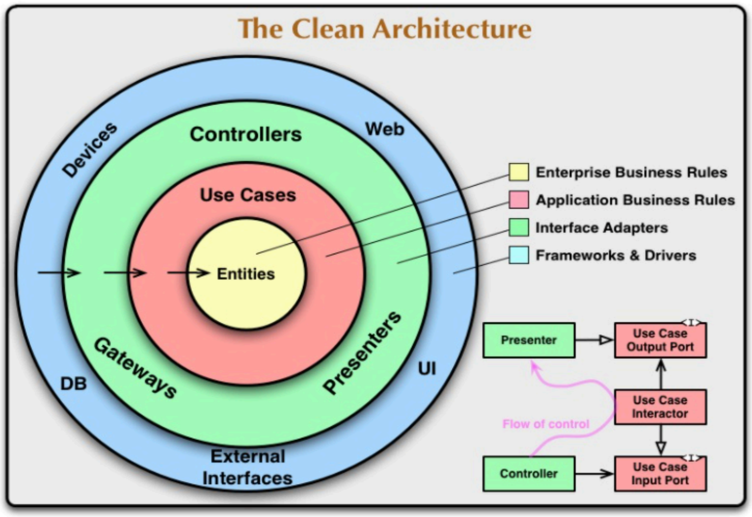
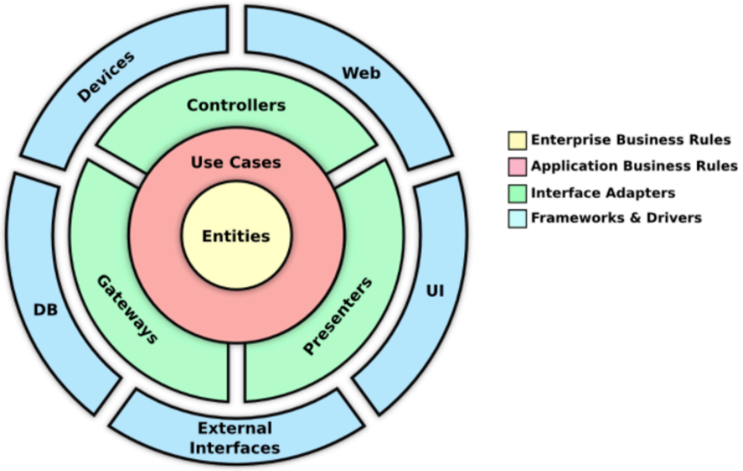
1. What is the CLEAN Architecture?

- The CLEAN Architecture was created by Robert C. Martin (“Uncle Bob”) to make code more maintenable and easier to manage if modifying for different services.
- Separation of UI, Entity, Models, and Services



2. Clean Architecture Goals



- Isolation of Responsibilities

- Business Logic (BLoc)
- UI
- Use Cases
- Service Adapters

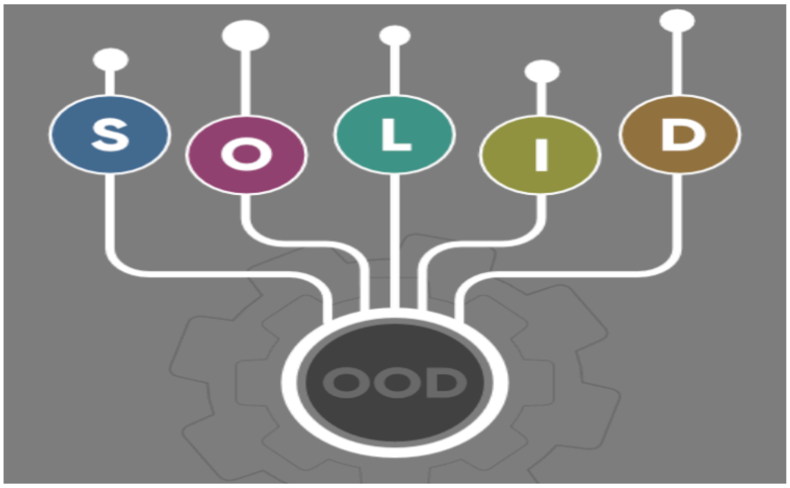
- Decoupled Structure

- Easily swap services, databases, etc.
- Classes are not dependent on each other
  - Entities are concerned only about their own functionalities
- Use Cases
- Service Adapters

- Single Purpose Entities

- Classes serve one purpose

3. Built on SOLID Principles



The CLEAN Architecture is built upon the "SOLID" principles which are:

- **S: Single Responsibility**

  - Classes/Entities should have one purpose, not be God classes

- **O: Open/Closed Approach**

  - Classes should be inheritable for functionality but not modifiable

- **L: Liskov Substitution**

  - Every subclass should be substitutable for their parent class

- **I: Interface Segregation**

  - Don’t use a generalized interface as a one-all be all interface. Instead, implement specific interfaces for their specific needs.

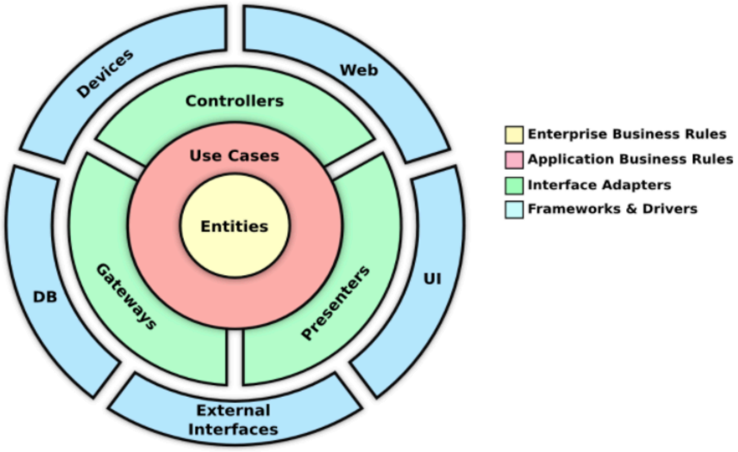
- **D: Dependency Inversion**

  - Entities should depend on abstractions not on each other.

