashdavies


UI
ViewModel
Repository
Database


ListAdapter
UI
ViewModel
LiveData<List>
Repository
LiveData<List>
Database


PagedListAdapter
PagedList
UI
ViewModel
LiveData<PagedList>
Repository
LivePagedListBuilder
LiveData<PagedList>
Database

PagedList ?



PagedList

```
PagedList<T> : List<T>
```

`PagedList`

`PagedListBuilder` 

- Data sources / cache management
- Page size / prefetch distance
- Offline characteristics
- Loading behaviour

PagedList

PagedListBuilder 🏗

- LiveDataPagedListBuilder
- RxPagedListBuilder
- FlowPagedListBuilder[1]

---

[1] github.com/chrisbanes/tivi/blob/master/data-android/src/main/java/app/tivi/data/FlowPagedListBuilder.kt

Observability

`PagedList`

PagedList

LiveDataPagedListBuilder

PagedList

LiveDataPagedListBuilder 🏗

```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    /* ... */
  }
}
```


@askashdavies

PagedList

LiveDataPagedListBuilder 🏗

```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    return LivePagedListBuilder(factory, PAGE_SIZE).build()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

PagedList.Config

LiveDataPagedListBuilder 🏗

```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .build()

    return LivePagedListBuilder(factory, config).build()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

PagedList.Config

LiveDataPagedListBuilder 

```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .setInitialLoadSizeHint(50)
        .build()

    return LivePagedListBuilder(factory, config).build()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

PagedList.Config

LiveDataPagedListBuilder 🏗

```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .setInitialLoadSizeHint(50)
        .setPrefetchDistance(10)
        .build()

    return LivePagedListBuilder(factory, config).build()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

PagedList.Config

LiveDataPagedListBuilder 

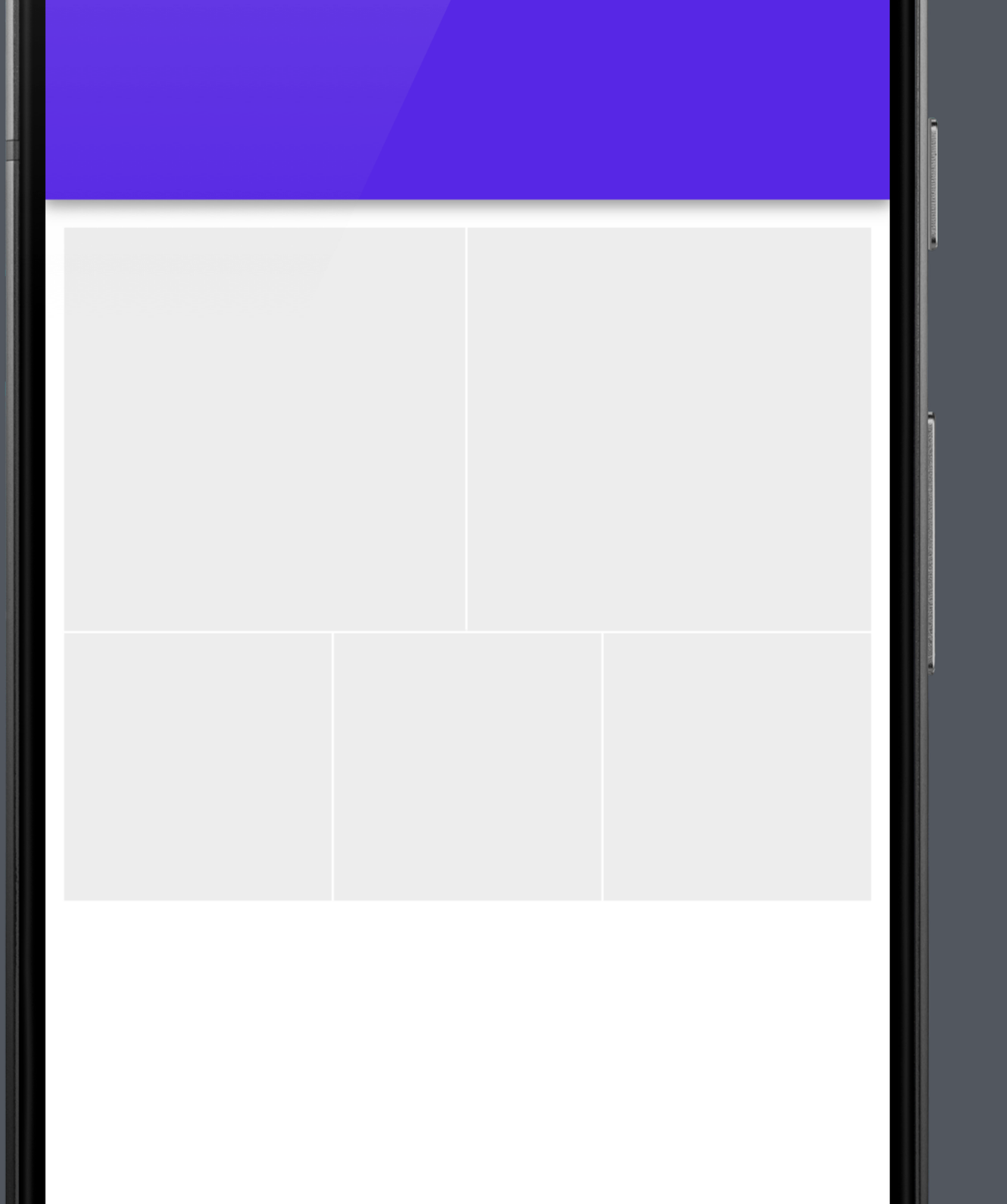
```
class UserRepository(private val service: UserService) {

  fun users(): LiveData<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .setInitialLoadSizeHint(50)
        .setPrefetchDistance(10)
        .setEnablePlaceholders(false)
        .build()

    return LivePagedListBuilder(factory, config).build()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

@askashdavies

Placeholders

Advantages

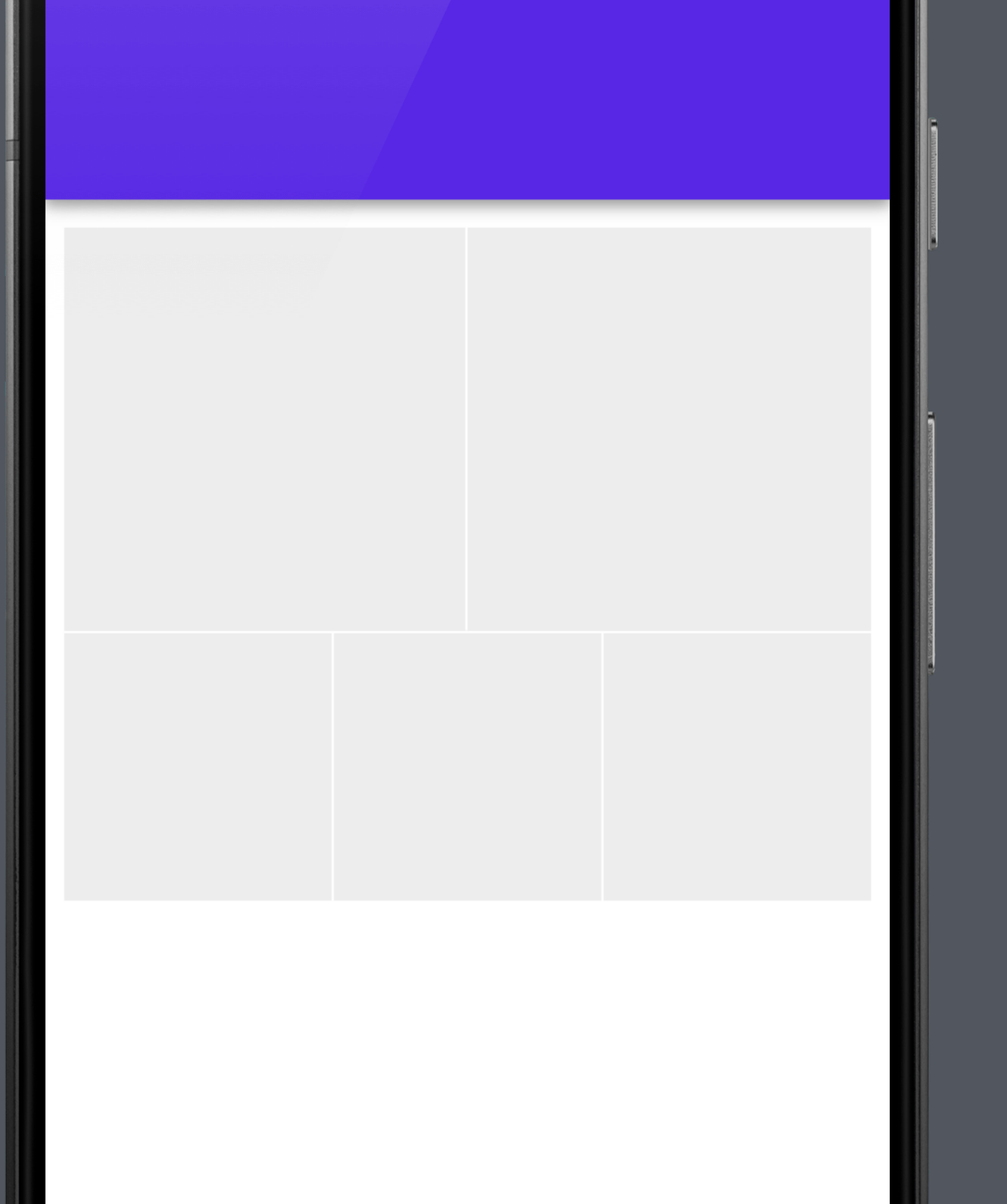
- Continuous scrolling
- Less abrupt UI changes
- Scrollbars maintain consistency
- Accurately indicate loading state



Placeholders

Disadvantages

- Irregular sized items cause UI jank
- Prepare view holder without item
- Data set must be quantifiable



PagedList.Config

RxPagedListBuilder 🏗

