UR Nyarugenge

College of Science and Technology

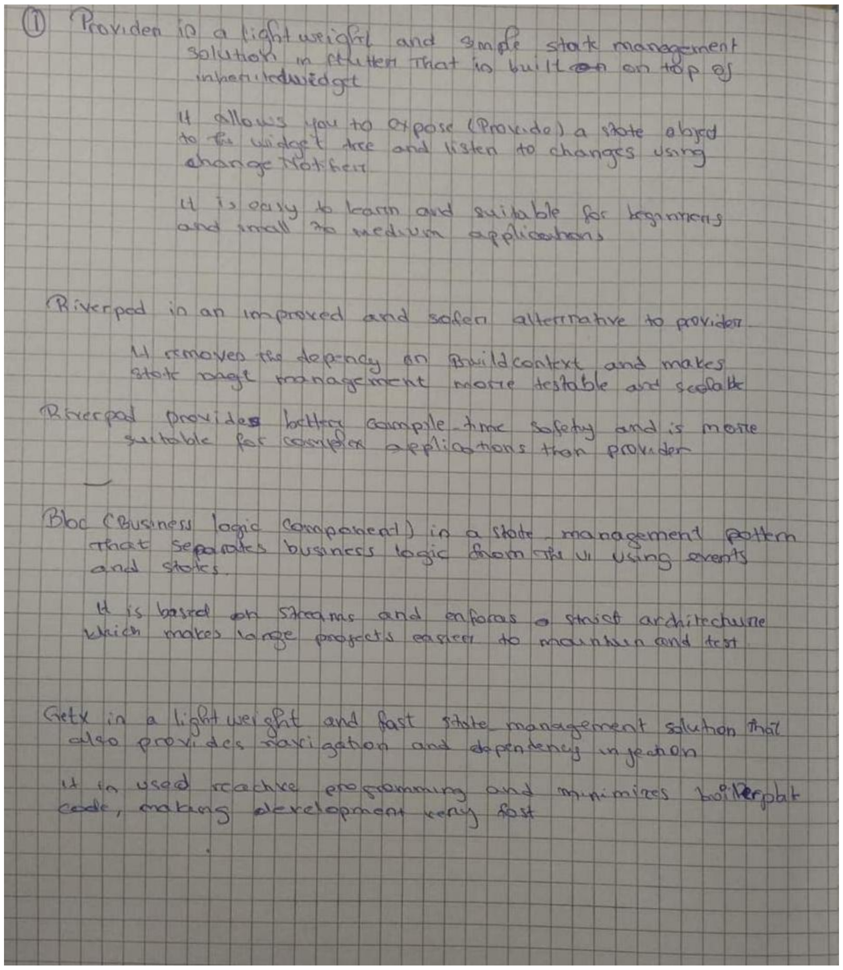
Computer and software engineering

Year 3

223007635

Assignment

1.



(1) Provider is a lightweight and simple state management solution in flutter That is built ~~on~~ on top of inheritedwidget

It allows you to expose (Provide) a state object to the widget tree and listen to changes using changeNotifier

It is easy to learn and suitable for beginners and small to medium applications

Riverpod is an improved and safer alternative to provider.

It removes the depency on Buildcontext and makes state page management more testable and scalable

Riverpod provides better compile-time safety and is more suitable for complex applications than provider

Bloc (Business logic component) is a state management pattern That separates business logic from the ui using events and states.

It is based on streams and enforces a strict architecture which makes large projects easier to maintain and test

Getx is a lightweight and fast state management solution that also provides navigation and dependency injection

it is used reactive programming and minimizes boilerplat code, making development very fast

2.