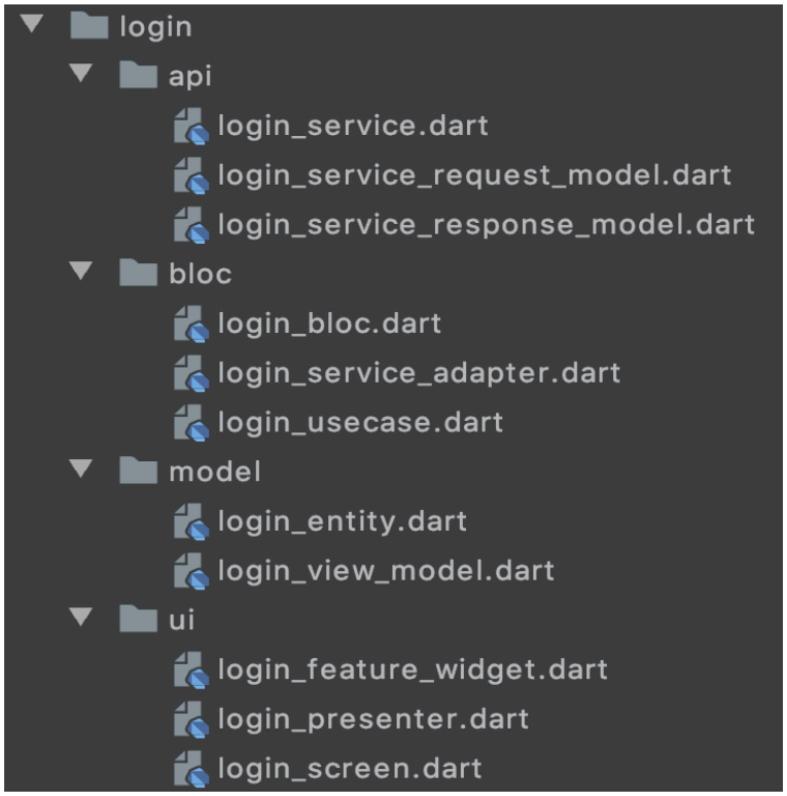
4. CLEAN Architecture Directory Structure

Directory Structure:

- Feature Directory (Separation of layers)
  - api **(Service and Service Models directory)**
  - bloc **(Business Logic, Use Case, & Adapter directory)**
  - model **(Entity and View Model directory)**
  - ui **(Feature Widget, Presenter, and UI directory)**

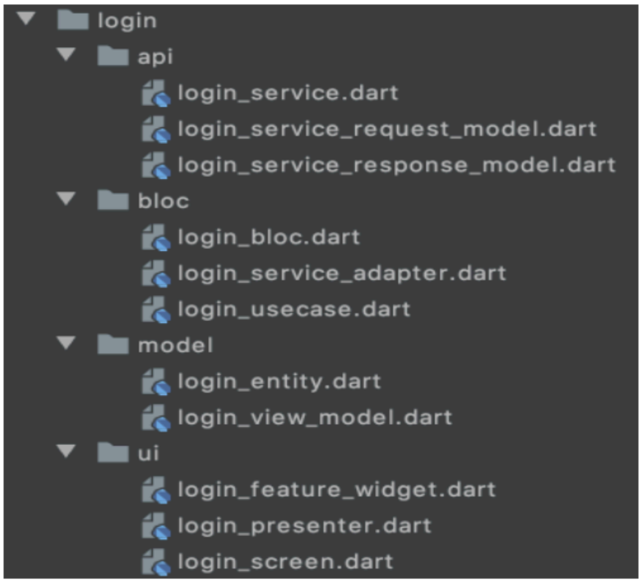


- Example Directory Structure ↓



```
▼ login
  ▼ api
      login_service.dart
      login_service_request_model.dart
      login_service_response_model.dart
  ▼ bloc
      login_bloc.dart
      login_service_adapter.dart
      login_usecase.dart
  ▼ model
      login_entity.dart
      login_view_model.dart
  ▼ ui
      login_feature_widget.dart
      login_presenter.dart
      login_screen.dart
```

5. Directory Structure Examined



```
login
  api
    login_service.dart
    login_service_request_model.dart
    login_service_response_model.dart
  bloc
    login_bloc.dart
    login_service_adapter.dart
    login_usecase.dart
  model
    login_entity.dart
    login_view_model.dart
  ui
    login_feature_widget.dart
    login_presenter.dart
    login_screen.dart
```

Service Models

- The Request Model is created and provided to the Service Adapter
- The Response Model is created and provided to the Service Adapter
- The Service Adapter maps the JSON response to the Response Model Entity

BLoc

- Executes the use case(s) created
- Utilizes the pipe(s) created (setup listeners and send data)
- Properly disposes pipe with dispose method
- Provides BLoc constructor
- Sets up the Use Case(s) with a pipe to send out the view model
    - However, the View Model call back functionality resides in the Use Case

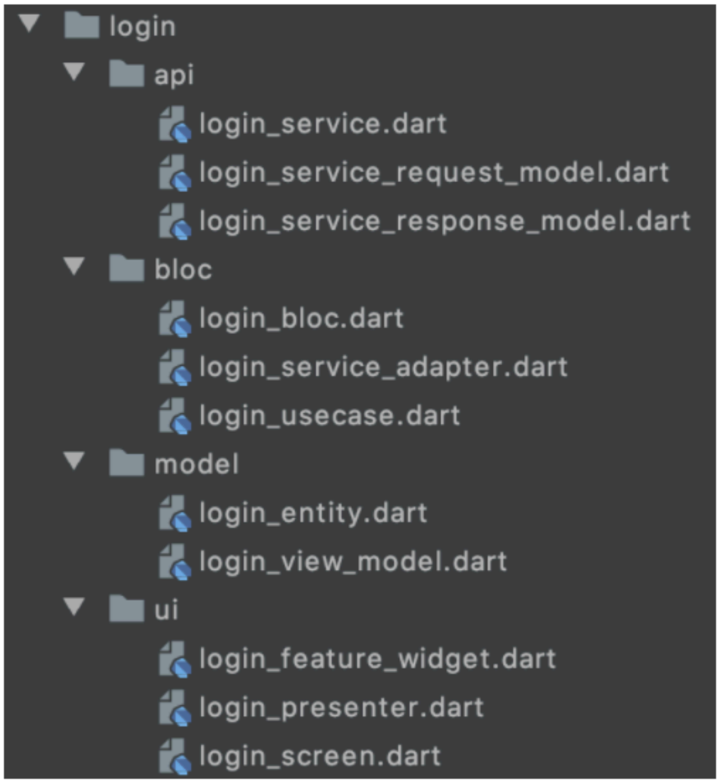
Service Adapter

- Provides the following to the specific service
    - Entity Model
    - JSON Model
    - Service Response Model
    - Service Class Being Called
- Provides method for mapping the response to the Entity Model

Use Case

- Utilizes a Repository Scope
- Provides Constructor for Specific Use Case Types
- Provides linkage to parent Repository Scope
- Define linkage to children Entity Scope Types
- Enables construction of View Model