```
class UserRepository(private val service: UserService) {

  fun users(): Observable<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .build()

    return RxPagedListBuilder(factory, config)
        .buildObservable() // or buildFlowable()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

@askashdavies

Coroutines

FlowPagedListBuilder 💪

```
class UserRepository(private val service: UserService) {

  fun users(): Flow<PagedList<User>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .build()

    return FlowPagedListBuilder(factory, config).buildFlow()
  }

  companion object {

    private const val PAGE_SIZE = 20
  }
}
```

github.com/chrisbanes/tivi/blob/master/data-android/src/main/java/app/tivi/data/FlowPagedListBuilder.kt

DataSource.Factory

Paging ❤ Room

Paging ❤ Room

```
@Dao
interface UserDao {

  @Query("SELECT * FROM user")
  fun users(): DataSource.Factory<Int, User>
}
```

Paging ❤ Room

```
@Dao
interface UserDao {

  @Query("SELECT * FROM user")
  fun users(): DataSource.Factory<Int, User>
}
```


SO?
@askashdavies

Remote Data Source

Backend ☁️

Remote Data Source

Index 

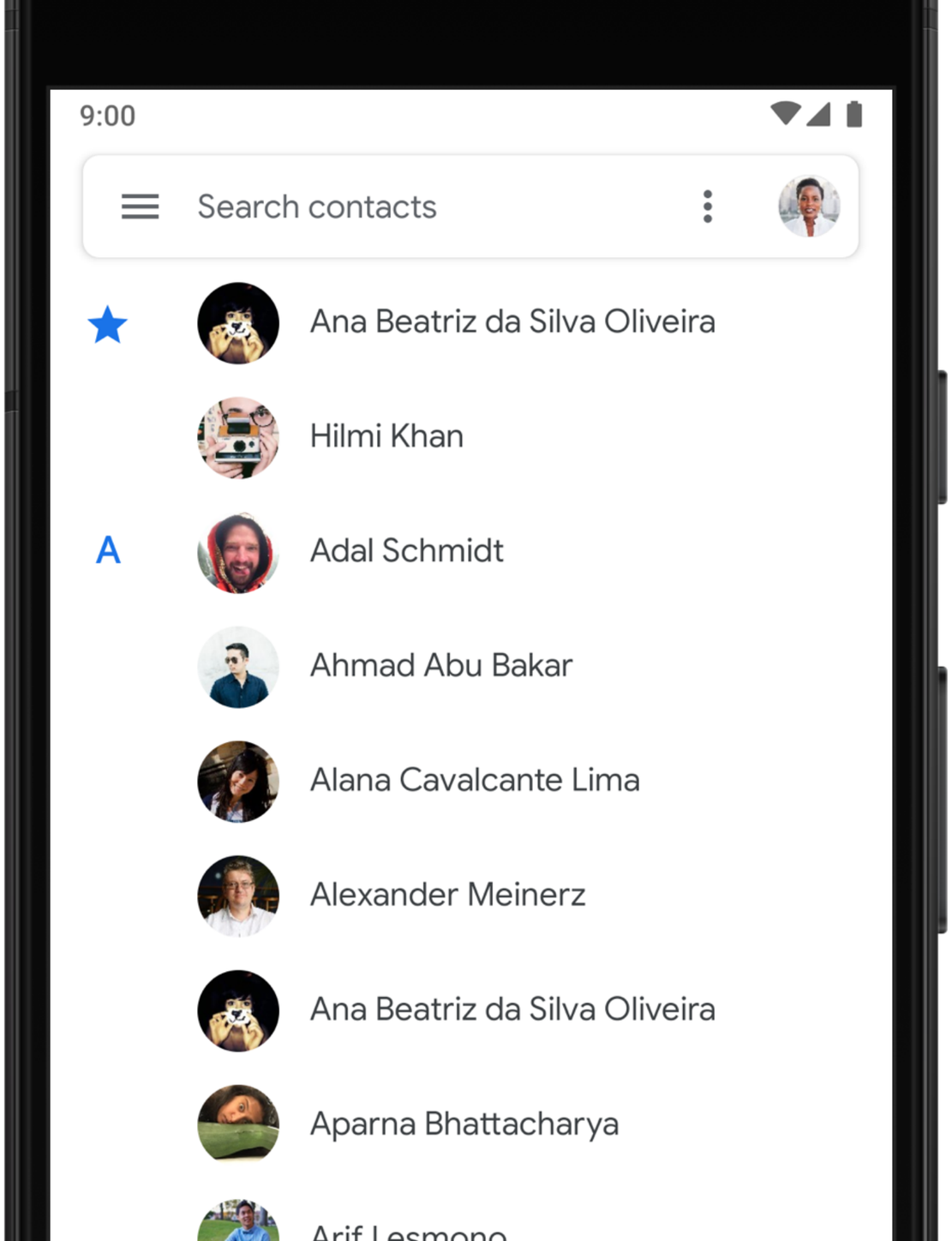
DataSource<K, V>

- PositionalDataSource 🏎
- ItemKeyedDataSource 🔑
- PageKeyedDataSource 🔢

PositionalDataSource

PositionalDataSource<User>

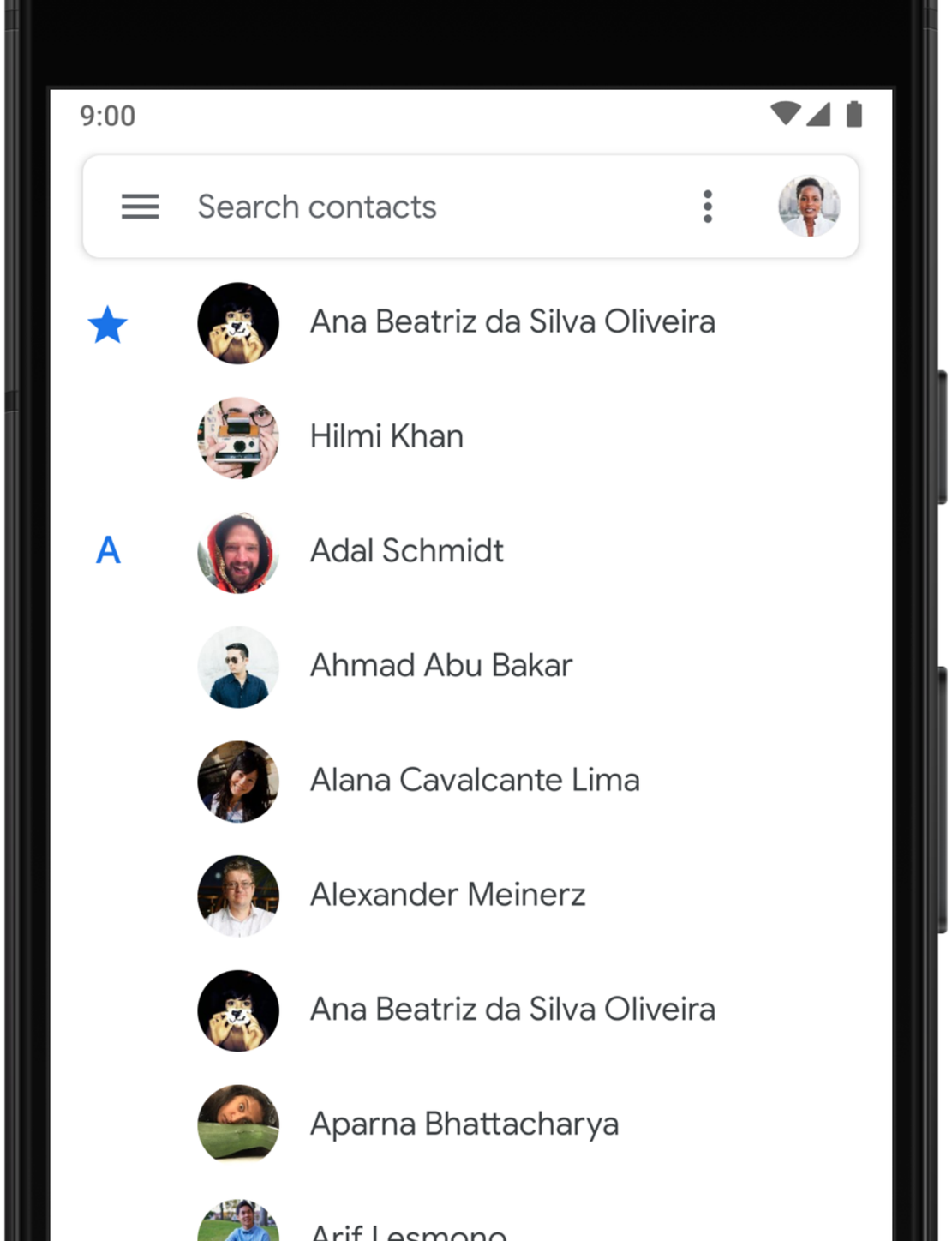
- Able to scroll to different elements
- Load pages of requested sizes
- Load pages at arbitrary positions
- Assumed ordering by integer index
- Provide a fixed item count



