

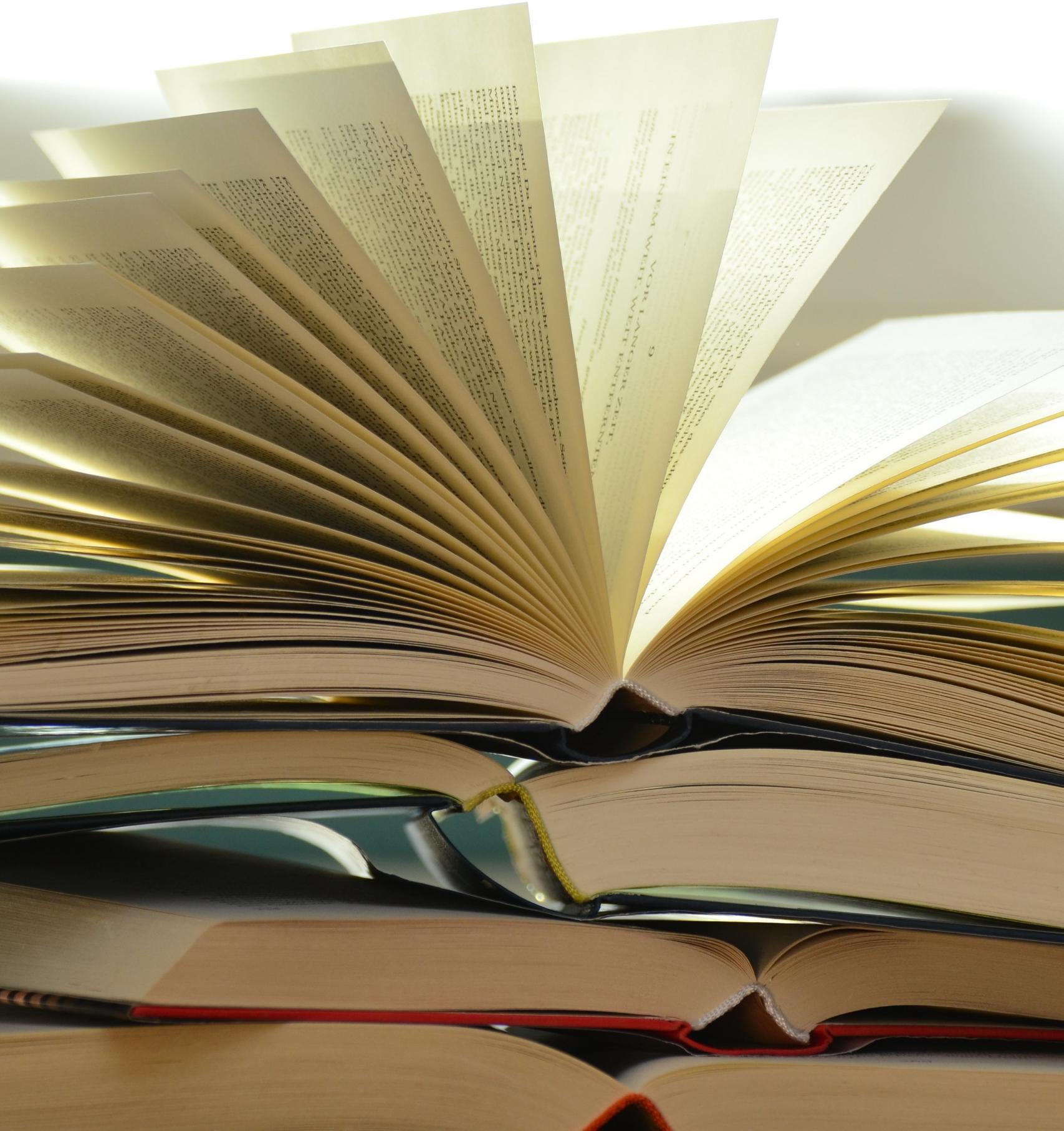
IMMOBILIEN
SCOUT24

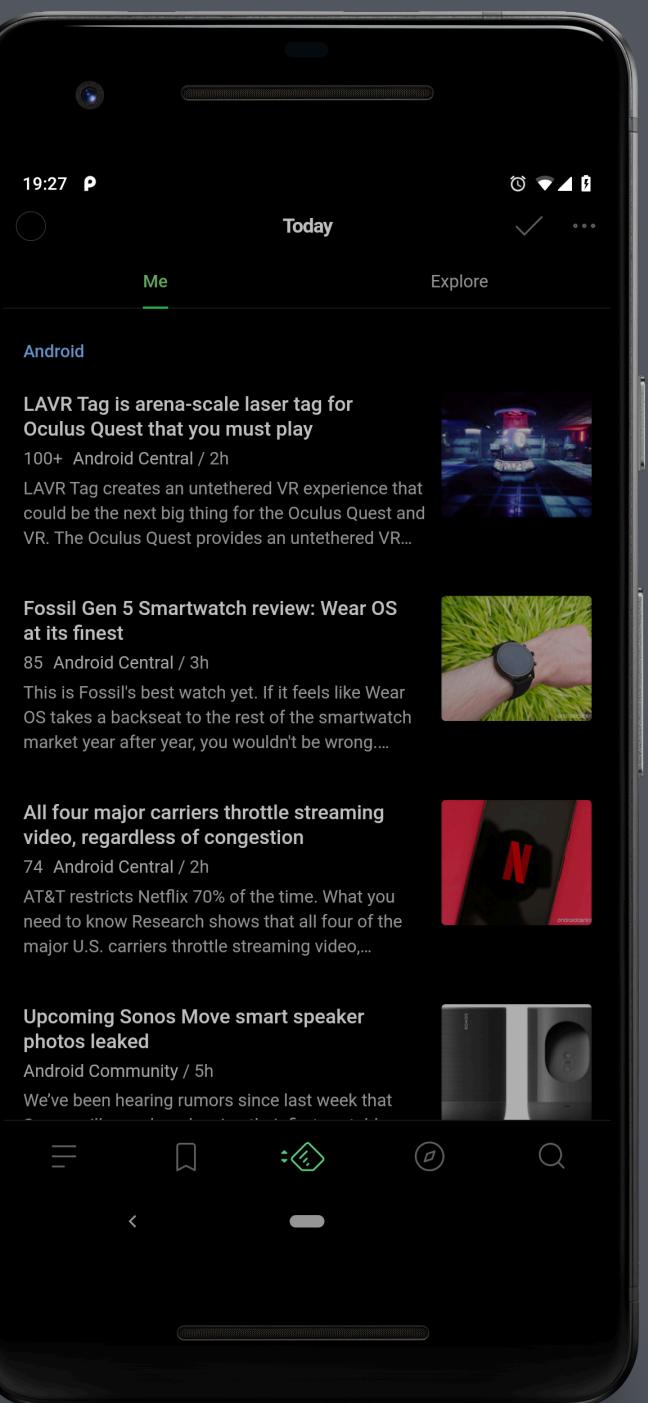
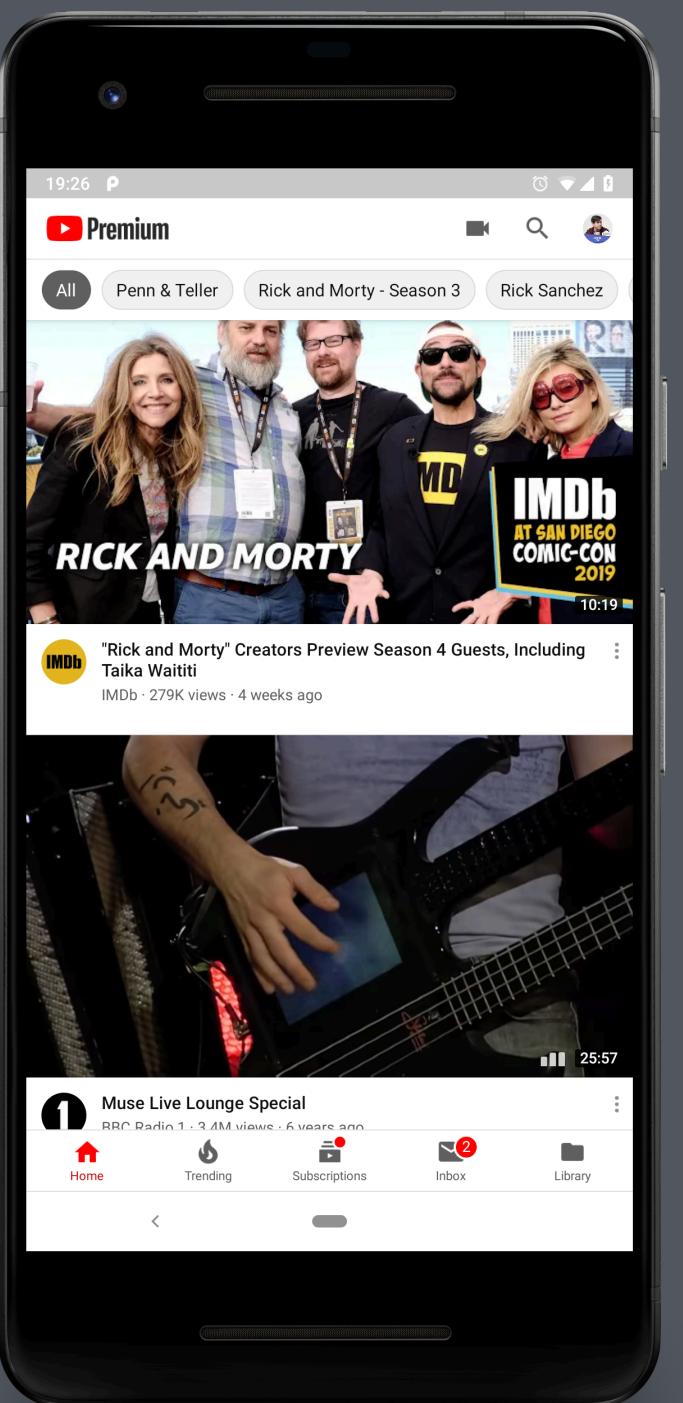
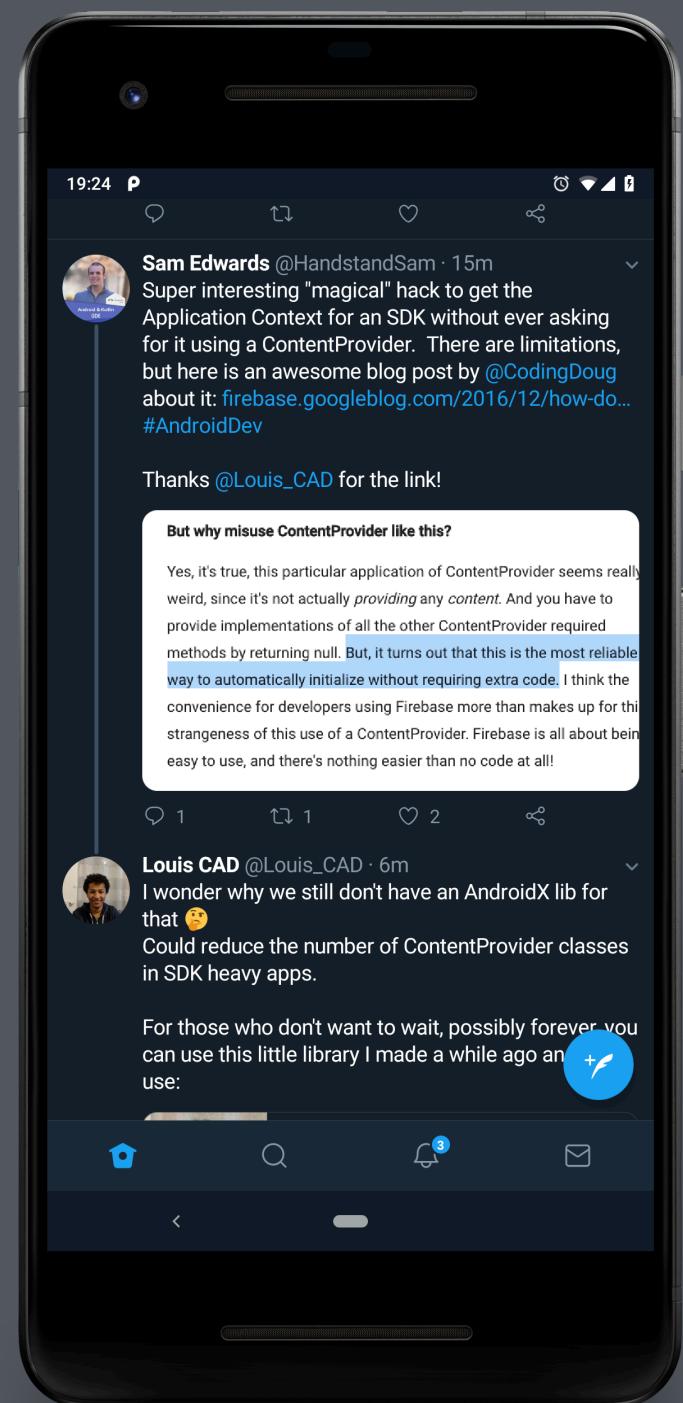
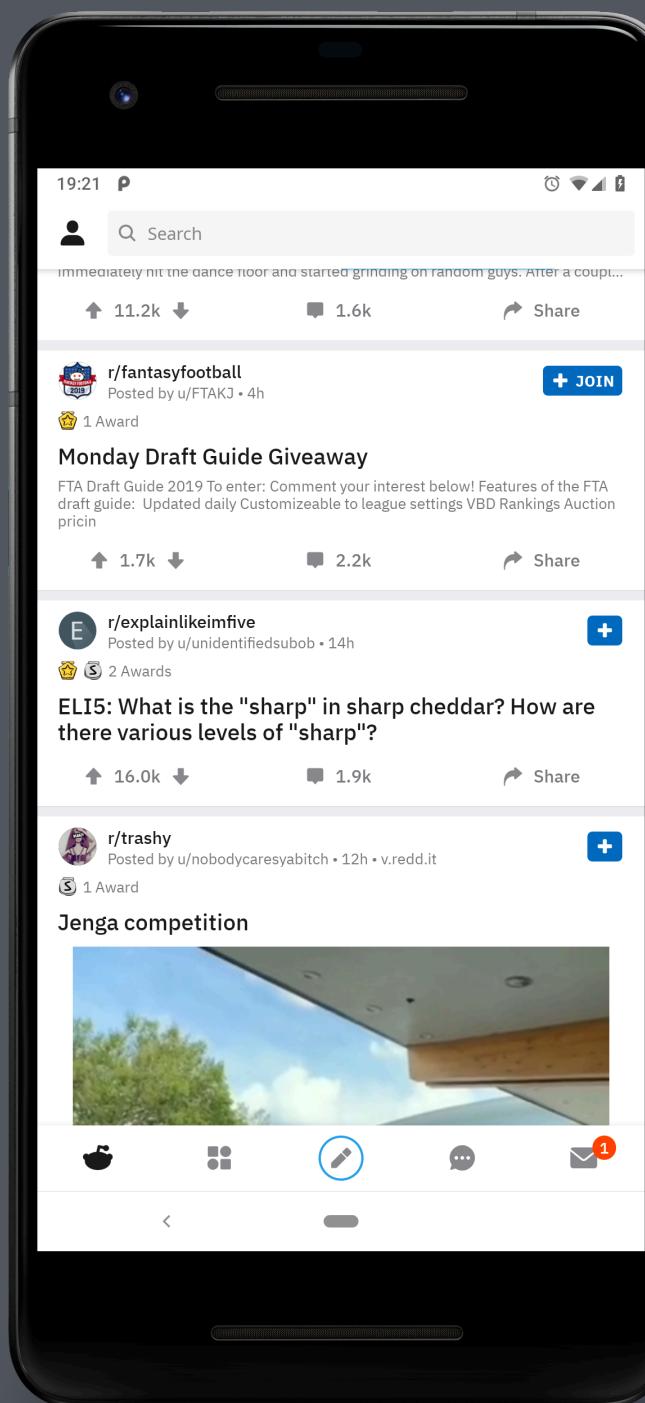
Implementing the Paging Library