6. Directory Structure Examined Continued...



```
login
    api
        login_service.dart
        login_service_request_model.dart
        login_service_response_model.dart
    bloc
        login_bloc.dart
        login_service_adapter.dart
        login_usecase.dart
    model
        login_entity.dart
        login_view_model.dart
    ui
        login_feature_widget.dart
        login_presenter.dart
        login_screen.dart
```

Entity

- The object types created for the particular use case(s)
- Also known as the Domain Model

View Model

- View Model will be the middle man between the UI and the Use Case
- The object mapping to the UI to be displayed
- A View Model List may also be provided instead of a View Model if multiple JSON objects are returned

Feature Widget

- Creates BLoc
- Associates a Presenter to the BLoc

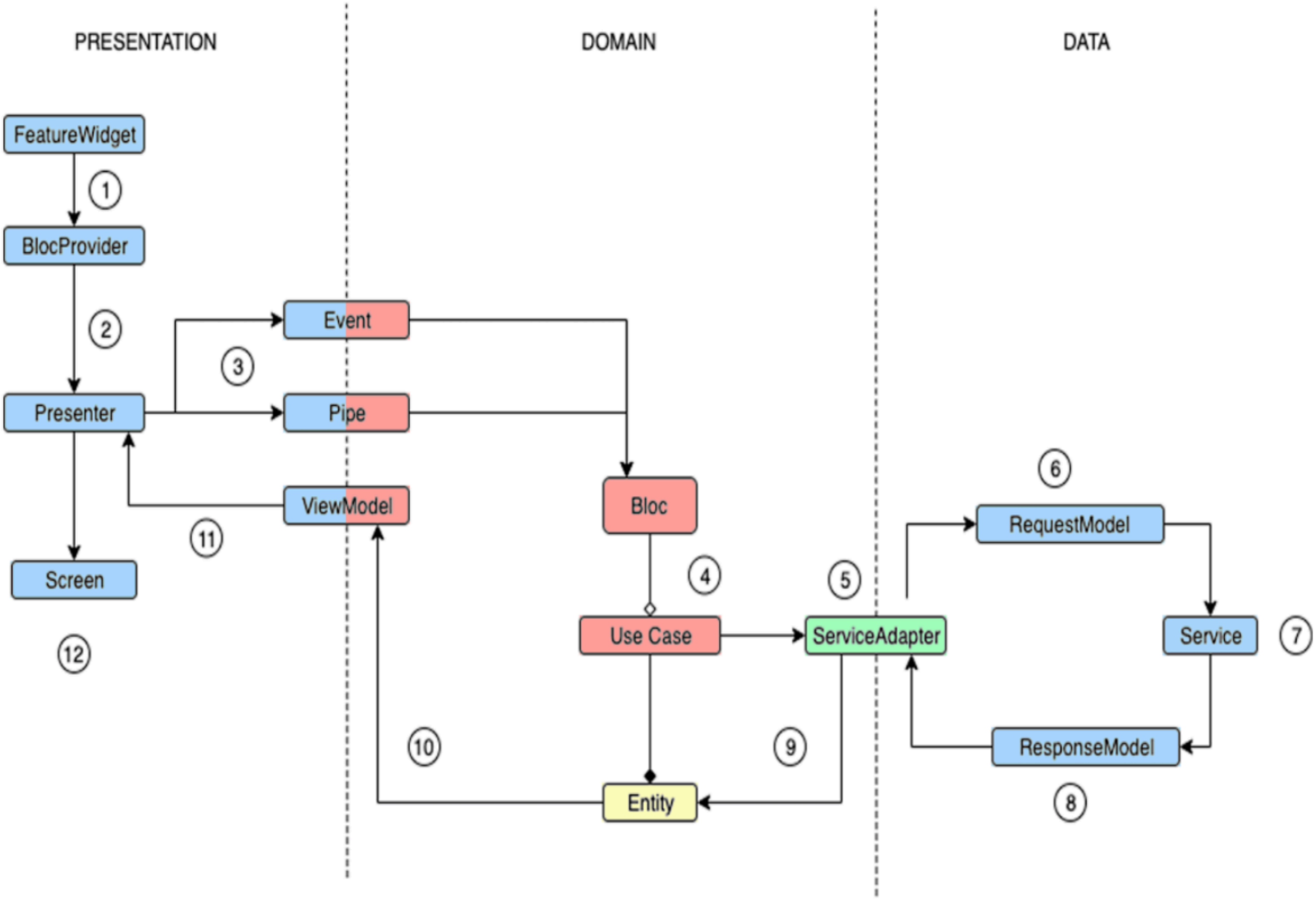
Presenter

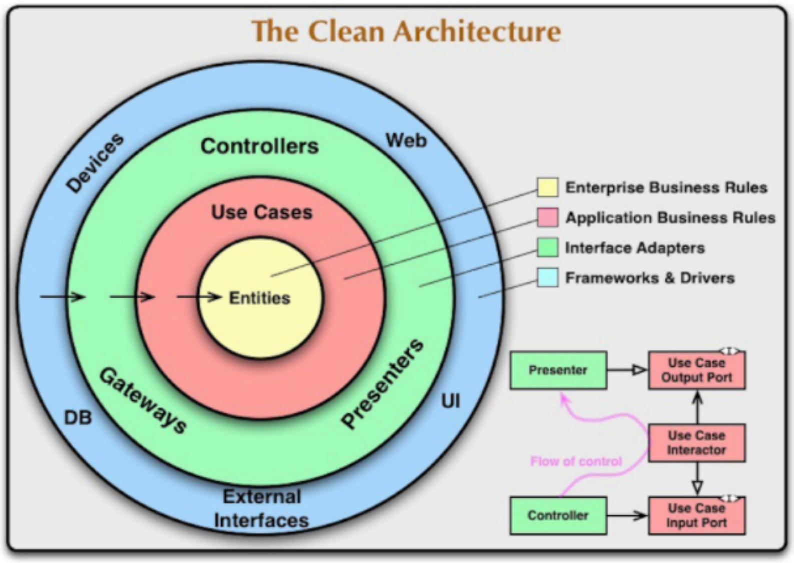
- Creates the "Screen" based off of:
    - BuildContext
    - BLoc
    - View Model