PositionalDataSource

PositionalDataSource<User>

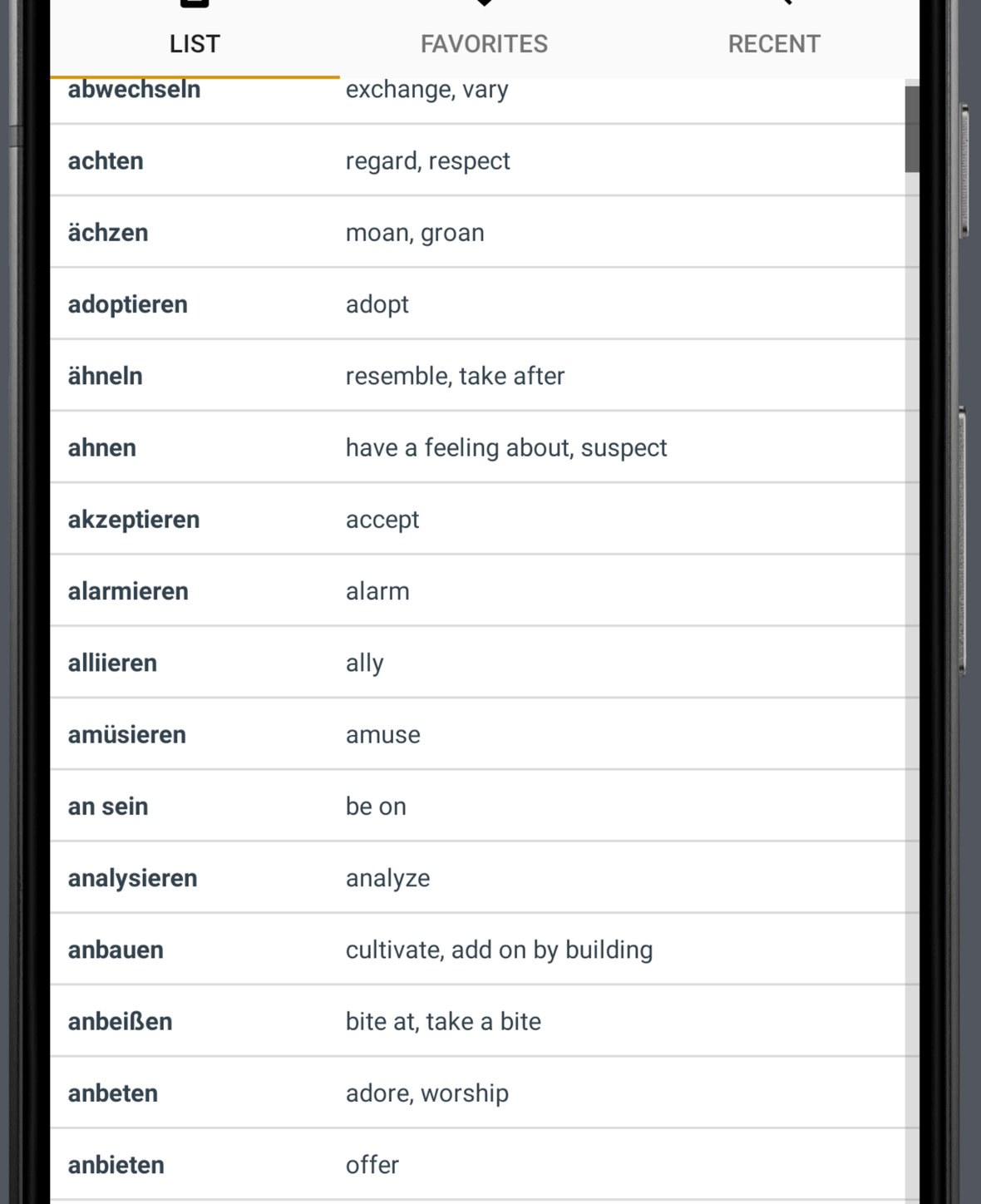
- loadInitial()
  - requestedStartPosition
  - requestedLoadSize
  - pageSize
  - placeholdersEnabled
- loadRange()
  - startPosition
  - loadSize



ItemKeyedDataSource

`ItemKeyedDataSource<String, User>`

- Great for ordered data sets
- Items can be uniquely identified
- Item key indicates position
- Detect items before or after

