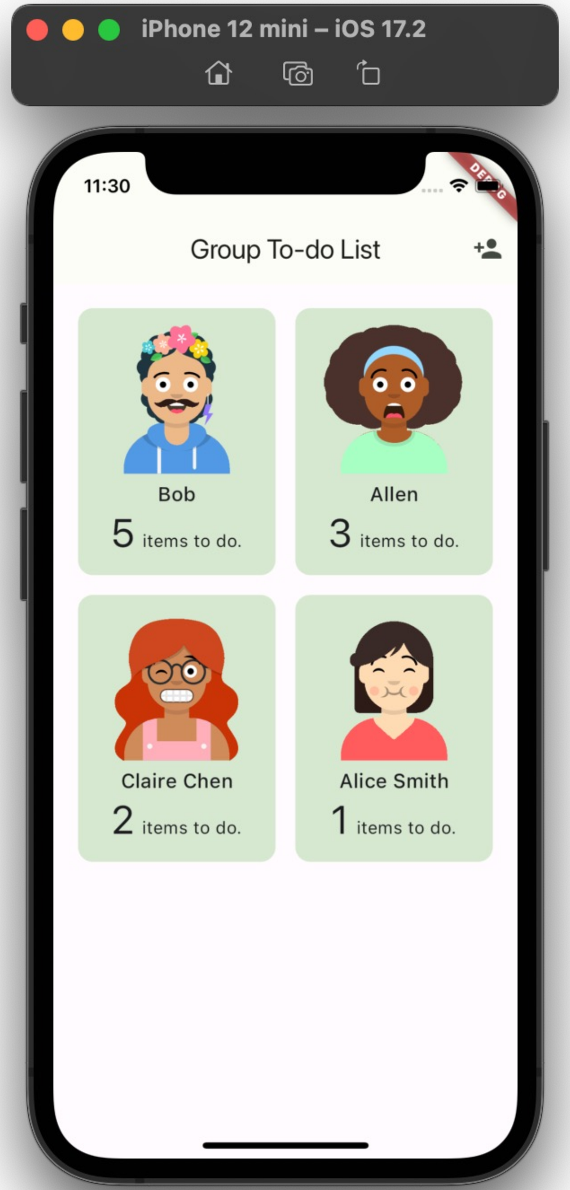
Backend Database & MVVM

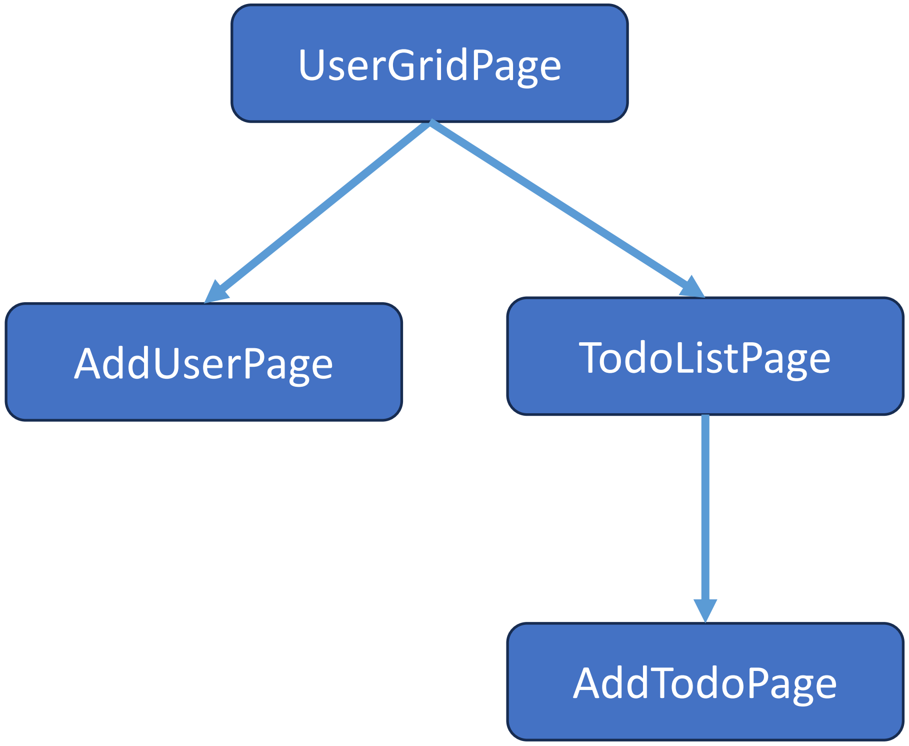
Shan-Hung Wu
CS, NTHU

Group To-do List

- Fluttermoji
- Create and delete to-dos
- Mark “done” or reassign to other user
- Persistent data