Droidcon NYC 🇺🇸

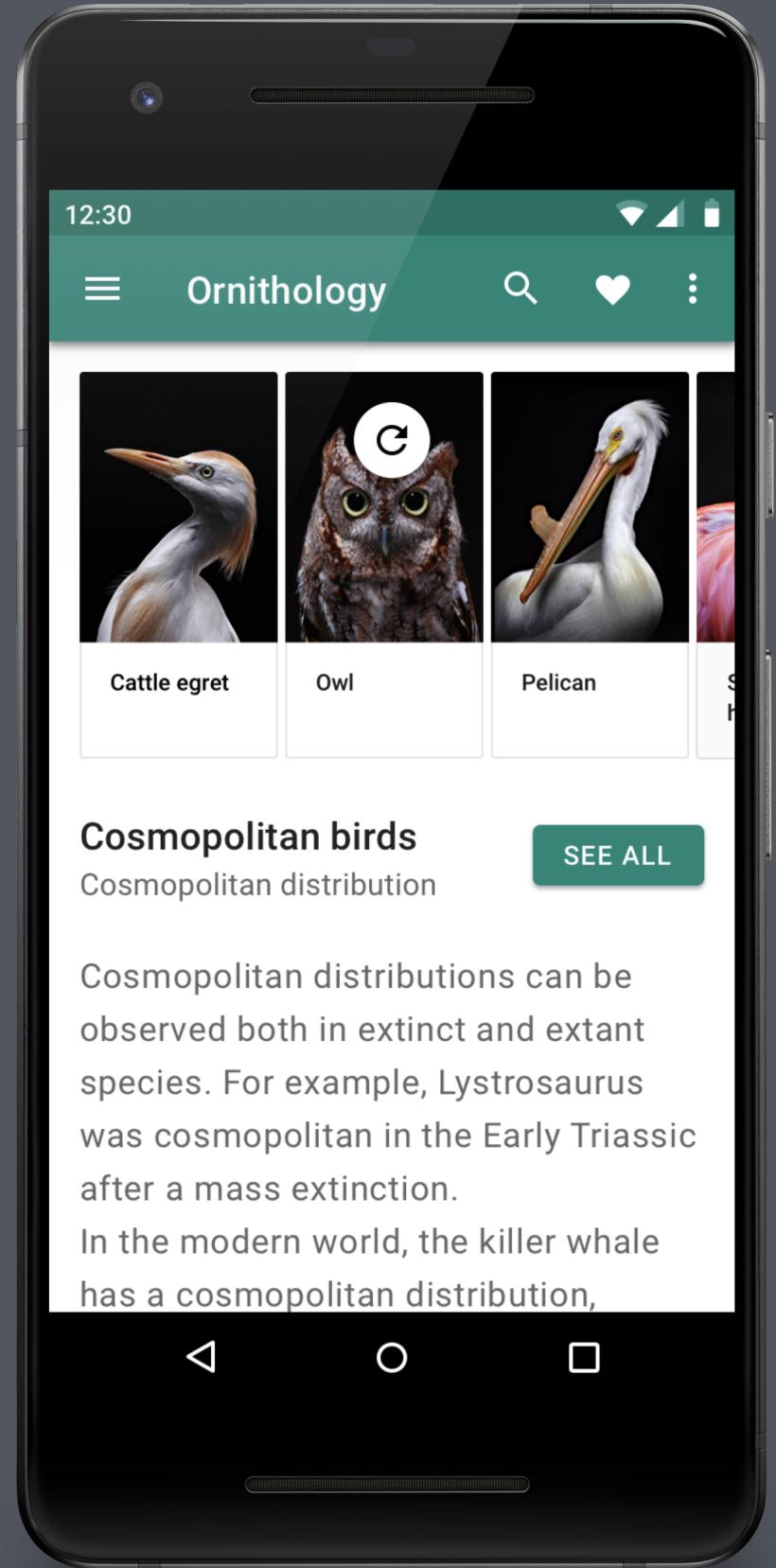


@askashdavies



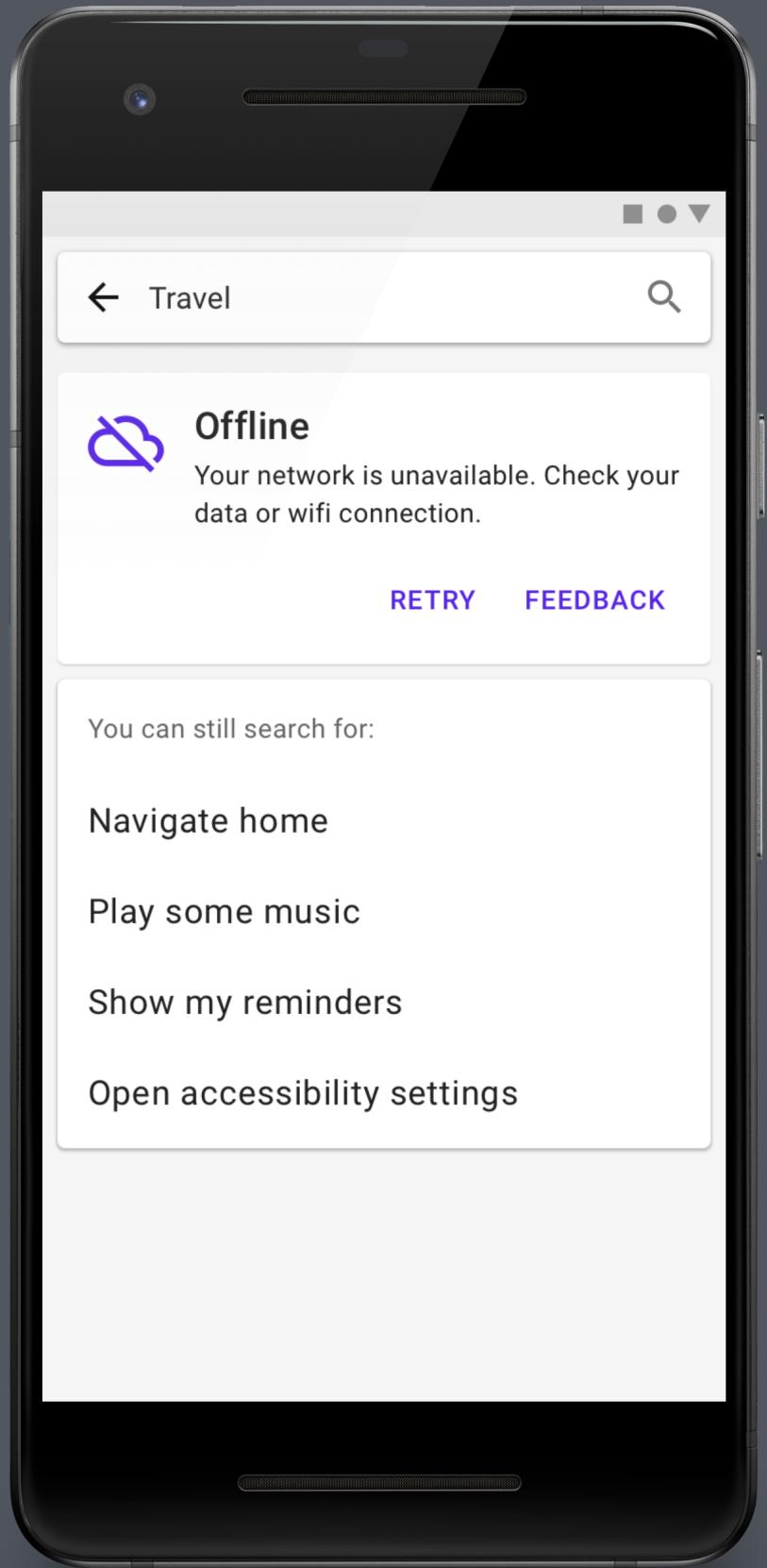


Up-To-Date

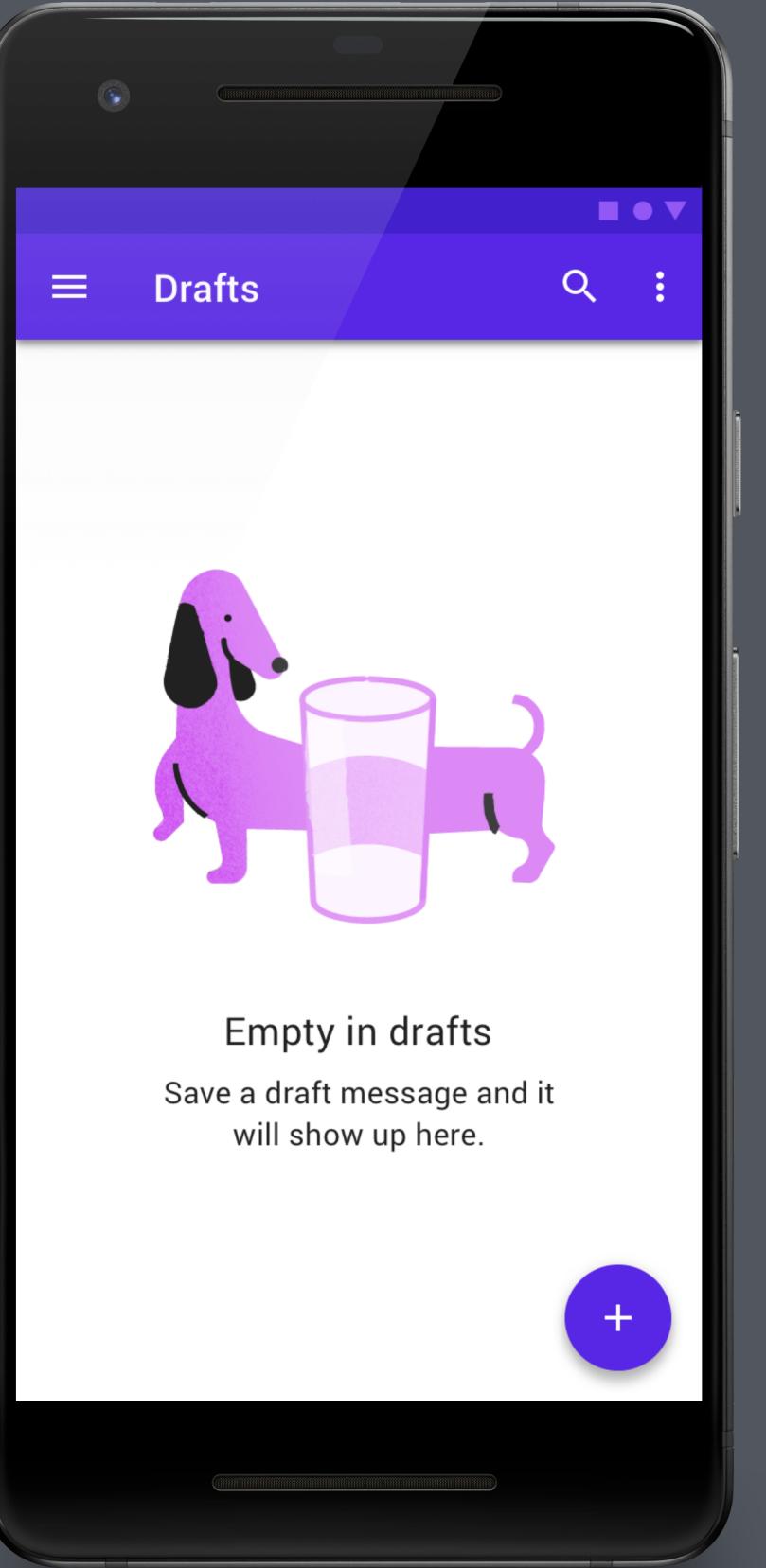


Large Data-Sets

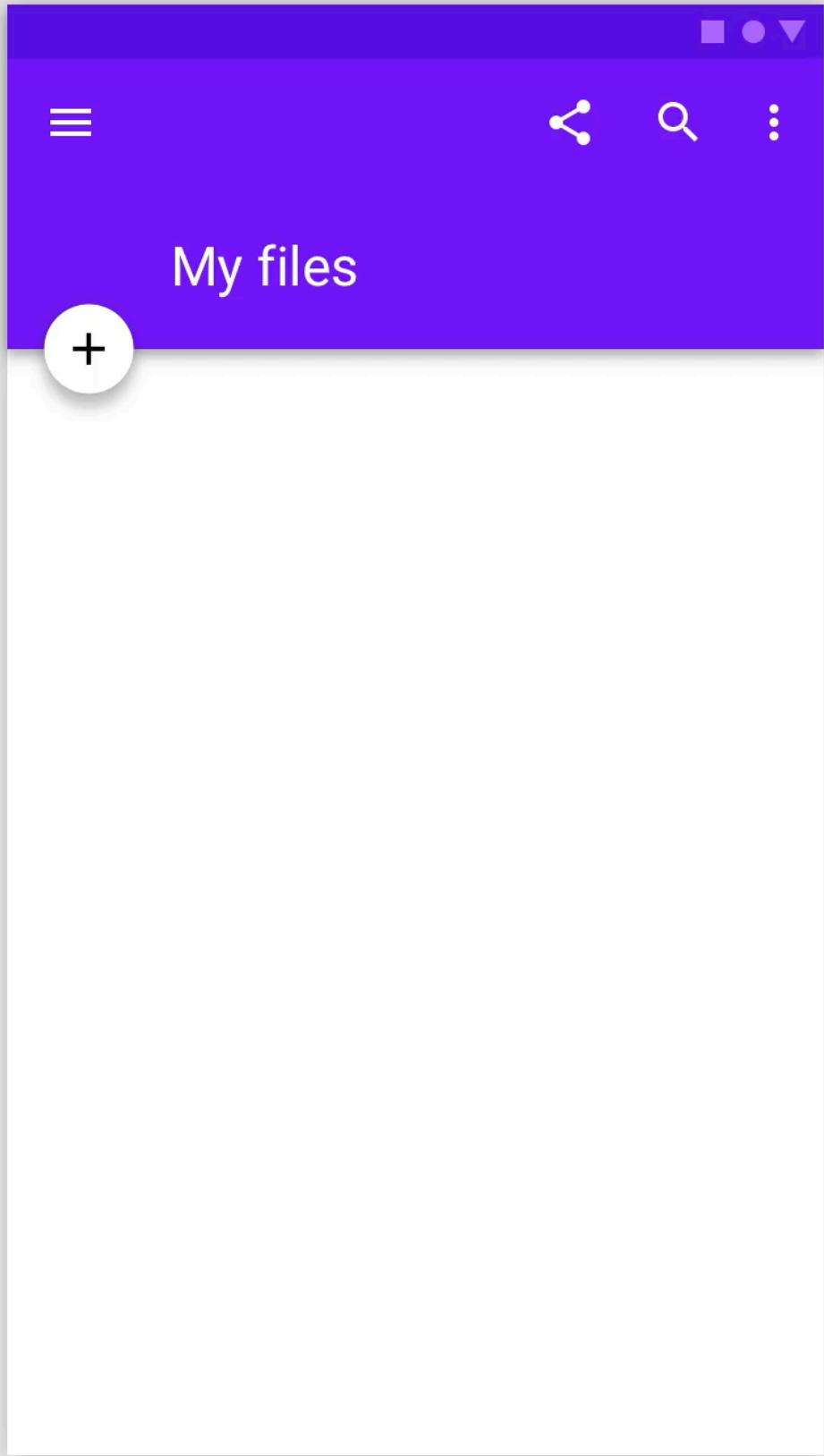
Offline



State



Progress





@askashdavies

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,
    array0f("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) {
        /* ... */
    }
}
```

Paging



Paging

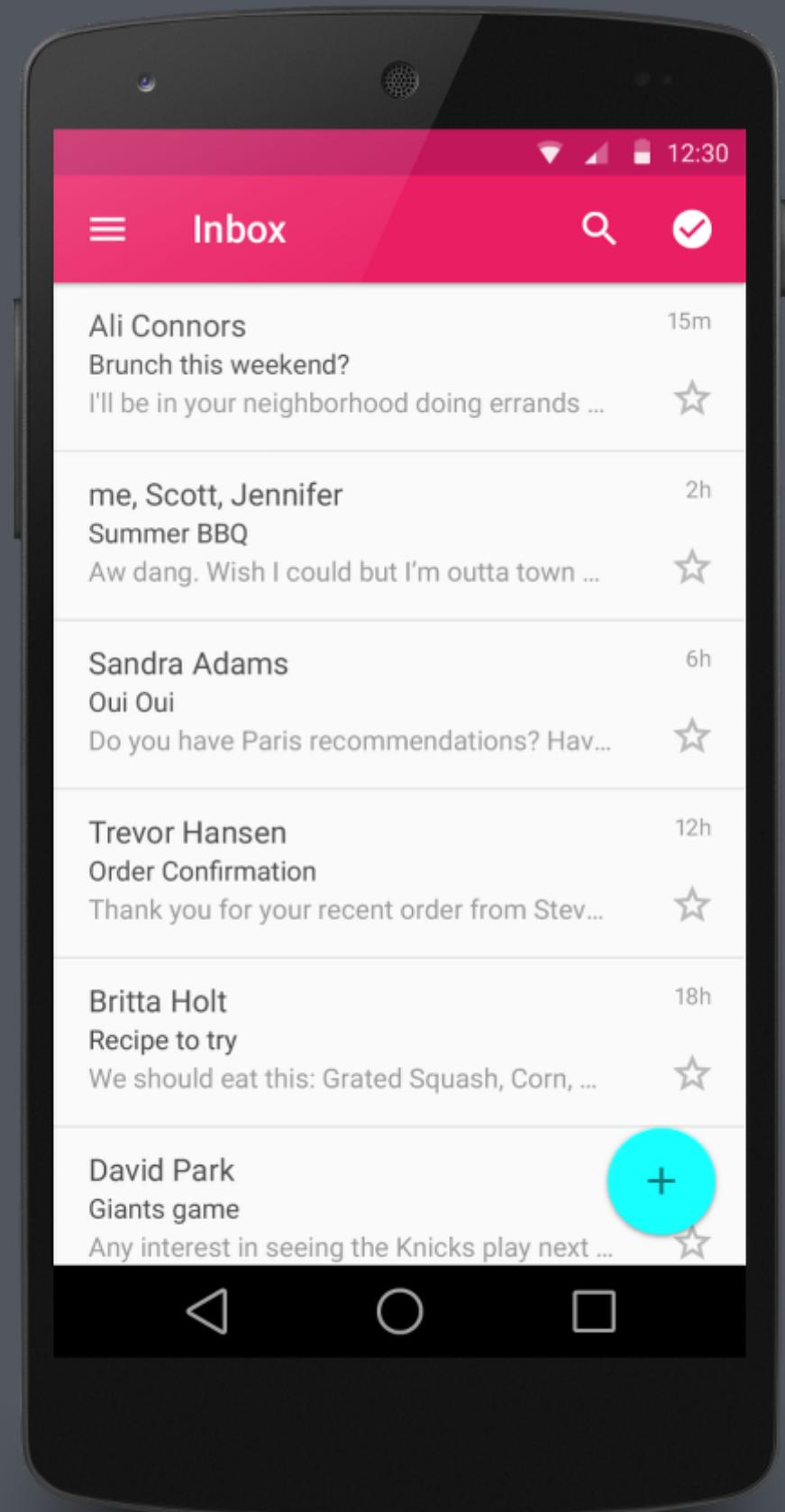


OnScrollListener

BaseAdapter / ListView

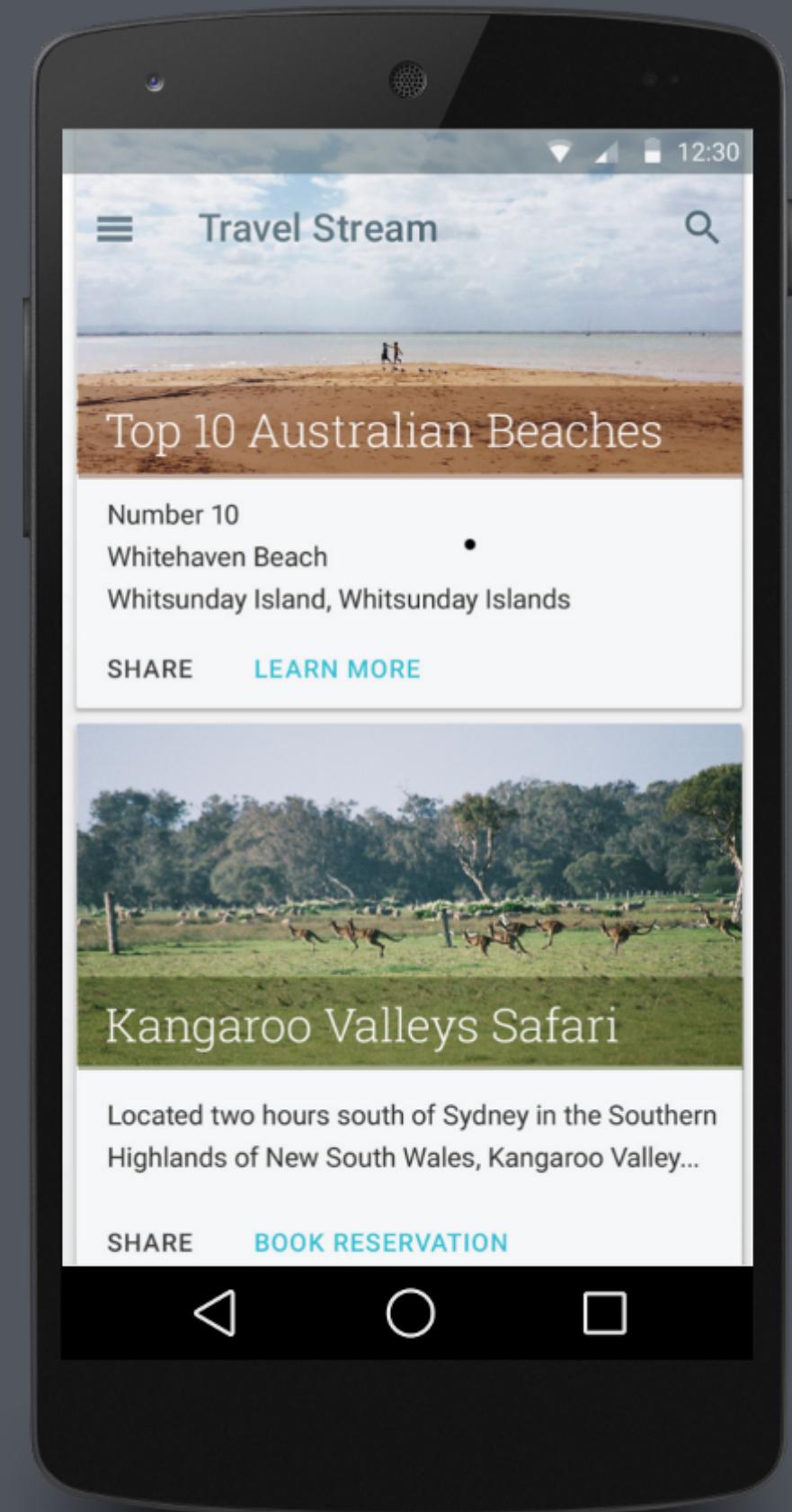
- Manages list of it's own data