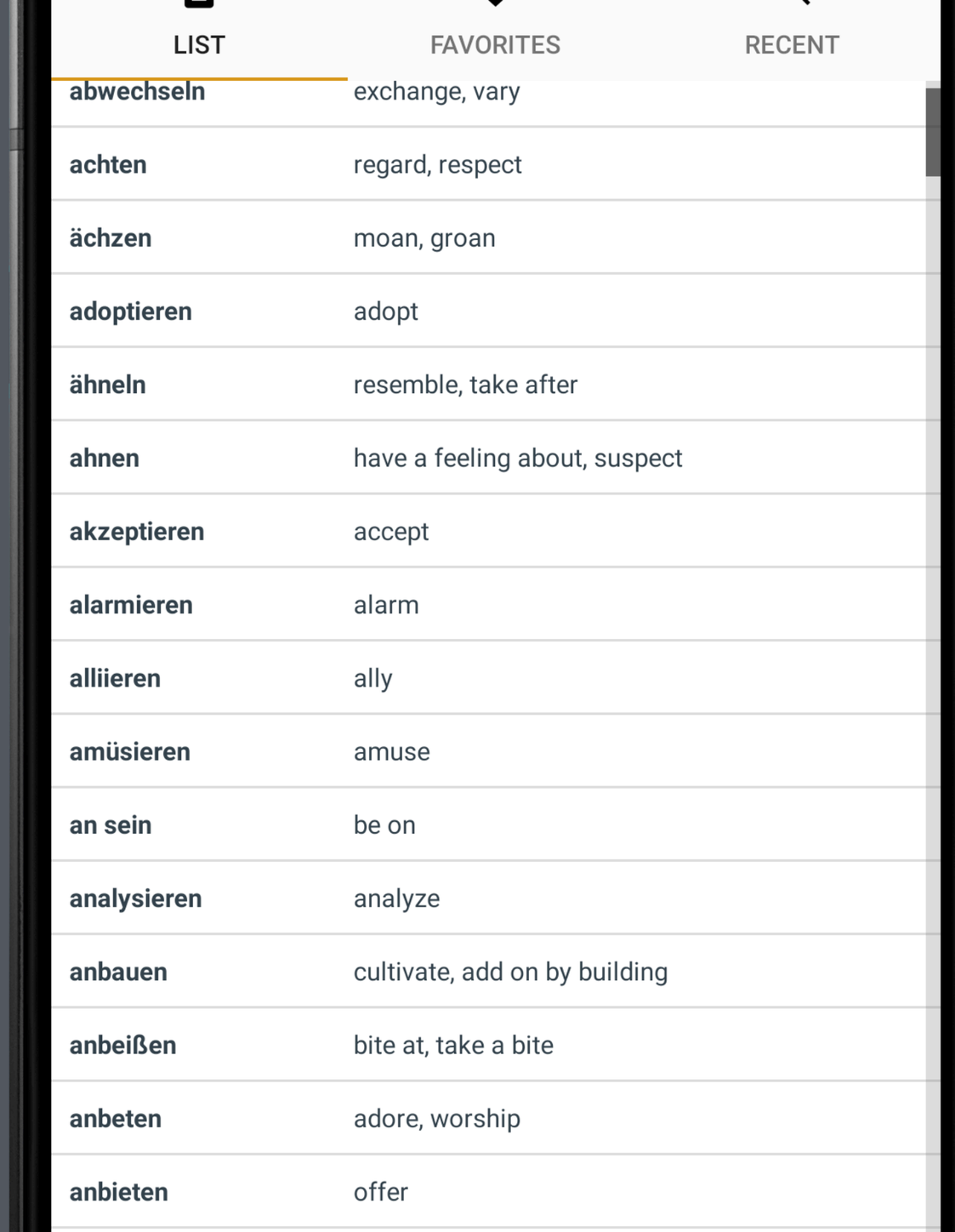


ItemKeyedDataSource

`ItemKeyedDataSource<String, User>`

- `getKey()`
- `loadInitial()`
  - `requestedInitialKey`
  - `requestedLoadSize`
  - `placeholdersEnabled`
- `loadAfter()`
  - `key`
  - `requestedLoadSize`
- `loadBefore()`
  - `key`
  - `requestedLoadSize`