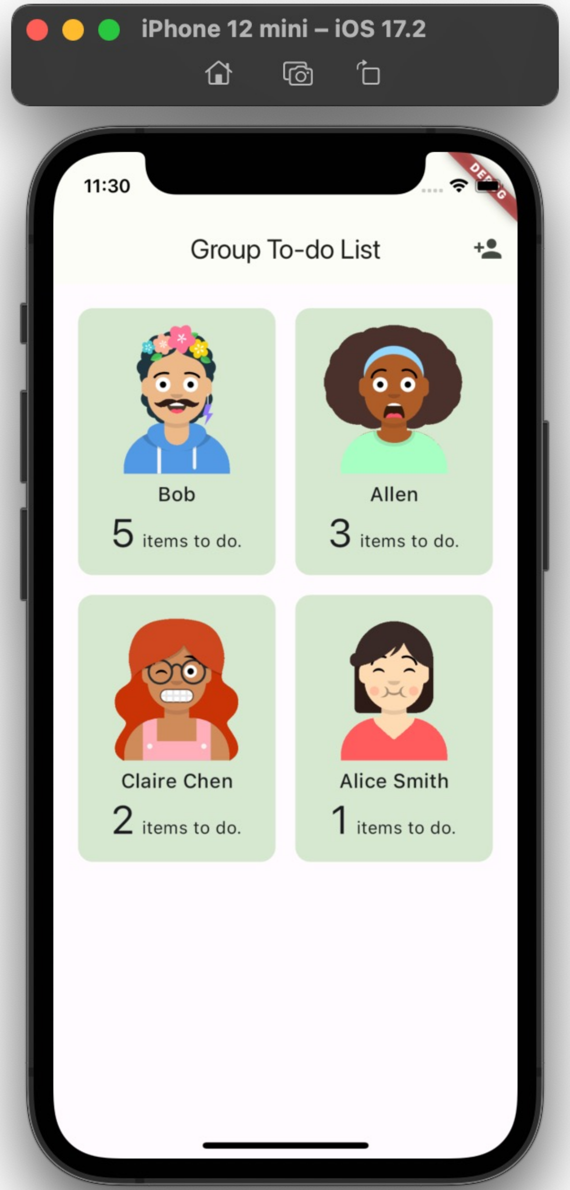


Information Architecture



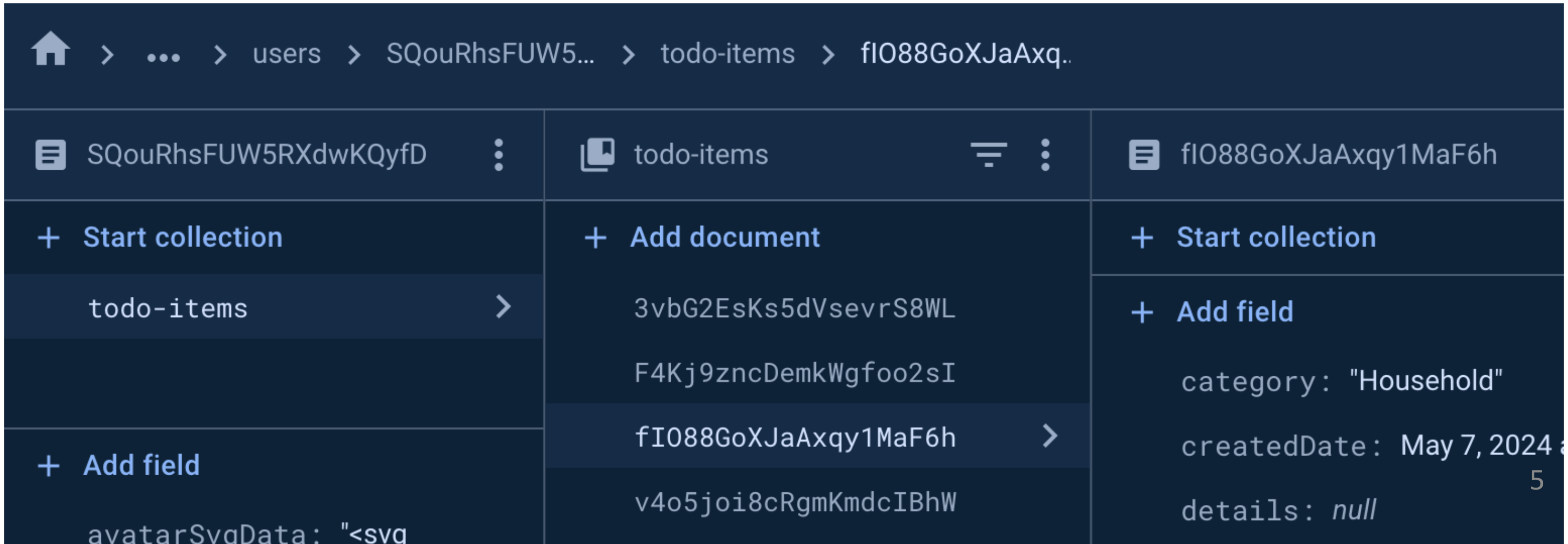
Outline

- Google Cloud Firestore as backend database
- MVVM architecture
- Advanced UI
  - Line counting in `ListTile` & post-frame callbacks
  - Animation with `AnimatedList`



Google Cloud Firestore

- A NoSQL database-as-a-service in the cloud
- Stores ***documents*** & ***collections***
- Supports CRUD ops, queries, and ***transactions***
- Support ***listening*** to dynamic query results



users > SQouRhsFUW5... > todo-items > fIO88GoXJaAxq..

SQouRhsFUW5RXdwKQyfD
+ Start collection
todo-items
+ Add field
avatarSvgData: "<svg

todo-items
+ Add document
3vbG2EsKs5dVsevrS8WL
F4Kj9zncDemkWgfoo2sI
fIO88GoXJaAxqy1MaF6h
v4o5joi8cRgmKmdcIBhW

fIO88GoXJaAxqy1MaF6h
+ Start collection
+ Add field
category: "Household"
createdDate: May 7, 2024
details: null

Transactions

- Group CRUD operations into an ***atomic*** unit
- All operations succeed or fail together

- When to use?
- Toggling `isDone` prop of to-do item
- Reassigning to-do item (ownership transfer)