- Manages view inflation and configuration
- Notify entire data set of change
- Not capable of diffing items

RecyclerView



@askashdavies

RecyclerView



Paging



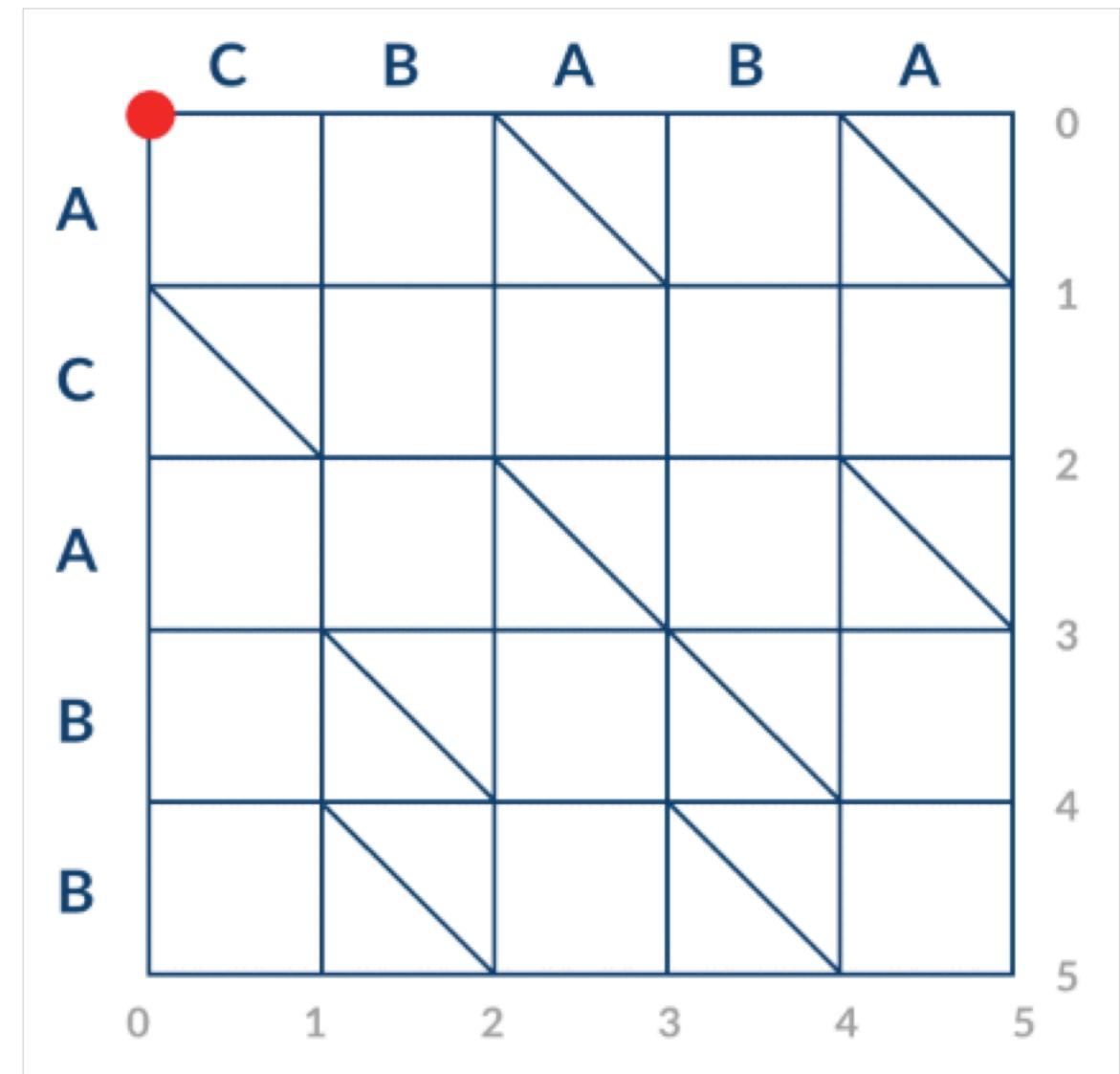
AsyncListUtil

Diffing

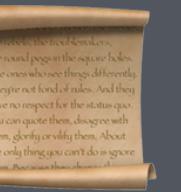
DiffUtil / AsyncListDiffer

DiffUtil

Myers Diff Algorithm

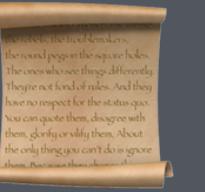


ListAdapter



The rebels, the free thinkers,
the round pegs in the square holes.
The ones who see things differently.
They're not fond of rules. And they
have no respect for the status quo.
You can quote them, disagree with
them, glorify or vilify them. About
the only thing you can't do is ignore
them. That's what makes them dangerous.

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) {
            /* ... */
        }
    }
}
```

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) {  
            /* ... */  
        }  
    }  
}
```

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) {  
            /* ... */  
        }  
    }  
}
```

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) {
            /* ... */
        }
    }
}
```

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) {
            /* ... */
        }
    }
}
```