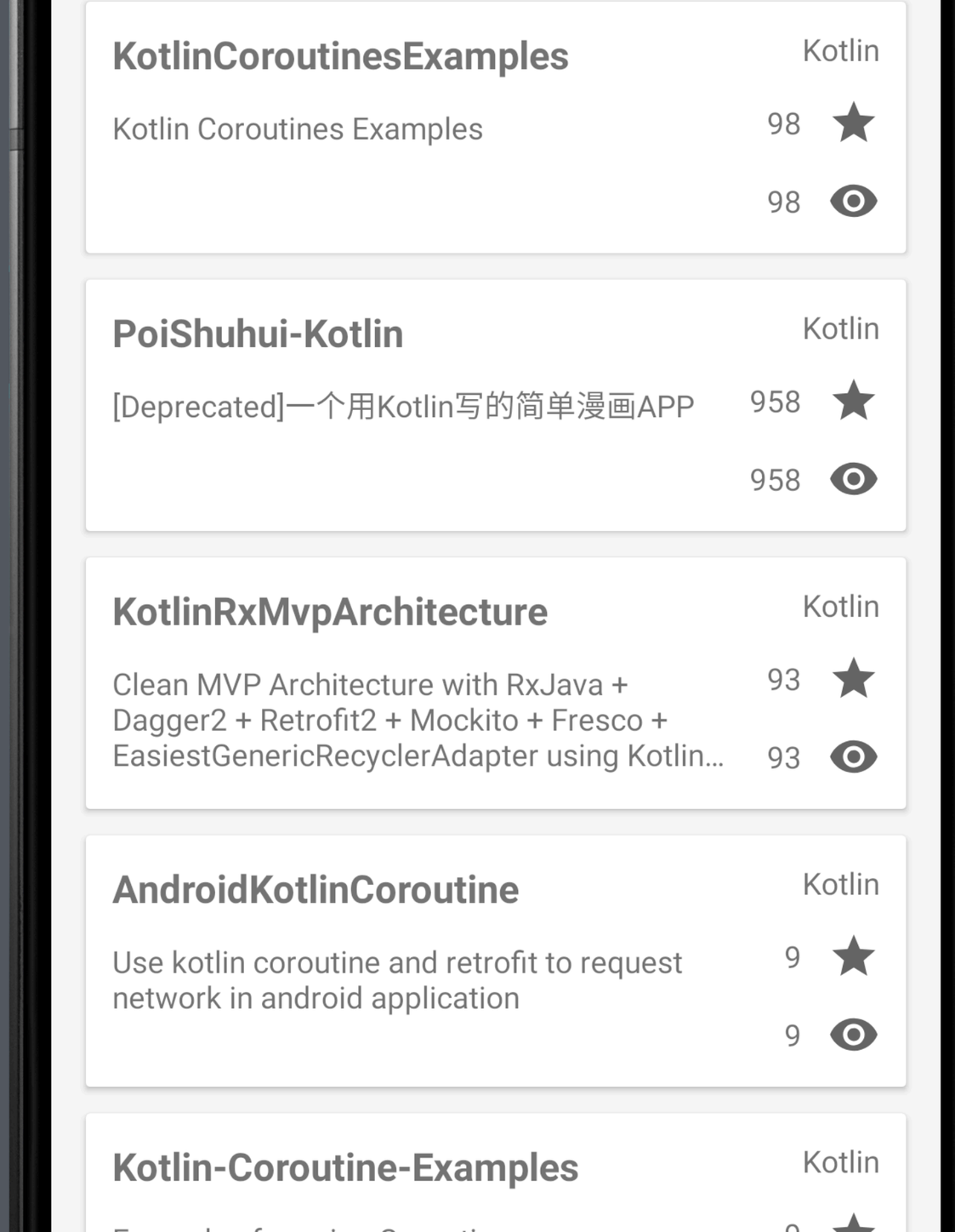
@askashdavies



PageKeyedDataSource

`PageKeyedDataSource<String, User>`

- Common for API responses
  - GitHub
  - Twitter
  - Reddit

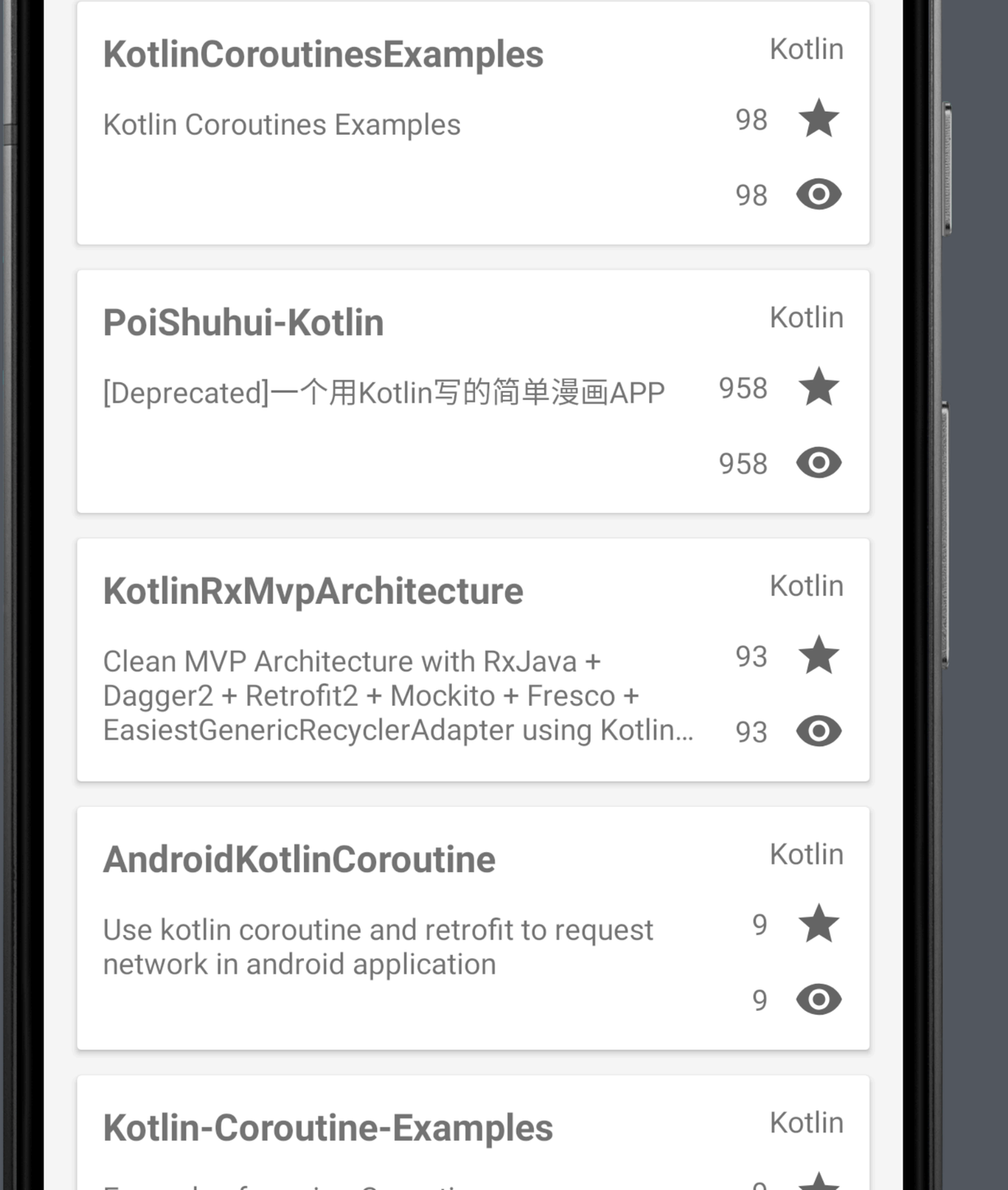


PageKeyedDataSource

`PageKeyedDataSource<String, User>`

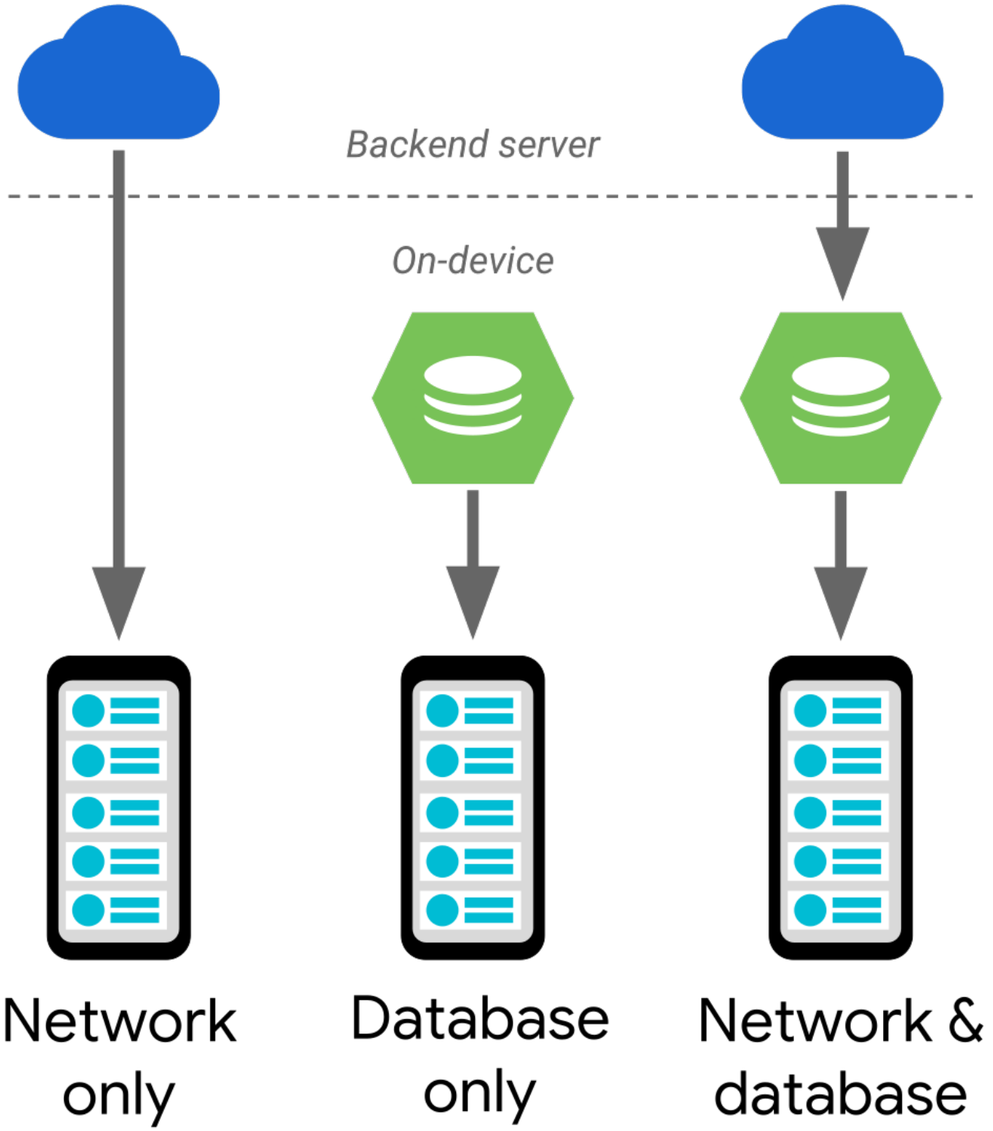
- `loadInitial()`
  - `requestedLoadSize`
  - `placeholdersEnabled`
- `loadAfter()`
  - `key`
  - `requestedLoadSize`
- `loadBefore()`
  - `key`
  - `requestedLoadSize`

@askashdavies




@askashdavies

Architecture 🏢


Backend server
On-device
Network only
Database only
Network & database

Database 

Source of truth

- Consistent data presentation
- Simple process - need more, load more
- Gracefully degrades on failure
- Optionally refresh on observe

`BoundaryCallback` 🏁

- Signals end of data from database
- Triggers network load to populate
- Provided to `PagedListBuilder`