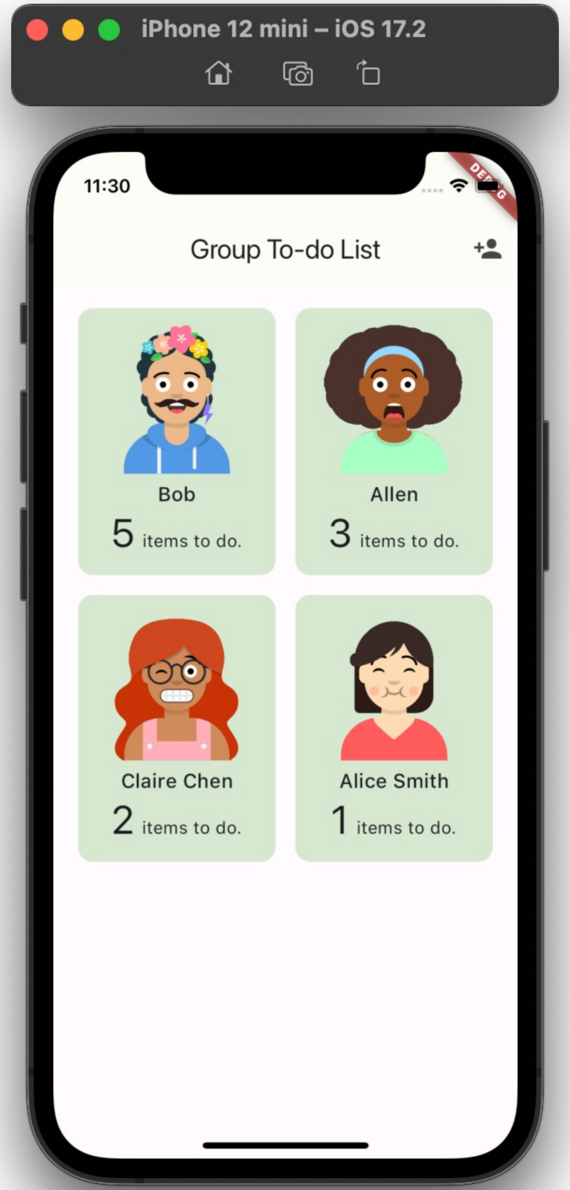
Listening & Server Streaming

- Firestore uses gRPC to support listening
  - Base on HTTP/2
  - Binary serialization (via Protocol Buffers)
  - ***Server streaming*** (and/or client streaming)

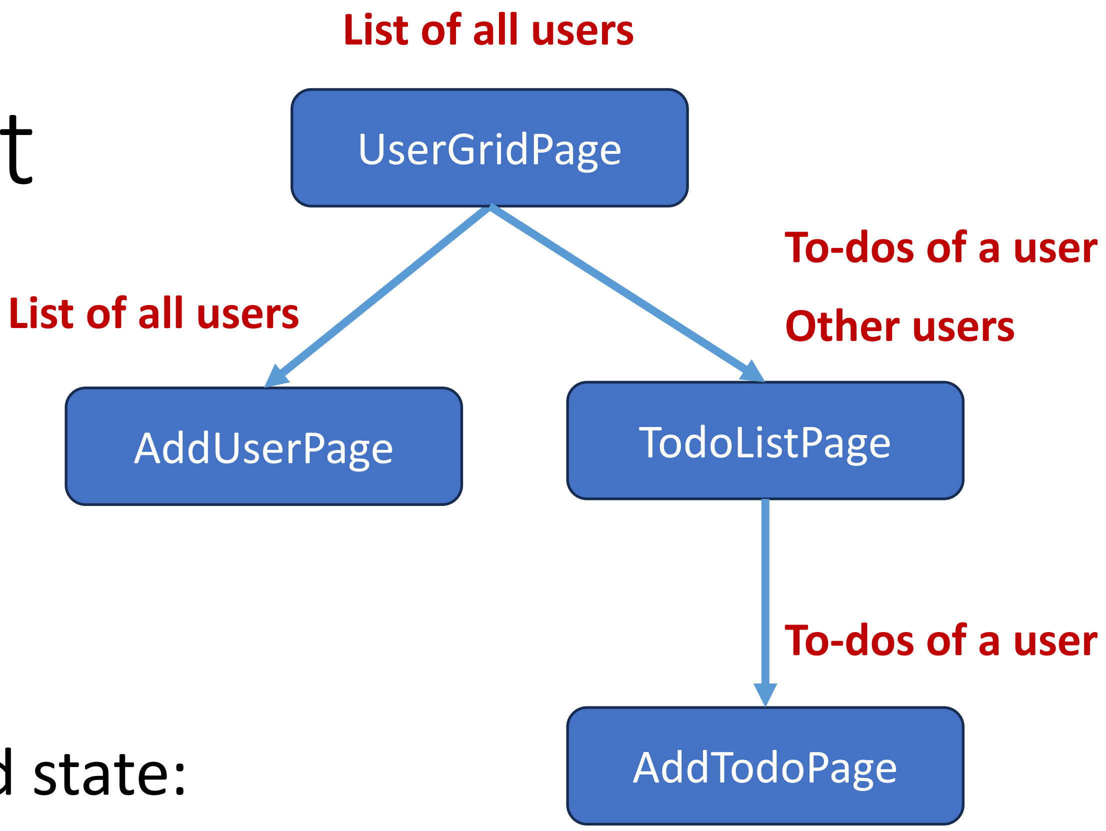
- If any doc in query result changes, the ***entire*** query result is pushed again
- Works seamlessly with decelerative UI