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) {
            /* ... */
        }
    }
}
```

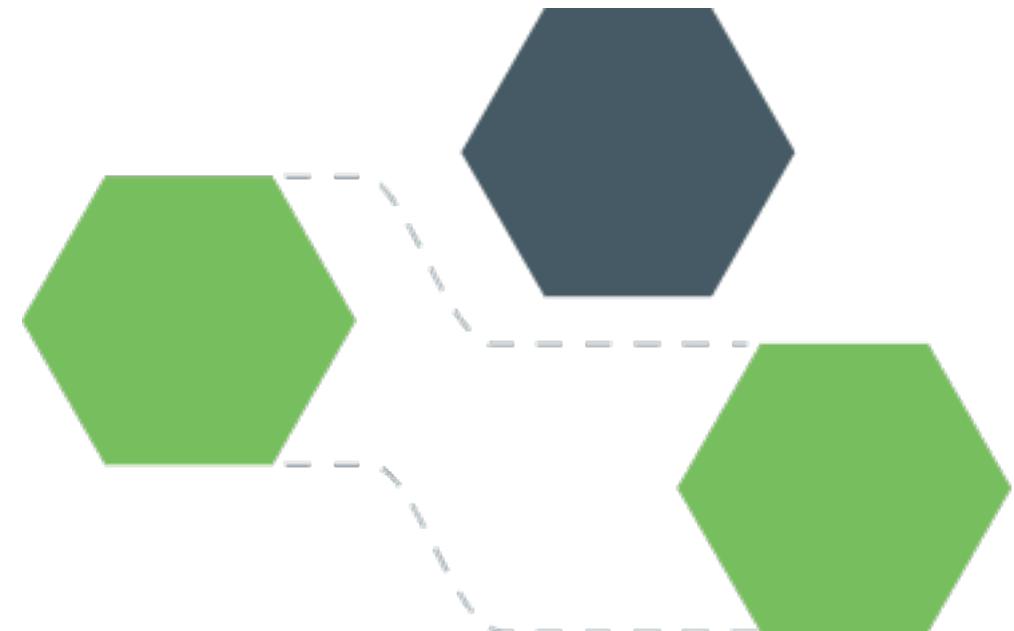
ListAdapter 💪



@askashdavies

Android JetPack

Foundation Components



Android JetPack

Architecture Components



Android JetPack

Behaviour Components



Android JetPack

UI Components



Android JetPack

Paging Library

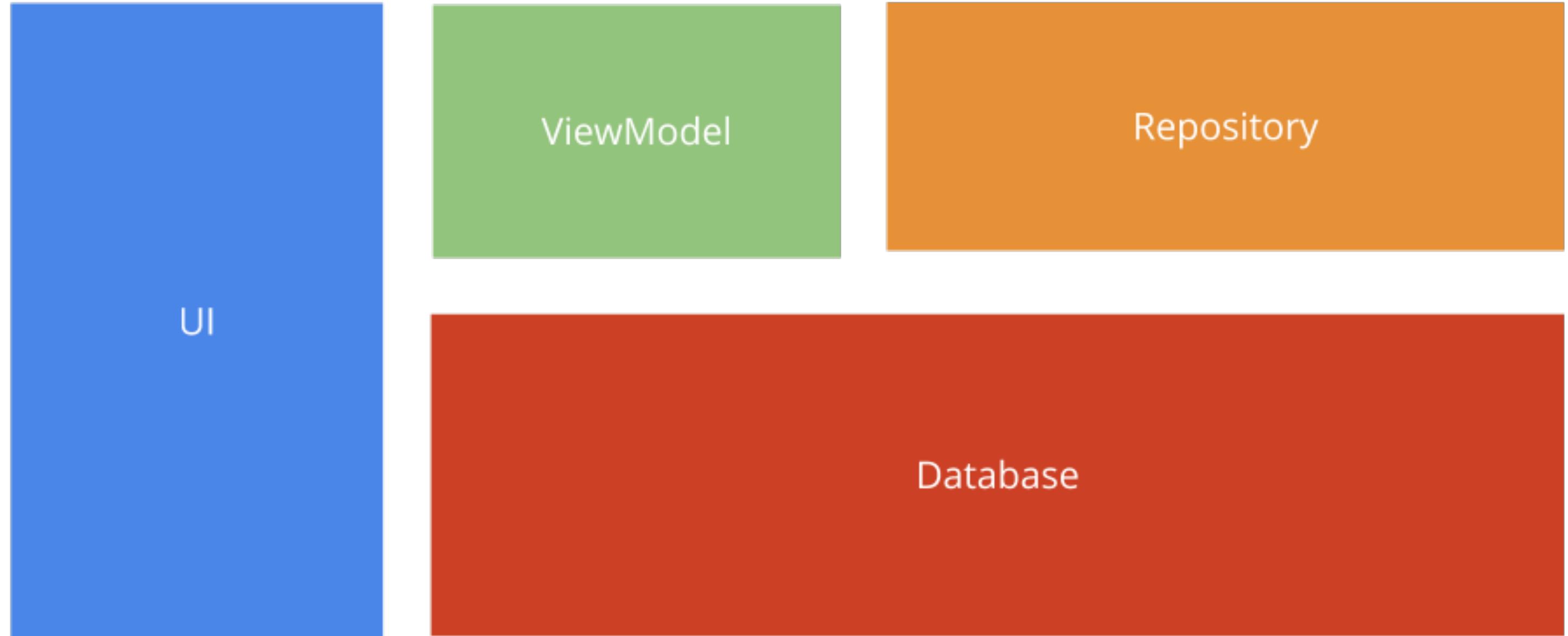