Screen

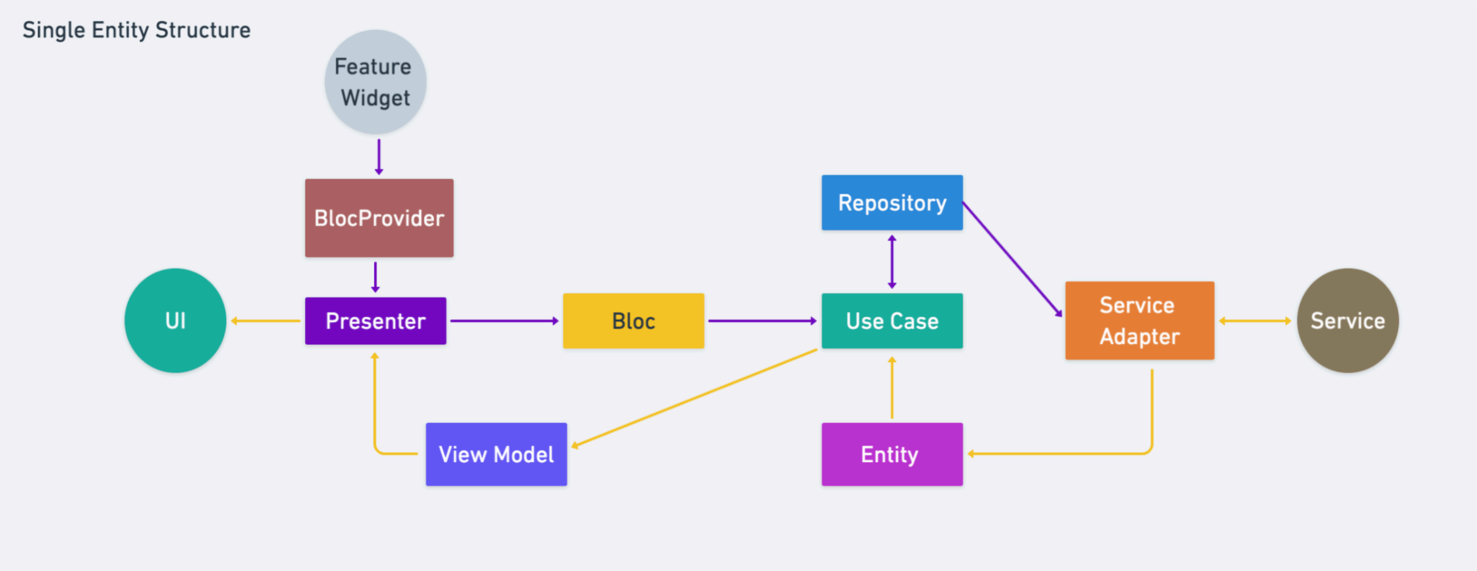
- The actual UI Widget being present for the specific Use Case
- This is where the View Model (or View Model List) will map the data to the UI