Outline

- Google Cloud Firestore as backend database
- MVVM architecture
- Advanced UI
  - Line counting in `ListTile` & post-frame callbacks
  - Animation with `AnimatedList`

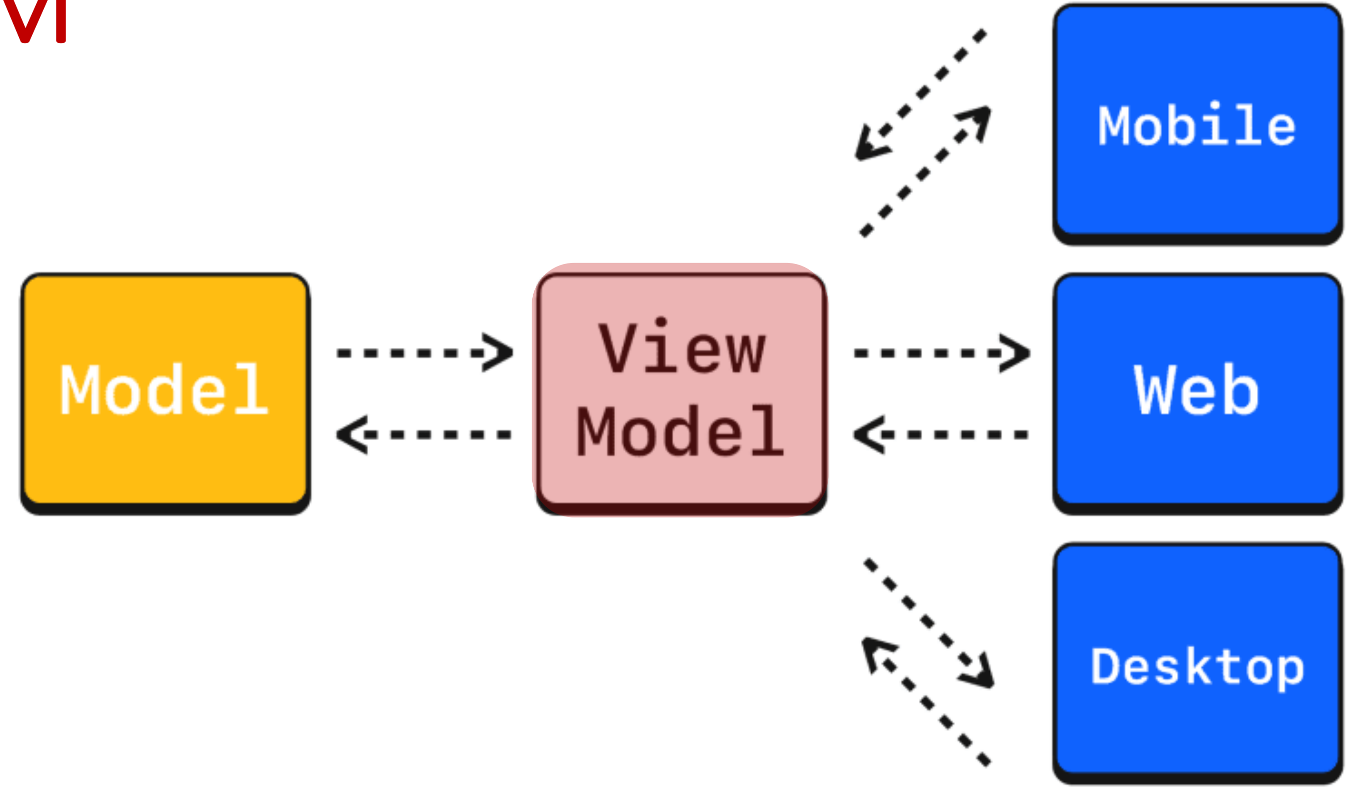


State Management



- Non-globally shared state:
- “To-do items” created dynamically based on user ID
- “Other users” created based on user ID and updated whenever “all users” change

MVVM



- ***Model***: data & business logic
- ***View***: UI & events (e.g., user actions)
- ***View Model***: shared state & event handling

Example