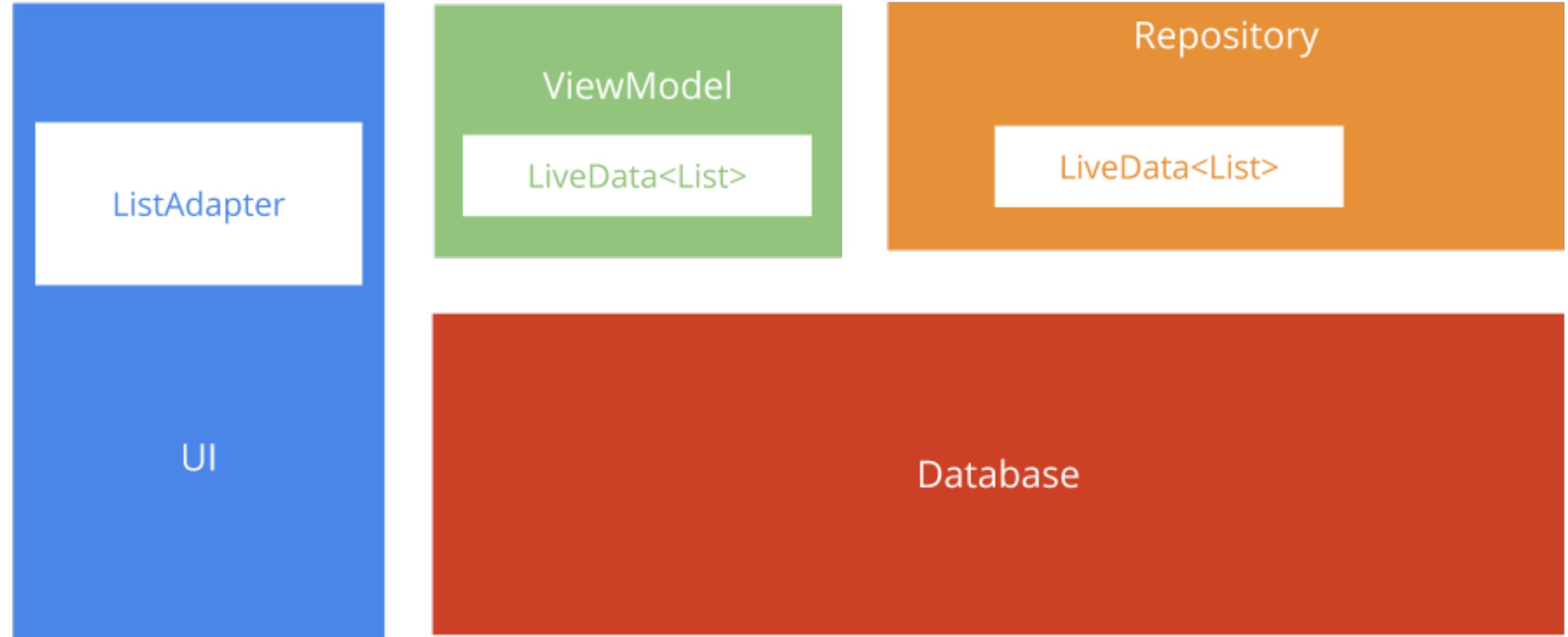
Android JetPack

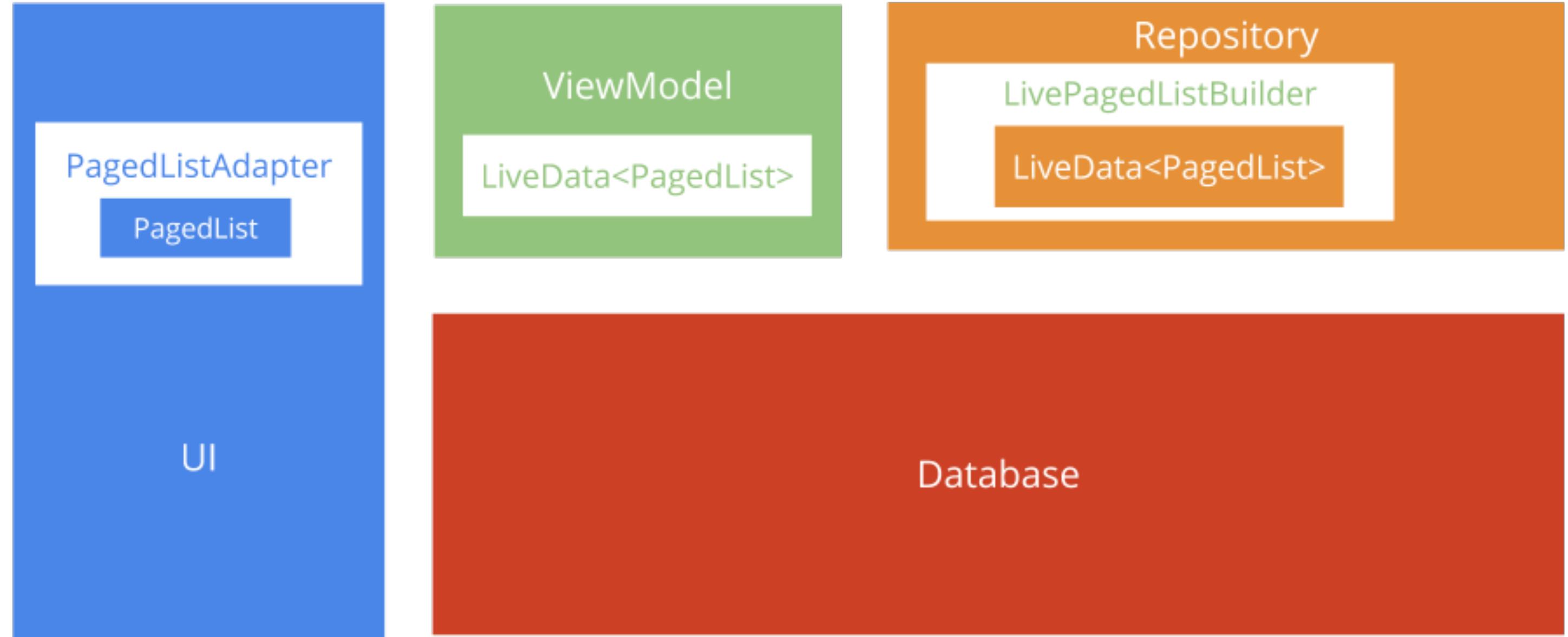


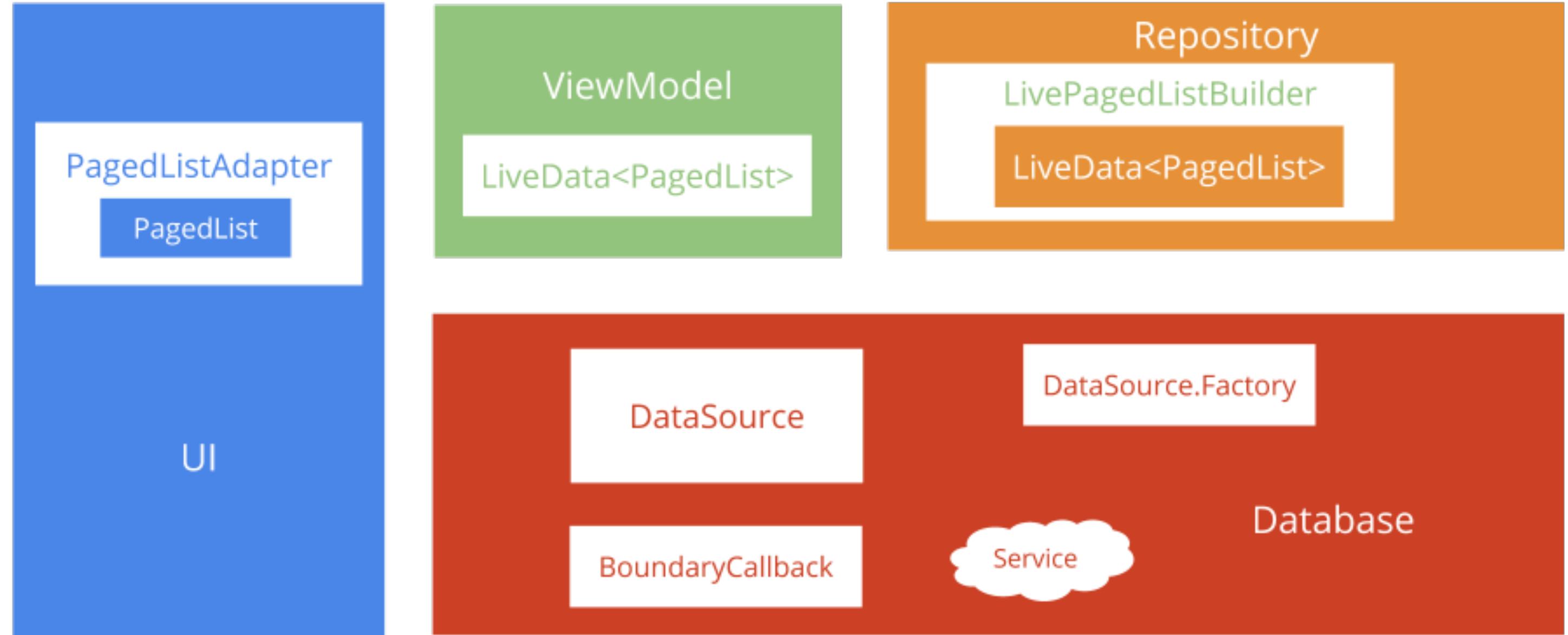
Paging Library

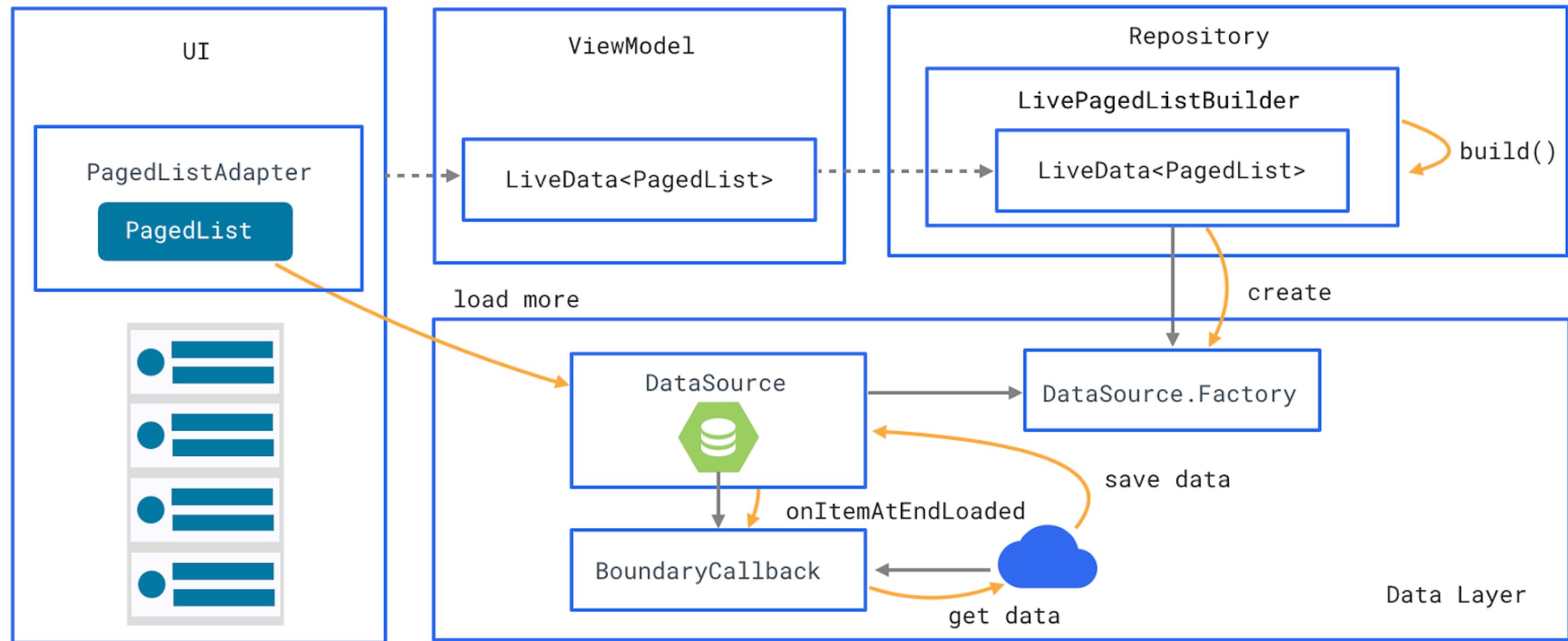
- PagedListAdapter
- PagedList
- DataSource / DataSource.Factory
- BoundaryCallback





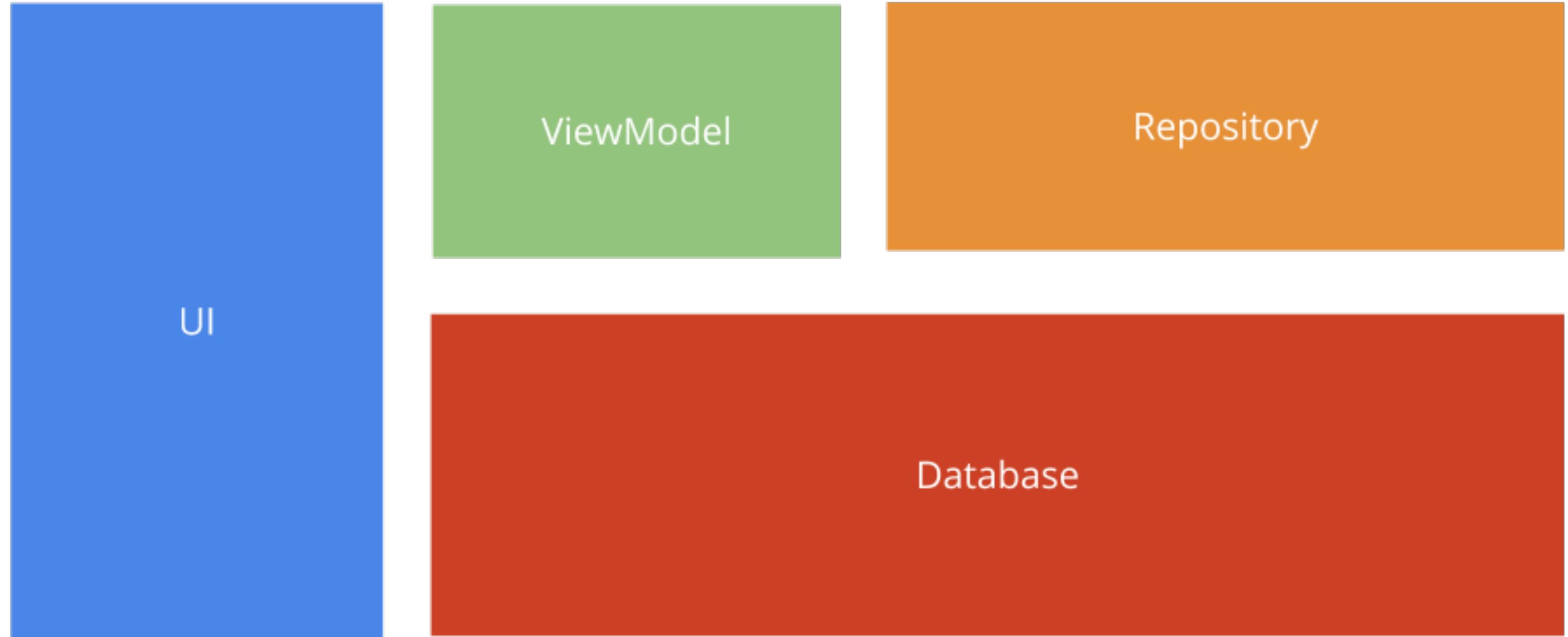


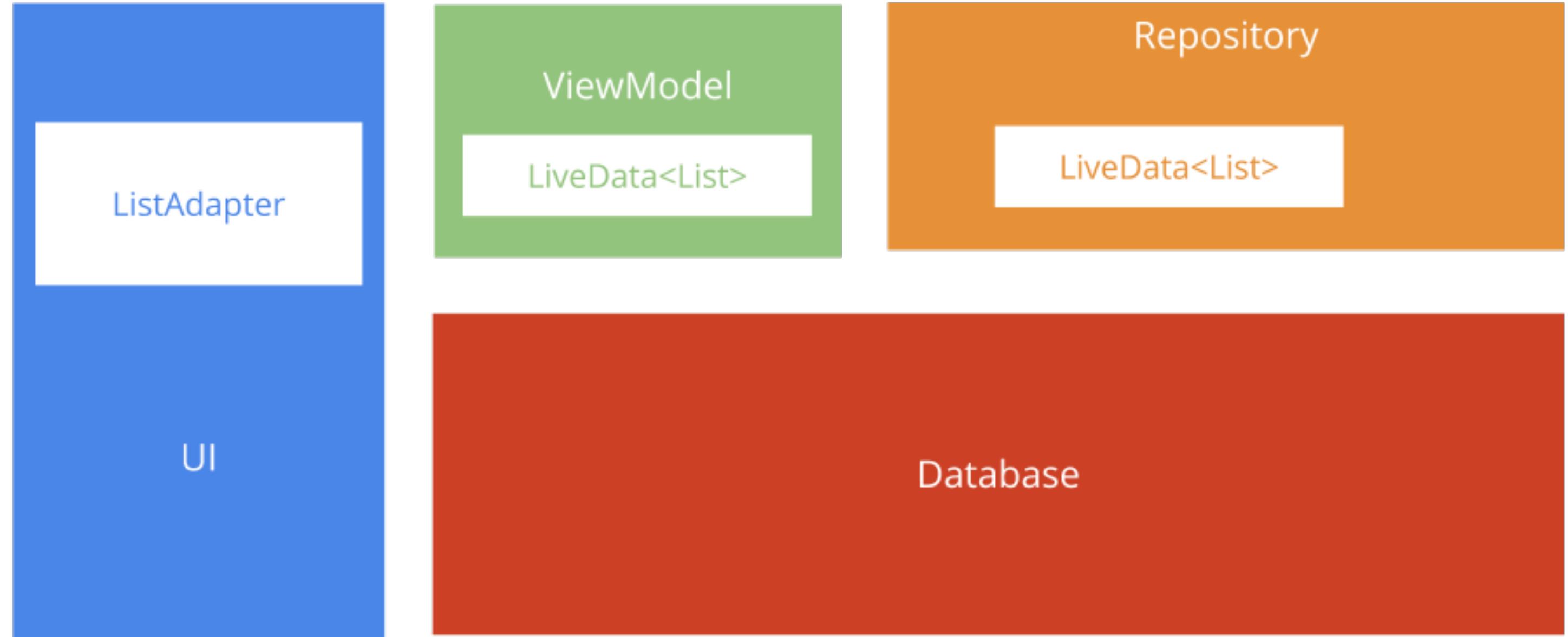


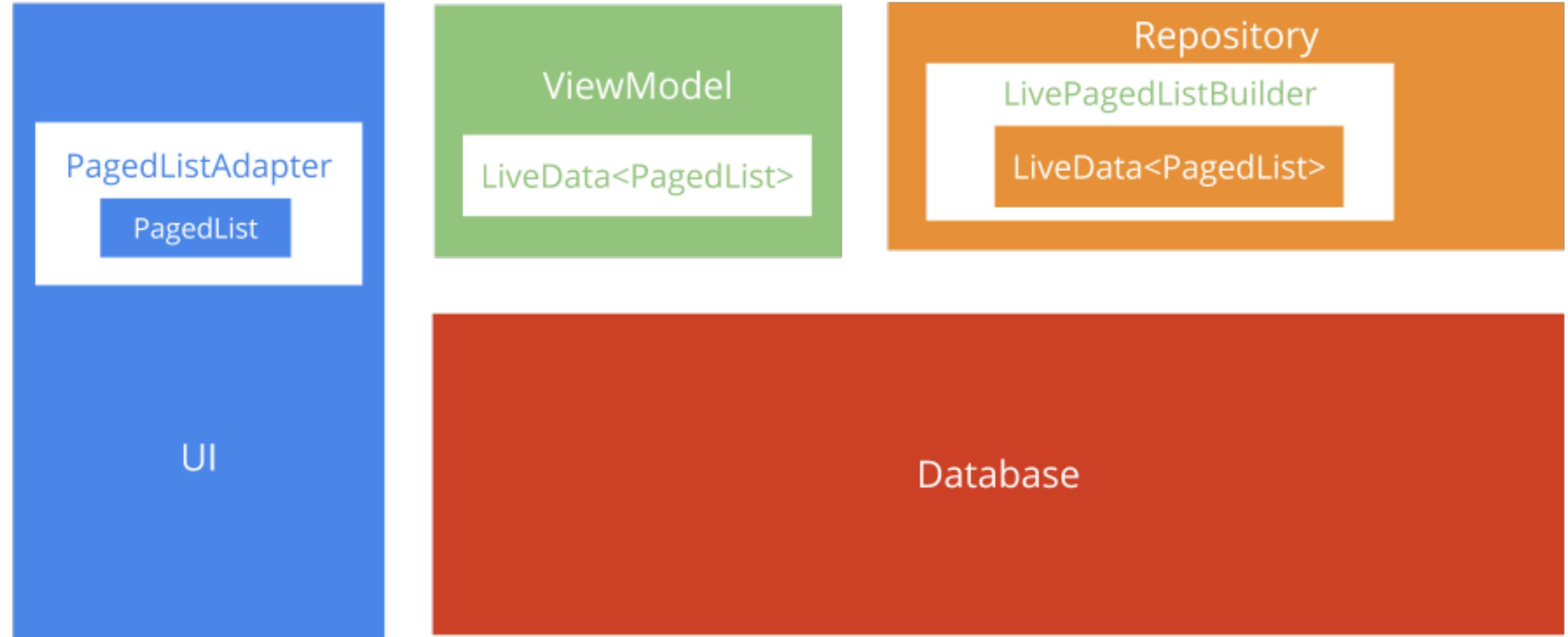




@askashdavies