BoundaryCallback 🏁

PagedList.BoundaryCallback<User>

```
public abstract static class BoundaryCallback<T> {

  public void onZeroItemsLoaded() {
    /* ... */
  }

  public void onItemAtFrontLoaded(@NonNull T itemAtFront) {
    /* ... */
  }

  public void onItemAtEndLoaded(@NonNull T itemAtEnd) {
    /* ... */
  }
}
```

@askashdavies

BoundaryCallback 🏁

```
class UserBoundaryCallback(
    private val service: UserService,
    private val dao: UserDao,
    private val query: String
) : PagedList.BoundaryCallback<User>() {

  private var page: Int = 0

  override fun onZeroItemsLoaded() {
    requestItems()
  }

  override fun onItemAtEndLoaded(itemAtEnd: User) {
    requestItems()
  }

  private fun requestItems() {
    GlobalScope.launch { // Ignore structured concurrency
      dao.insert(service.users(query, page, 50))
      page++
    }
  }
}
```

@askashdavies

BoundaryCallback 🏁

```
class UserBoundaryCallback(
    private val service: UserService,
    private val dao: UserDao,
    private val query: String
) : PagedList.BoundaryCallback<User>() {

  private var page: Int = 0

  override fun onZeroItemsLoaded() {
    requestItems()
  }

  override fun onItemAtEndLoaded(itemAtEnd: User) {
    requestItems()
  }

  private fun requestItems() {
    GlobalScope.launch { // Ignore structured concurrency
      dao.insert(service.users(query, page, 50))
      page++
    }
  }
}
```

@askashdavies

BoundaryCallback 🏁

```
class UserRepository(
    private val service: UserService
) {

  fun users(query: String): LiveData<PagedList<Repo>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .setInitialLoadSizeHint(50)
        .setPrefetchDistance(10)
        .setEnablePlaceholders(false)
        .build()

    return LivePagedListBuilder(factory, config)
      .build()
  }
}
```

BoundaryCallback 🏁

```
class UserRepository(
    private val service: UserService
) {

  fun users(query: String): LiveData<PagedList<Repo>> {
    val factory: DataSource.Factory = service.users()
    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(PAGE_SIZE)
        .setInitialLoadSizeHint(50)
        .setPrefetchDistance(10)
        .setEnablePlaceholders(false)
        .build()

    return LivePagedListBuilder(factory, config)
        .build()
  }
}
```

BoundaryCallback 🏁

```
class UserRepository(
    private val service: UserService,
    private val dao: UserDao
) {

  fun repos(query: String): LiveData<PagedList<Repo>> {
    val factory: DataSource.Factory<Int, Repo> = dao.repos(query)

    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(20)
        .setEnablePlaceholders(true)
        .setPrefetchDistance(50)
        .build()

    return LivePagedListBuilder(factory, config)
        .build()
  }
}
```

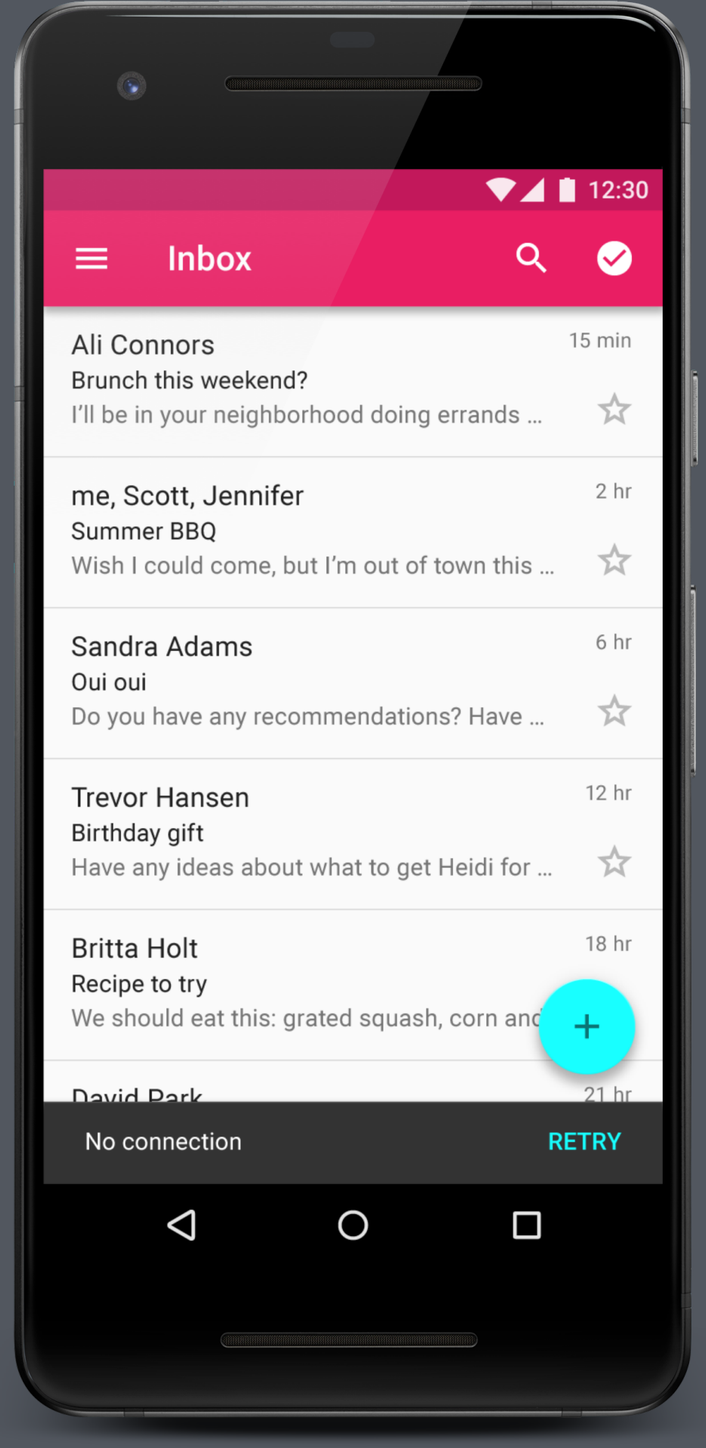
@askashdavies

BoundaryCallback 🏁

```
class UserRepository(
    private val service: UserService,
    private val dao: UserDao
) {

  fun repos(query: String): LiveData<PagedList<Repo>> {
    val factory: DataSource.Factory<Int, Repo> = dao.repos(query)
    val callback = RepoBoundaryCallback(service, dao, query)

    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(20)
        .setEnablePlaceholders(true)
        .setPrefetchDistance(50)
        .build()

    return LivePagedListBuilder(factory, config)
        .setBoundaryCallback(callback)
        .build()
  }
}
```

