🧠 Dart + GetX Cheat Sheet (For Real Apps)

◇ 1. Model Classes with fromJson / toJson

- Use for API or storage data mapping.

```dart
class Product {
  final String name;
  final double? price;

  Product({required this.name, this.price});

  factory Product.fromJson(Map<String, dynamic> json) => Product(
    name: json['name'],
    price: json['price']?.toDouble(), // handle int or double
  );

  Map<String, dynamic> toJson() => {
    'name': name,
    'price': price,
  };
}
```

☑ Used in API response handling, local storage, Cloud Firestore, or any structured data transfer.

◇ 2. Nested Model Example

```dart
class Author {
  final String name;
  Author({required this.name});

  factory Author.fromJson(Map<String, dynamic> json) => Author(name:
json['name']);
  Map<String, dynamic> toJson() => {'name': name};
}

class Book {
  final String title;
  final Author author;

  Book({required this.title, required this.author});

  factory Book.fromJson(Map<String, dynamic> json) => Book(
    title: json['title'],
    author: Author.fromJson(json['author']),
  );
  Map<String, dynamic> toJson() => {
    'title': title,
    'author': author.toJson(),
  };
}
```

◇ 3. GetStorage Basics (⚙ Local key-value store)

- Add package: `get_storage`
- Initialize in `main()`:

```
await GetStorage.init();
final box = GetStorage();
```

☑ Store a simple value:

```
box.write('token', 'abc123');
String token = box.read('token');
```

☑ Store a map/model:

```
box.write('profile', user.toJson());
User user = User.fromJson(box.read('profile'));
```

☑ Store a list of models:

```
List<Product> productList = [...];
box.write('cart', productList.map((p) => p.toJson()).toList());

List<Product> stored = (box.read('cart') as List)
  .map((item) => Product.fromJson(item))
  .toList();
```

📌 Good Practices:

- Always call `.toJson()` when storing model/map
- Always `.fromJson()` when retrieving
- Avoid storing heavy objects (e.g., images, binary)

◇ 4. map() vs forEach() (⚔️ Know the difference)

Feature	`map()`	`forEach()`
Purpose	Transform each item	Perform action for each item
Returns	New list	Returns `void`
Use case	Convert, transform, build new list	Print, side effects, UI updates

```
// map() example - convert to labels
List<String> labels = products.map((p) => p.label).toList();

// forEach() example - just printing
products.forEach((p) => print(p.label));
```

◇ 5. Fold - Sum or Reduce Complex Lists

```
double total = products.fold(0, (sum, item) => sum + (item.price ?? 0));
```

◇ 6. Enums for Fixed Options

```
enum UserRole { admin, editor, viewer }

class User {
  final UserRole role;
  String get label => "Role: ${role.name}";
}
```

◇ 7. Inheritance & Abstract Classes

```
abstract class Shape {
  double get area;
  void describe();
}

class Circle extends Shape {
  final double radius;
  double get area => 3.14 * radius * radius;

  void describe() => print("Area: $area");
}
```

🔄 Dart Tips & Practice Patterns

- ☑ Use `List.map().toList()` when transforming items
- ☑ Use `List.where().toList()` for filtering
- ☑ Use `fold()` for sum/average
- ☑ Use `enum` for roles/status types
- ☑ Use `abstract` for contracts (like `Shape`, `ApiService`)
- ☑ Use `GetStorage` to persist:
    - tokens, flags (`isLoggedIn`)
    - last used filter/search
    - offline data caching

🧪 Practice Ideas for Mastery

1. Store & retrieve a `User` object from `GetStorage`
2. Save and restore a list of `Book` objects
3. Store nested structure (Catalog with List)
4. Track app theme (`isDarkMode`) using bool in GetStorage
5. Make reusable `StorageService` using `GetxService` + `GetStorage`