7. CLEAN Architecture Flow Chart in Flutter

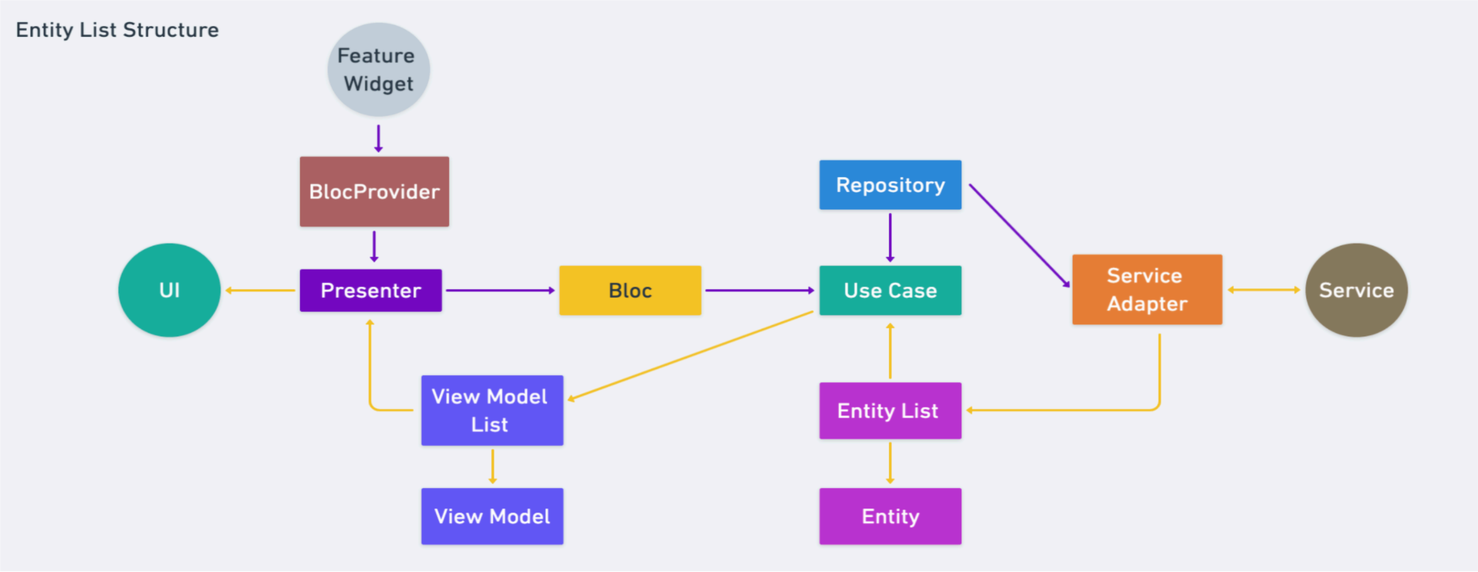




8. CLEAN Architecture - Single Entity Structure

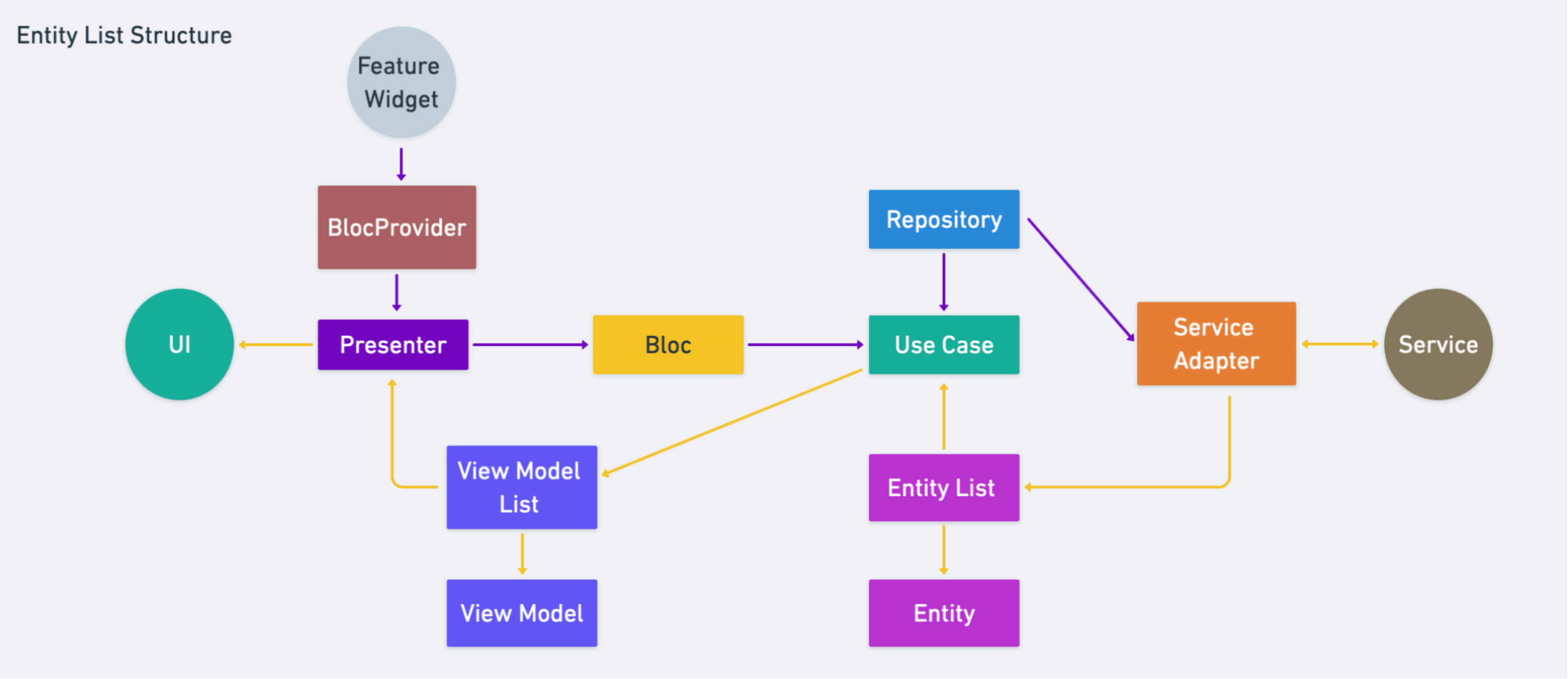


9. CLEAN Architecture - Entity List Structure



10. CLEAN Architecture Cash Accounts Example

- **Use Case**: Retreive Cash Accounts
  - Implementation Type: Entity List Structure



11. CLEAN Architecture - Repository

- Abstraction of Data/Dependency Layer
  - Creates usage of abstraction instead of concretes
- Maps Scope to Entities
  - Literally with a Map object collection
- Manages Scopes
  - Associates Scopes to Entity Types
- Executes Service Adapter Functionality
  - Via the Repository Scope
- Essentially, provides a "container" for any specific Entity to operate within the CLEAN architecture

Maps Scope to Entities

Repository contains Entities with data and Scope mapped to that Entity:

```
/// Creates a Map collection of Repository Scope and Entity Type.
Map<RepositoryScope, Entity> scopes = {};
```

To get a Scope for current Entity use method:

```
/// Checks if a Scope is associated with a specific Entity Type.
containsScope<E extends Entity>()
```

To get an Entity for current scope use method:

```
/// Returns the Entity associated with the Scope in the Map collection.
get<E extends Entity>(RepositoryScope scope)
```

Manages Scopes

To create a new scope for an existing Entity use method:

```
/// Creates a Scope if One is not already set, if set return existing scope.
create<E extends Entity>(E entity, Function(dynamic) subscription)
```

To update a scope for an existing Entity use method:

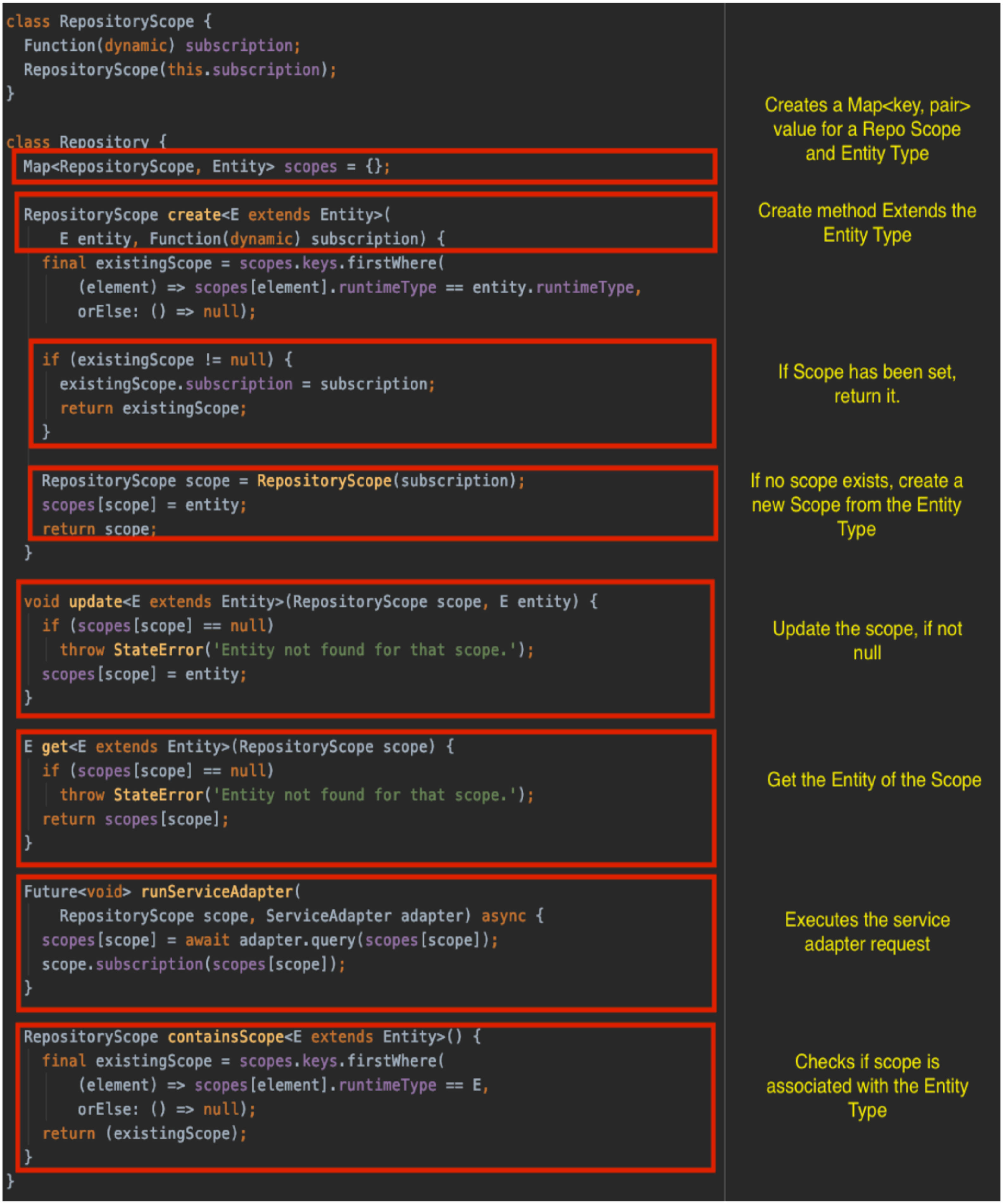
```
/// Updates an existing Scope to a scope provided to the method.
update<E extends Entity>(RepositoryScope scope, E entity)
```