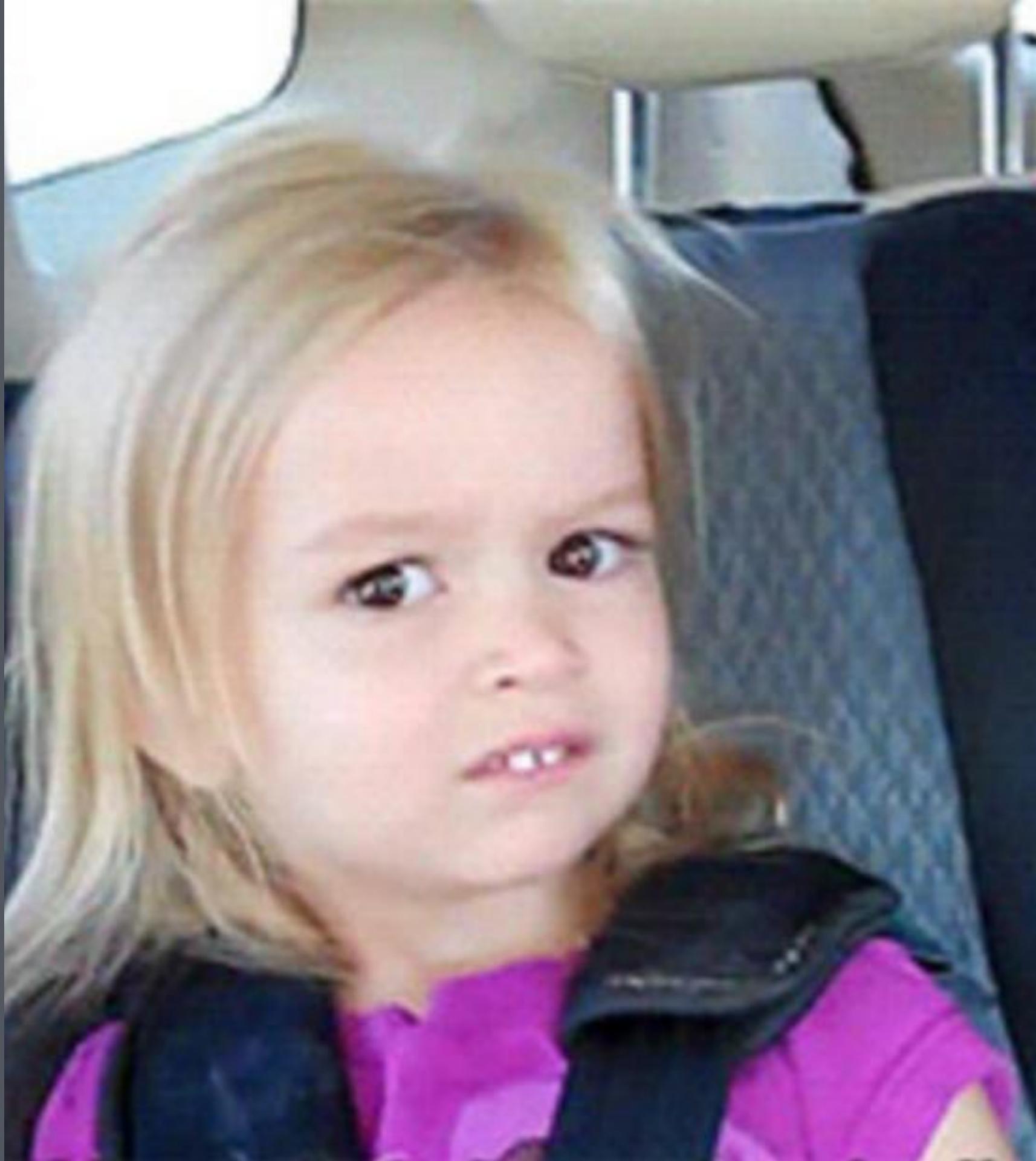






PagedList ?

@askashdavies



PagedList

PagedList<T> : List<T>

PagedList

PagedListBuilder

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

PagedList

PagedListBuilder

- LiveDataPagedListBuilder
- RxPagedListBuilder
- FlowPagedListBuilder¹

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

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
    }
}
```