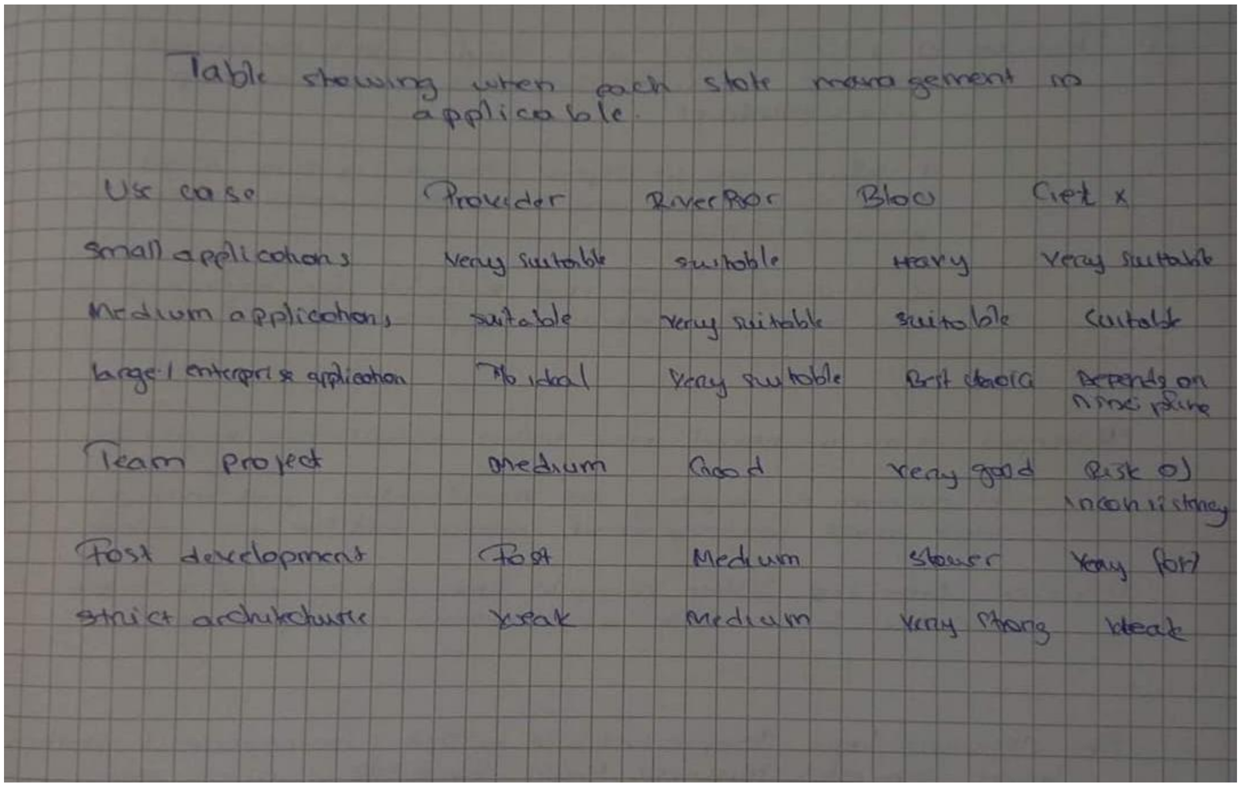


Table showing when each state management is applicable.

Use case	Provider	RiverPod	Bloc	Get x
small applications	Very suitable	suitable	Heavy	Very suitable
Medium applications	suitable	Very suitable	suitable	suitable
large / enterprise application	No ideal	Very suitable	Best choice	Depends on discipline
Team project	Medium	Good	Very good	Risk of inconsistency
Fast development	Fast	Medium	slower	Very fast
strict architecture	weak	Medium	Very strong	weak

3. **Adding dependency**

First, add Provider to the project dependencies.

```
dependencies:
  provider: ^6.0.0
```

This allows the project to use Provider classes such as ChangeNotifierProvider and Consumer

Creating a state class

A state class is created by extending ChangeNotifier

```
class CounterProvider extends ChangeNotifier {
  int _count = 0;

  int get count => _count;
}
```

This class stores the application state and exposes it to the UI

Providing the state

The state object is placed above the widget tree using a provider widget

```
ChangeNotifierProvider(
  create: (_) => CounterProvider(),
  child: MyApp(),
)
```

This makes the state available to all widgets under MyApp

Accessing the state

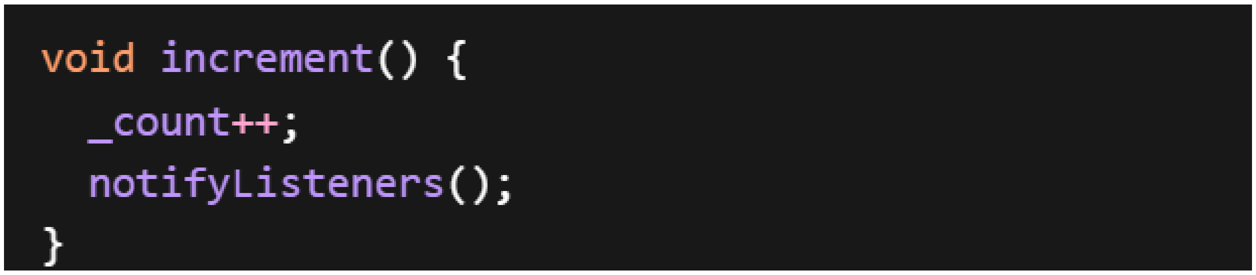
Widgets can access the state using context.watch() or Consumer

```
final counter = context.watch<CounterProvider>();
```

This allows the widget to read the current value of the state.

Updating the state

The state is updated inside the provider class and then notifies listeners.



```
void increment() {
  _count++;
  notifyListeners();
}
```

Calling notifyListeners() informs all listening widgets that the state has changed

How UI rebuild happens

When notifyListeners() is called:

- Provider informs all widgets that are listening to this provider
- Only the widgets that use context.watch() or Consumer are rebuilt
- The rest of the widget tree is not affected

This makes UI updates efficient and controlled.