Executes Service Adapter

To execute the Service Adapter use method:

```
/// Execute the provided Service Adapter with the associated Scope.
runServiceAdapter(RepositoryScope scope, ServiceAdapter adapter)
```

12. CLEAN Architecture - Repository Class



```
class RepositoryScope {
  Function(dynamic) subscription;
  RepositoryScope(this.subscription);
}

class Repository {
  Map<RepositoryScope, Entity> scopes = {};

  RepositoryScope create<E extends Entity>(
      E entity, Function(dynamic) subscription) {
    final existingScope = scopes.keys.firstWhere(
        (element) => scopes[element].runtimeType == entity.runtimeType,
        orElse: () => null);

    if (existingScope != null) {
      existingScope.subscription = subscription;
      return existingScope;
    }

    RepositoryScope scope = RepositoryScope(subscription);
    scopes[scope] = entity;
    return scope;
  }

  void update<E extends Entity>(RepositoryScope scope, E entity) {
    if (scopes[scope] == null)
      throw StateError('Entity not found for that scope.');
    scopes[scope] = entity;
  }

  E get<E extends Entity>(RepositoryScope scope) {
    if (scopes[scope] == null)
      throw StateError('Entity not found for that scope.');
    return scopes[scope];
  }

  Future<void> runServiceAdapter(
      RepositoryScope scope, ServiceAdapter adapter) async {
    scopes[scope] = await adapter.query(scopes[scope]);
    scope.subscription(scopes[scope]);
  }

  RepositoryScope containsScope<E extends Entity>() {
    final existingScope = scopes.keys.firstWhere(
        (element) => scopes[element].runtimeType == E,
        orElse: () => null);
    return (existingScope);
  }
}
```

Creates a Map<key, pair> value for a Repo Scope and Entity Type

Create method Extends the Entity Type

If Scope has been set, return it.

If no scope exists, create a new Scope from the Entity Type

Update the scope, if not null

Get the Entity of the Scope

Executes the service adapter request

Checks if scope is associated with the Entity Type

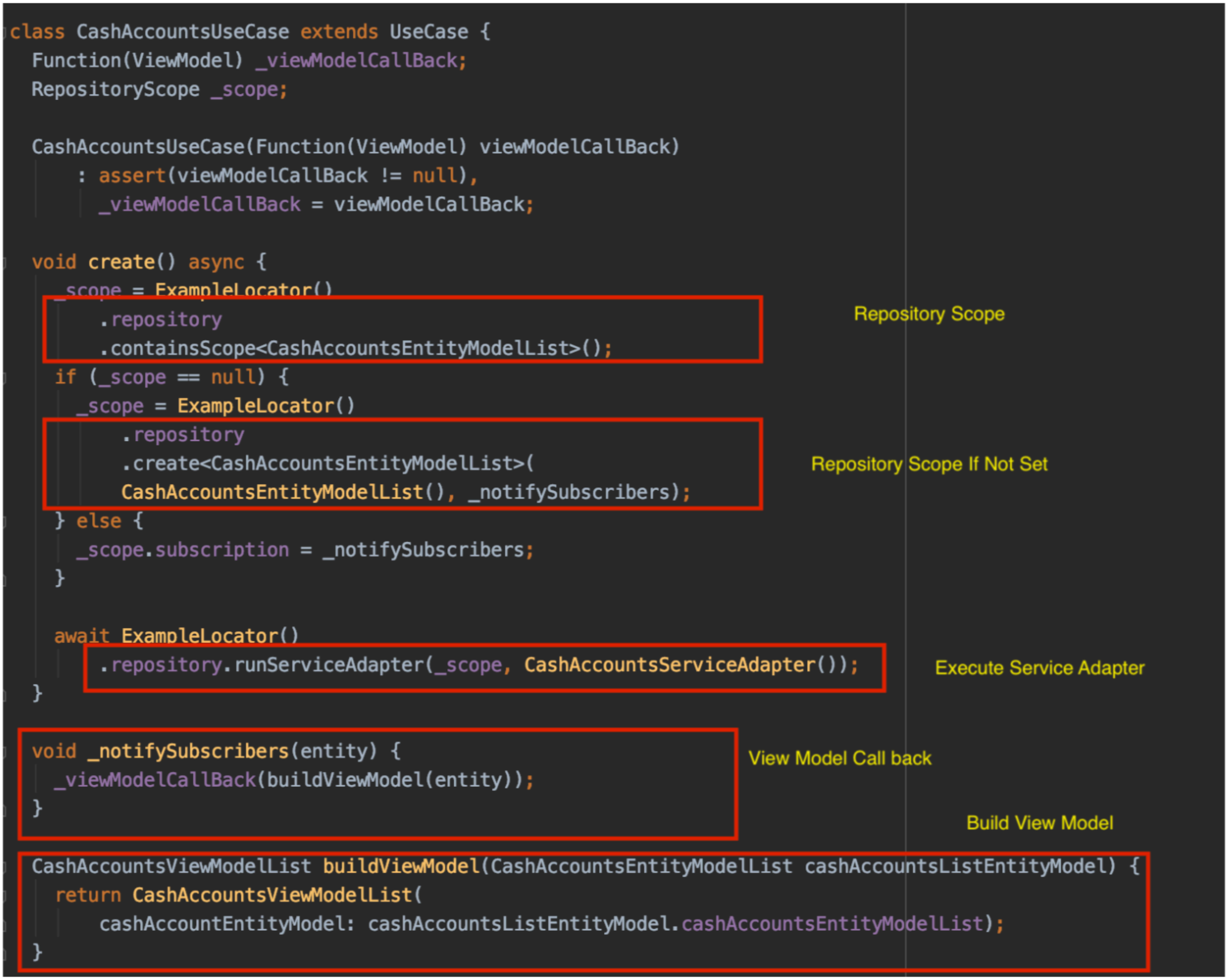
13. CLEAN Architecture - Use Case

- Contains the primary logic for the functionality of the feature being implemented
- Contains implementation of all the methods that should be executed in response to events
- Contains View Model Callback which communicates with View Model Pipe in the BLoc

What should be contained in a Use Case?