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
    }
}
```

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?

Remote Data Source

Backend 

Remote Data Source

Index 

DataSource<K, V>

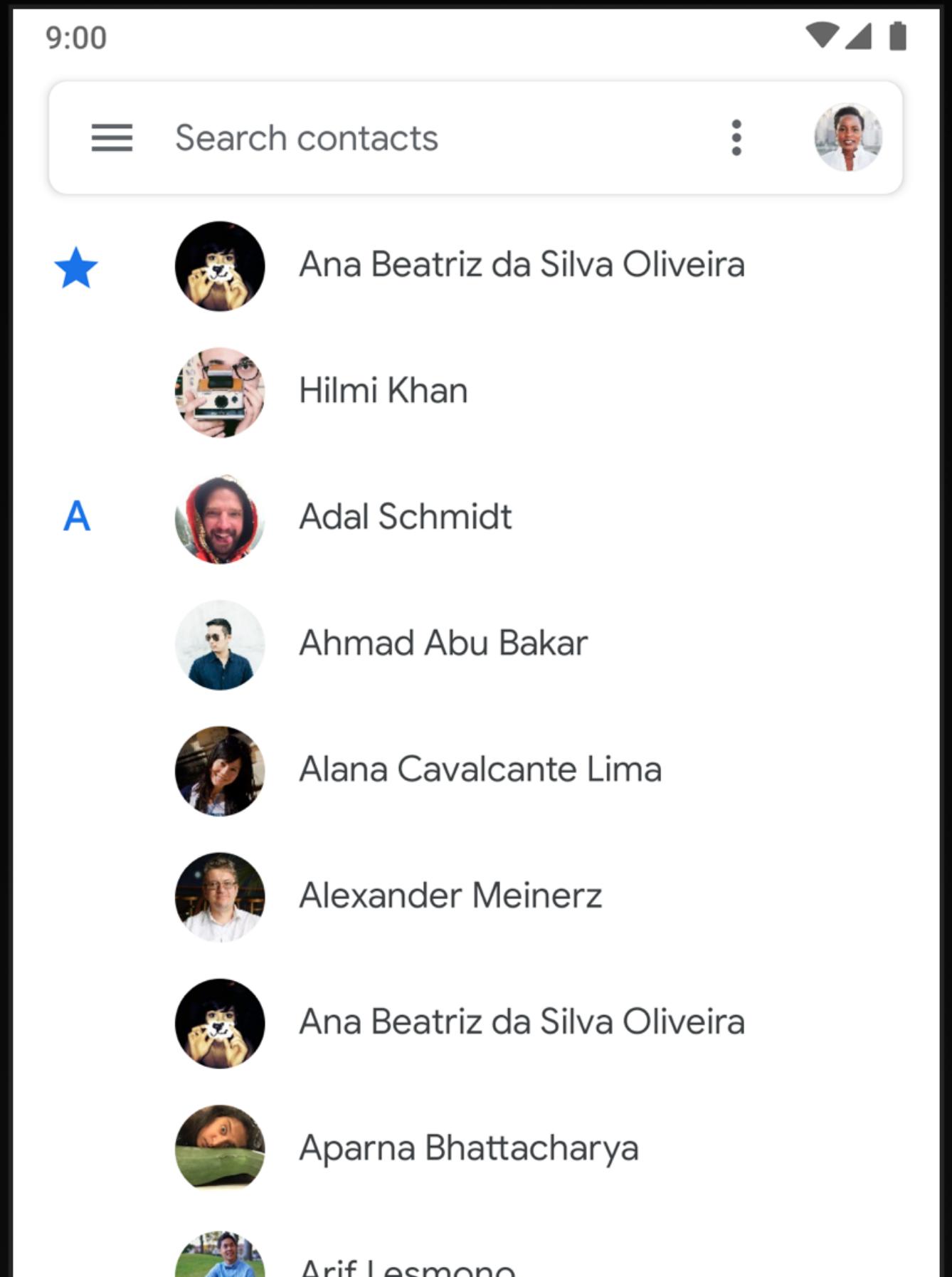
- PositionalDataSource 🚗
- ItemKeyedDataSource 🔑
- PageKeyedDataSource 12
34

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

@askashdavies

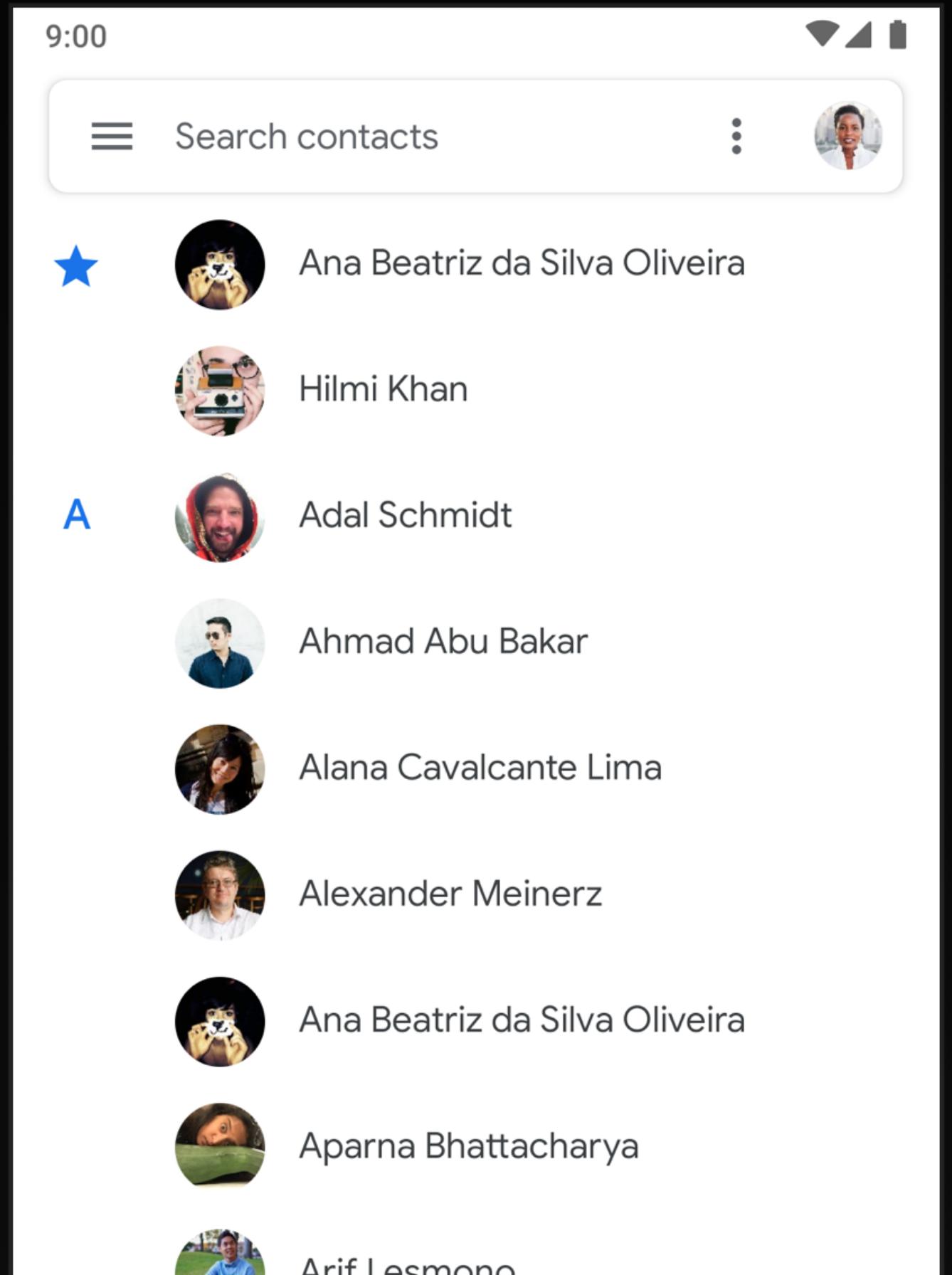


PositionalDataSource

PositionalDataSource<User>

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

@askashdavies



ItemKeyedDataSource

ItemKeyedDataSource<String, User>

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

LIST	FAVORITES	RECENT
abwechseln	exchange, vary	
achten	regard, respect	
ächzen	moan, groan	
adoptieren	adopt	
ähneln	resemble, take after	
ahnen	have a feeling about, suspect	
akzeptieren	accept	
alarmieren	alarm	
alliiieren	ally	
amüsieren	amuse	
an sein	be on	
analysieren	analyze	
anbauen	cultivate, add on by building	
anbeißen	bite at, take a bite	
anbeten	adore, worship	
anbieten	offer	

ItemKeyedDataSource

ItemKeyedDataSource<String, User>

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

@askashdavies

LIST	FAVORITES	RECENT
abwechseln	exchange, vary	
achten	regard, respect	
ächzen	moan, groan	
adoptieren	adopt	
ähneln	resemble, take after	
ahnen	have a feeling about, suspect	
akzeptieren	accept	
alarmieren	alarm	
alliieren	ally	
amüsieren	amuse	
an sein	be on	
analysieren	analyze	
anbauen	cultivate, add on by building	
anbeißen	bite at, take a bite	
anbeten	adore, worship	
anbieten	offer	

PageKeyedDataSource

PageKeyedDataSource<String, User>

- Common for API responses
- GitHub
- Twitter
- Reddit

@askashdavies

KotlinCoroutinesExamples

Kotlin Coroutines Examples

Kotlin

98 ★

98 ○

PoiShuhui-Kotlin

[Deprecated]一个用Kotlin写的简单漫画APP

Kotlin

958 ★

958 ○

KotlinRxMvpArchitecture

Clean MVP Architecture with RxJava +
Dagger2 + Retrofit2 + Mockito + Fresco +
EasiestGenericRecyclerAdapter using Kotlin...

Kotlin

93 ★

93 ○

AndroidKotlinCoroutine

Use kotlin coroutine and retrofit to request
network in android application

Kotlin

9 ★

9 ○

Kotlin-Coroutine-Examples

Kotlin

9 ★

PageKeyedDataSource

PageKeyedDataSource<String, User>

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

@askashdavies

KotlinCoroutinesExamples

Kotlin Coroutines Examples

Kotlin

98 ★

98 ○

PoiShuhui-Kotlin

[Deprecated]一个用Kotlin写的简单漫画APP

958 ★

958 ○

KotlinRxMvpArchitecture

Clean MVP Architecture with RxJava +
Dagger2 + Retrofit2 + Mockito + Fresco +
EasiestGenericRecyclerAdapter using Kotlin...

93 ★

93 ○

AndroidKotlinCoroutine

Use kotlin coroutine and retrofit to request
network in android application

9 ★

9 ○

Kotlin-Coroutine-Examples

Kotlin

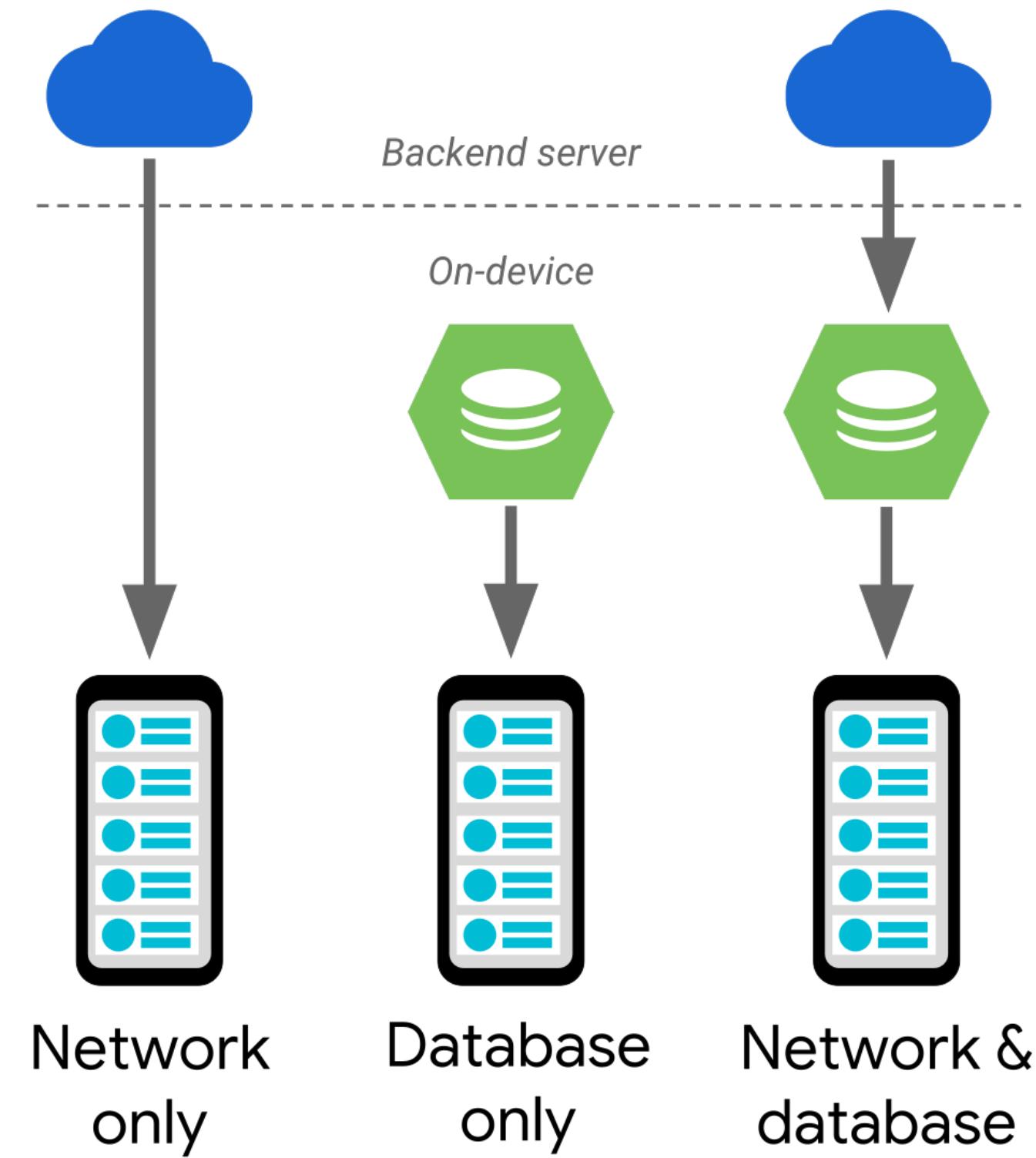
9 ★



@askashdavies

Architecture





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) { }  
}
```

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
- **Android Paging Codelab**
codelabs.developers.google.com/codelabs/android-paging/#0
- **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

KotlinCoroutinesExamples

Kotlin

Kotlin Coroutines Examples

98



98



PoiShuhui-Kotlin

Kotlin

[Deprecated]一个用Kotlin写的简单漫画APP

958



958



KotlinRxMvpArchitecture

Kotlin

Clean MVP Architecture with RxJava +

93



Dagger2 + Retrofit2 + Mockito + Fresco +

EasiestGenericRecyclerAdapter using Kotlin...

93



AndroidKotlinCoroutine

Kotlin

Use kotlin coroutine and retrofit to request
network in android application

9



9



Kotlin-Coroutine-Examples

Kotlin

Example for using Coroutines

9



Thanks!

@askashdavies