```
// model
class User{ ... }

// repository
class UserRepository {
  Stream<List<User>> streamUsers() {
    ...
  }
}
```

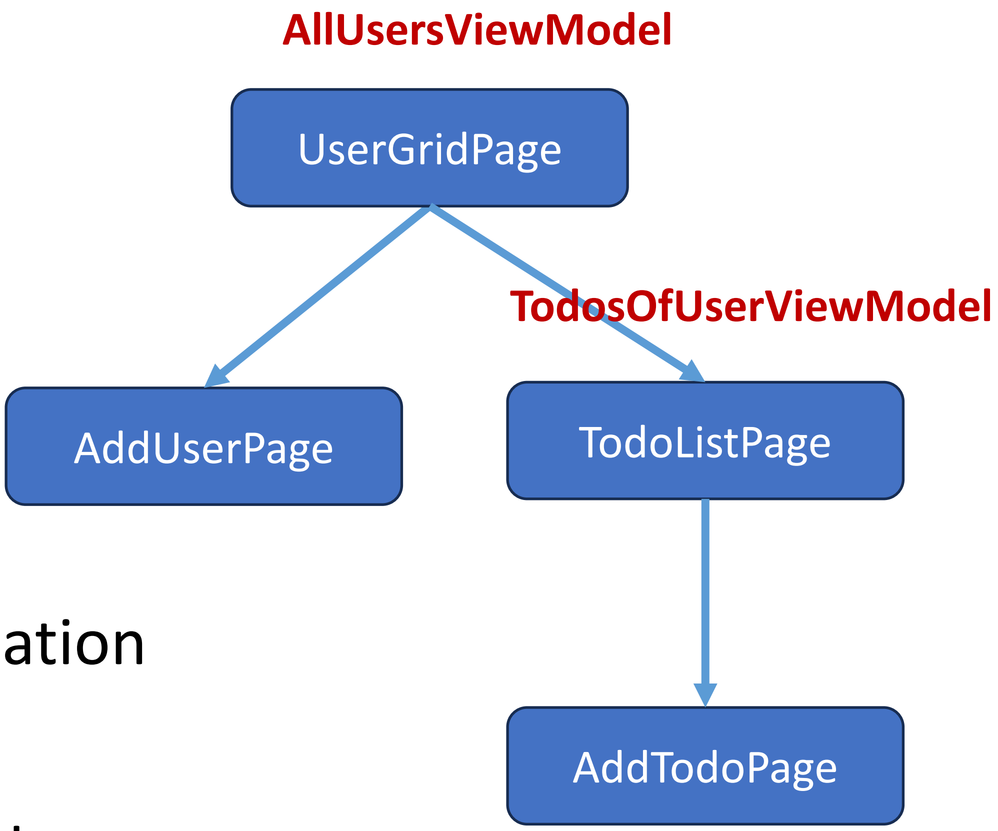
```
// view model as ChangeNotifier
class OtherUsersViewModel with ChangeNotifier {
  final List<User> otherUsers = ...;
}
```

```
// view as widget (either stateless or stateful)
class TodoListView extends StatelessWidget {
  Widget build(BuildContext context) {
    final otherUsers = Provider.of<OtherUsersViewModel>(
      context,
      listen: true
    ).otherUsers;
  }
}
```