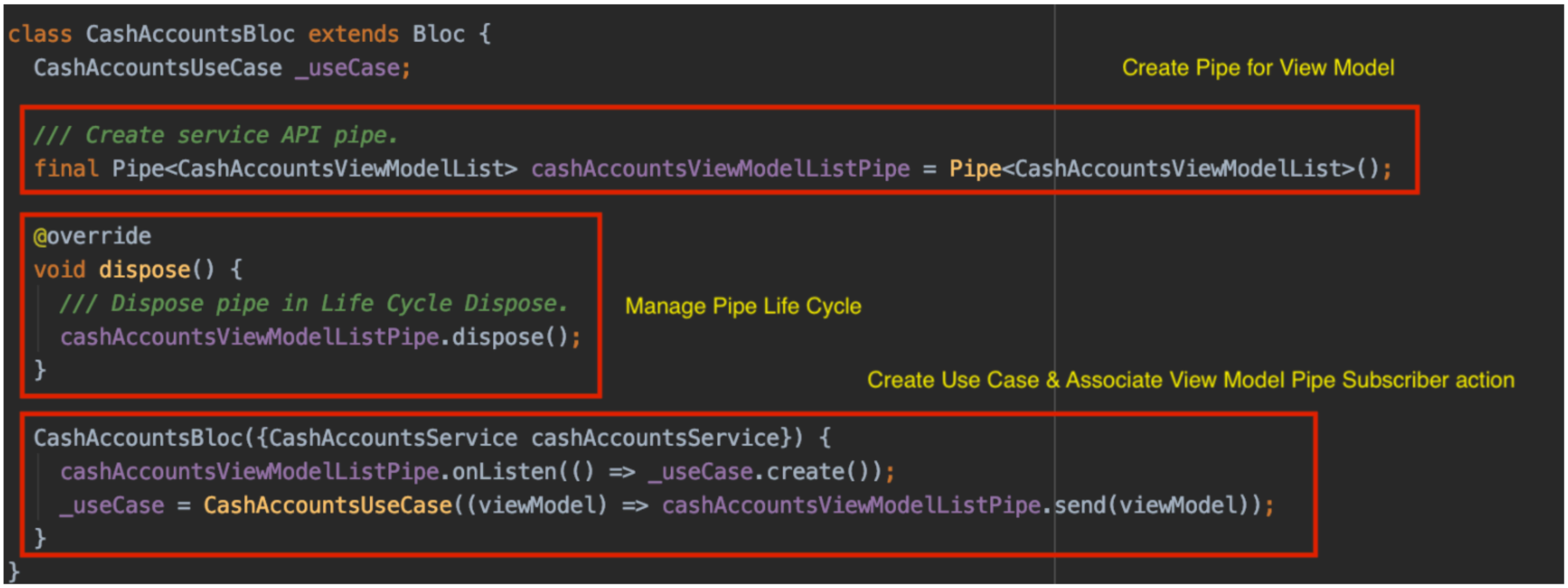
- A Repository associating a Scope with an Entity
- If a service is needed, the Use Case associates the Service Adapter to the Repository Scope
- If updating UI, building of a View Model and providing the View Model via a callback function through the Repository

14. Cash Accounts Example - Use Case



```
class CashAccountsUseCase extends UseCase {
  Function(ViewModel) _viewModelCallBack;
  RepositoryScope _scope;

  CashAccountsUseCase(Function(ViewModel) viewModelCallBack)
      : assert(viewModelCallBack != null),
        _viewModelCallBack = viewModelCallBack;

  void create() async {
    _scope = ExampleLocator()
        .repository
        .containsScope<CashAccountsEntityModelList>();
    if (_scope == null) {
      _scope = ExampleLocator()
          .repository
          .create<CashAccountsEntityModelList>(
          CashAccountsEntityModelList(), _notifySubscribers);
    } else {
      _scope.subscription = _notifySubscribers;
    }

    await ExampleLocator()
        .repository.runServiceAdapter(_scope, CashAccountsServiceAdapter());
  }

  void _notifySubscribers(entity) {
    _viewModelCallBack(buildViewModel(entity));
  }

  CashAccountsViewModelList buildViewModel(CashAccountsEntityModelList cashAccountsListEntityModel) {
    return CashAccountsViewModelList(
        cashAccountEntityModel: cashAccountsListEntityModel.cashAccountsEntityModelList);
  }
```