Error Handling 

Error Handling 💥



@askashdavies

Error Handling 💥

```
class UserBoundaryCallback : PagedList.BoundaryCallback<Repo>() {

  // LiveData of network errors.
  private val _errors = MutableLiveData<String>()
  val errors: LiveData<String> get() = _errors

   /*
    * ...
    * */
}
```

Error Handling 💥

```
class UserRepository(
    private val service: UserService,
    private val dao: UserDao
) {

  fun repos(query: String): LiveData<PagedList<Repo>> {
    val factory: DataSource.Factory<Int, Repo> = dao.repos(query)
    val callback = RepoBoundaryCallback(service, dao, query)

    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(20)
        .setEnablePlaceholders(true)
        .setPrefetchDistance(50)
        .build()

    return LivePagedListBuilder(factory, config)
        .setBoundaryCallback(callback)
        .build()
  }
}
```

@askashdavies

Error Handling 💥

```
class UserRepository(
    private val service: UserService,
    private val dao: UserDao
) {

  fun repos(query: String): Pair<LiveData<PagedList<User>>, LiveData<Throwable>> {
    val factory: DataSource.Factory<Int, User> = dao.repos(query)
    val callback = UserBoundaryCallback(service, dao, query)

    val config: PagedList.Config = PagedList.Config.Builder()
        .setPageSize(20)
        .setEnablePlaceholders(true)
        .setPrefetchDistance(50)
        .build()

    val data: LiveData<PagedList<User>> = LivePagedListBuilder(factory, config)
        .setBoundaryCallback(callback)
        .build()

    return data to callback.errors
  }
}
```

@askashdavies

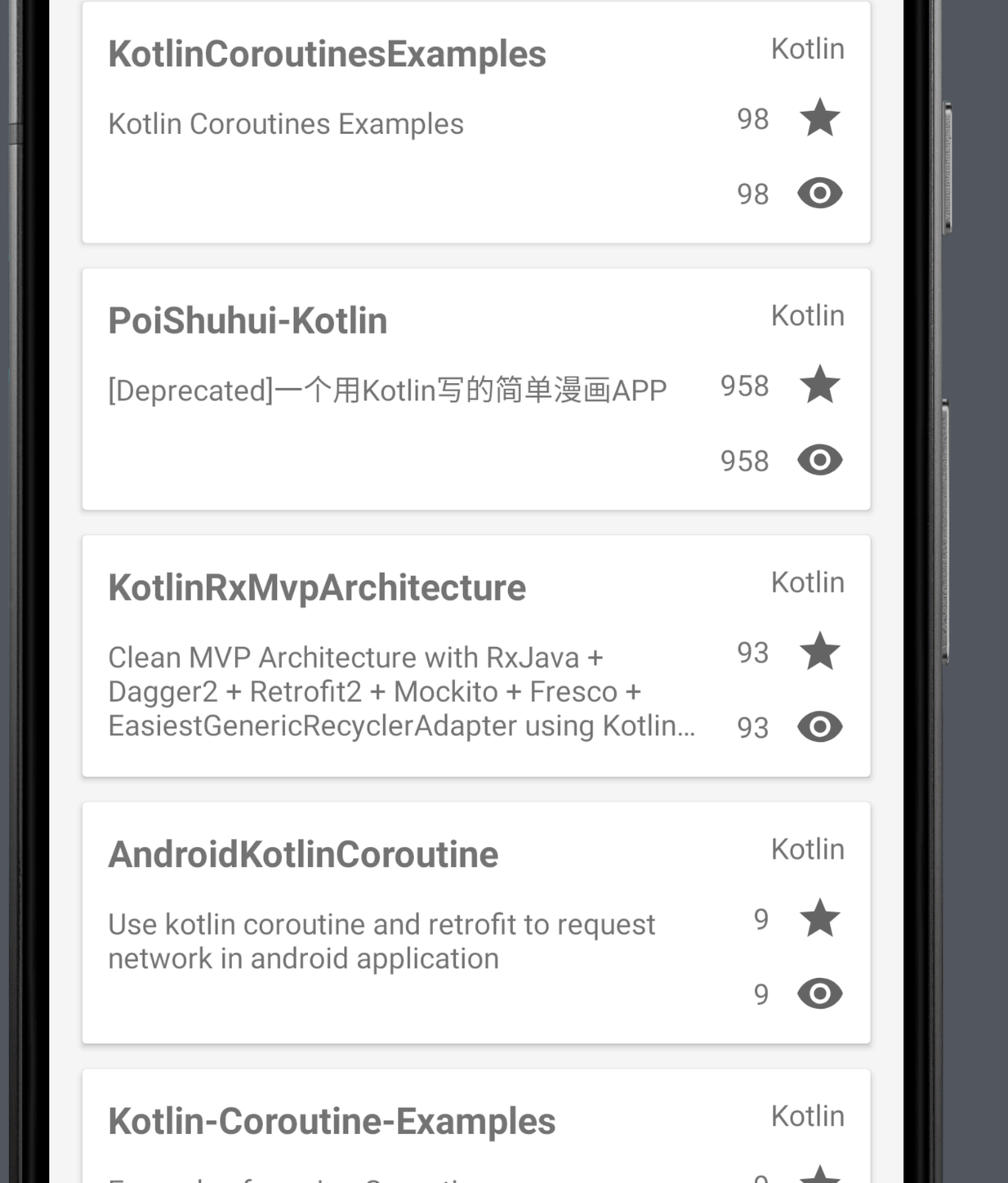
Further Reading 📖

- **Florina Muntenescu: Migrating to Paging Library**
  youtube.com/watch?v=8DPgwrV_9-g
- **Chris Craik & Yigit Boyar: Manage infinite lists with RecyclerView and Paging**
  youtube.com/watch?v=BE5bsyGGLf4
- **ADB: Prefetch and Paging**
  androidbackstage.blogspot.com/2018/10/episode-101-prefetch-and-paging.html
- **Android Paging Codelab**
  codelabs.developers.google.com/codelabs/android-paging/
- **Google Samples: Paging with Network Sample**
  github.com/googlesamples/android-architecture-components/tree/master/PagingWithNetworkSample
- **Chris Banes: FlowPagedListBuilder**
  github.com/chrisbanes/tivi/blob/master/data-android/src/main/java/app/tivi/data/FlowPagedListBuilder.kt

Implementing the Paging Library

bit.ly/github-repo-search

bit.ly/paging-library



@askashdavies

Happy Paging!

@askashdavies