Benefits

- Separation of concerns
- View “binds to” View Model to allow declarative UI
  - via `notifyListeners()`
- Loosely coupled code modules
  - ***Dependency injection*** via `Provider`
- Easier unit testing
  - Isolated dependency injection for each target under test

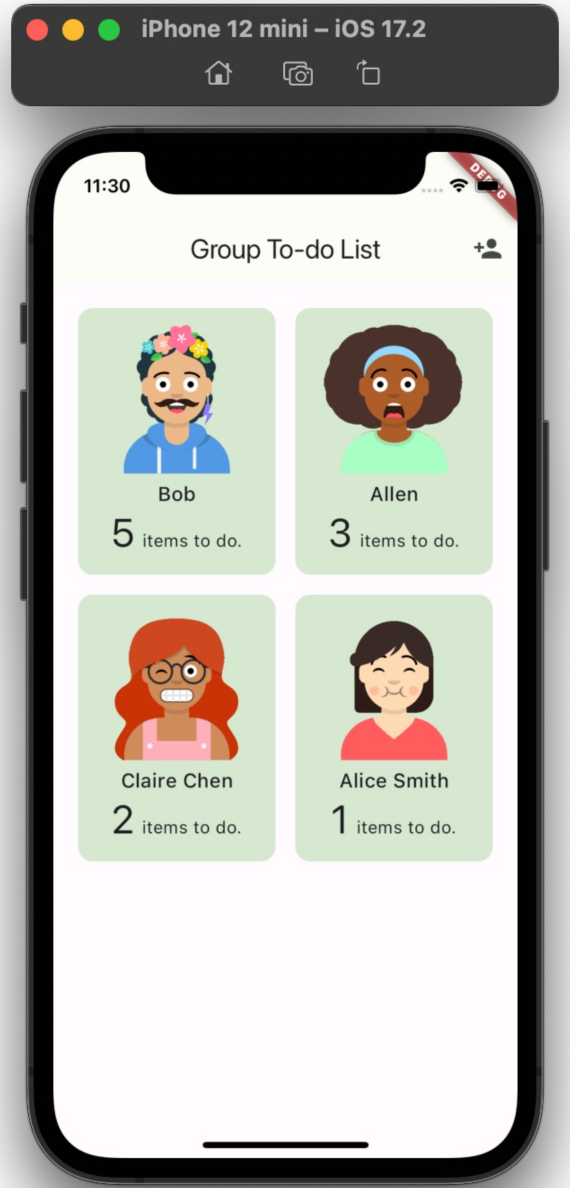
Dependency Injection



- Can follow the information architecture
- But information architecture **≠** widget tree
- In route config, glue Views with View Models by using Providers in **`ShellRoute`**
- `TodosOfUserViewModel` is a `ProxyProvider` depending on `AllUsersViewModel`

Outline

- Google Cloud Firestore as backend database
- MVVM architecture
- Advanced UI
  - Check `mounted` after async gap
  - Line counting in `ListTile` & post-frame callbacks
  - Animation with `AnimatedList`



AnimatedList

```
AnimatedList(
  key: _myListKey,
  initialItemCount: _myItems.length,
  itemBuilder: (context, index, animation) {
    return ...;
  },
);
```

```
// insert data and then animate
_myItems.insert(index, element);
_myListKey.currentState.insertItem(index);
```

```
// animate and then delete data
var removedItem = _myItems[index];
_myListKey.currentState.removeItem(
  index,
  (context, animation) => MyListItem(removedItem),
);
_myItems.removeAt(index);
```

- `initialItemCount` is only used during `initState()`
  - Not during subsequent `rebuild()`
- `AnimatedList` adjusts its internal item count after `insertItem()` and `removeItem()`