15. Cash Accounts Example - BLoc



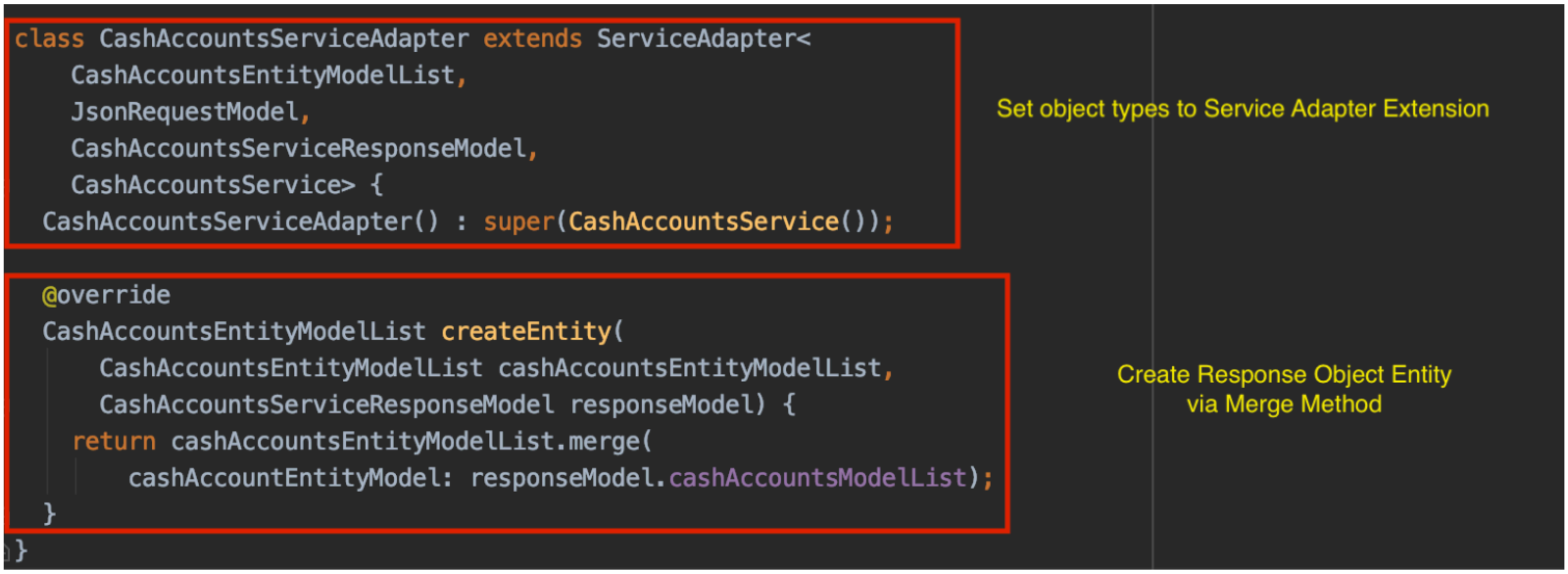
```
class CashAccountsBloc extends Bloc {
  CashAccountsUseCase _useCase;

  /// Create service API pipe.
  final Pipe<CashAccountsViewModelList> cashAccountsViewModelListPipe = Pipe<CashAccountsViewModelList>();

  @override
  void dispose() {
    /// Dispose pipe in Life Cycle Dispose.
    cashAccountsViewModelListPipe.dispose();
  }

  CashAccountsBloc({CashAccountsService cashAccountsService}) {
    cashAccountsViewModelListPipe.onListen(() => _useCase.create());
    _useCase = CashAccountsUseCase((viewModel) => cashAccountsViewModelListPipe.send(viewModel));
  }
}
```

Create Pipe for View Model

Manage Pipe Life Cycle

Create Use Case & Associate View Model Pipe Subscriber action

16. Cash Accounts - Service Adapter

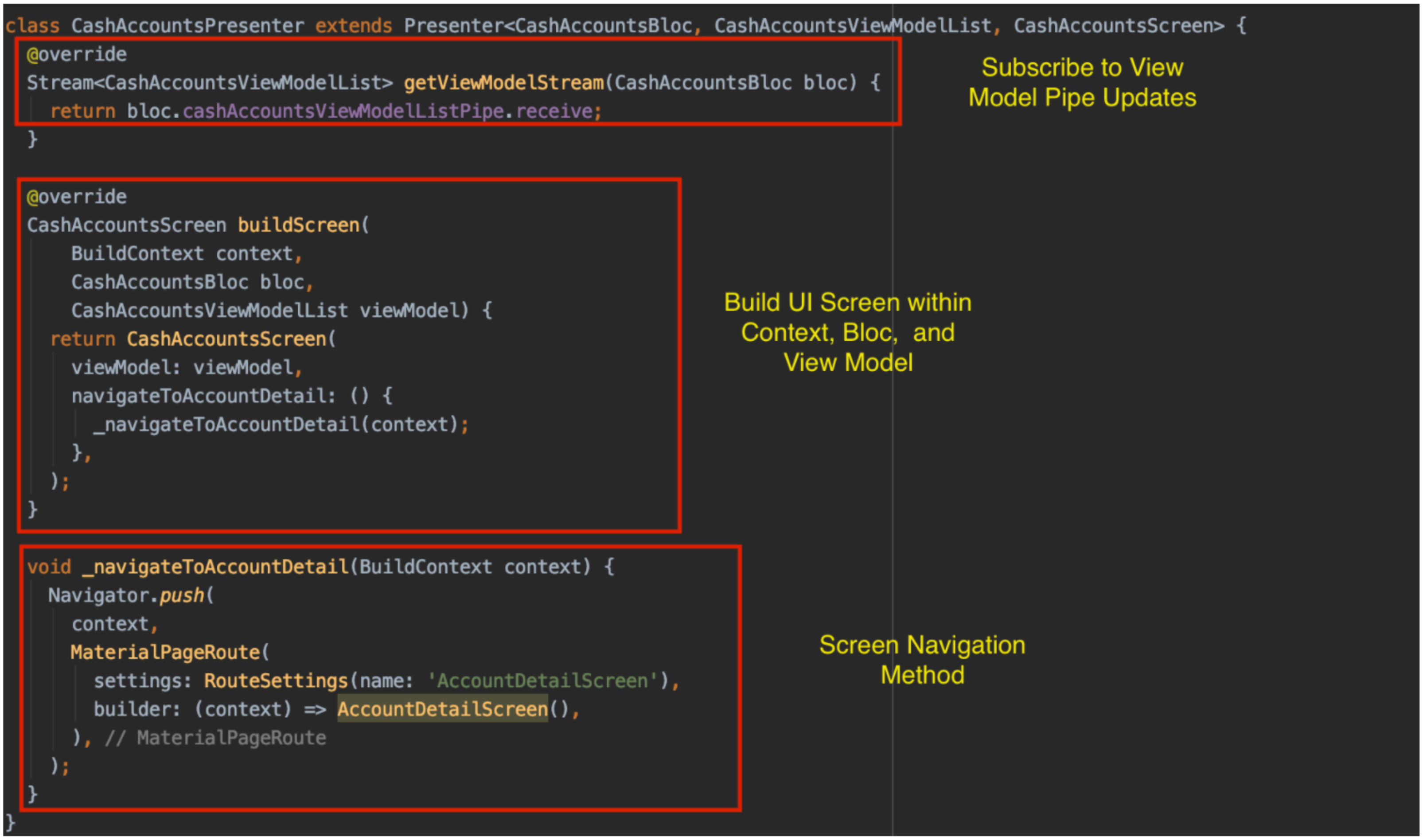


```
class CashAccountsServiceAdapter extends ServiceAdapter<
    CashAccountsEntityModelList,
    JsonRequestModel,
    CashAccountsServiceResponseModel,
    CashAccountsService> {
  CashAccountsServiceAdapter() : super(CashAccountsService());

  @override
  CashAccountsEntityModelList createEntity(
      CashAccountsEntityModelList cashAccountsEntityModelList,
      CashAccountsServiceResponseModel responseModel) {
    return cashAccountsEntityModelList.merge(
        cashAccountEntityModel: responseModel.cashAccountsModelList);
  }
}
```

Set object types to Service Adapter Extension

Create Response Object Entity
via Merge Method

17. Cash Accounts - Presenter



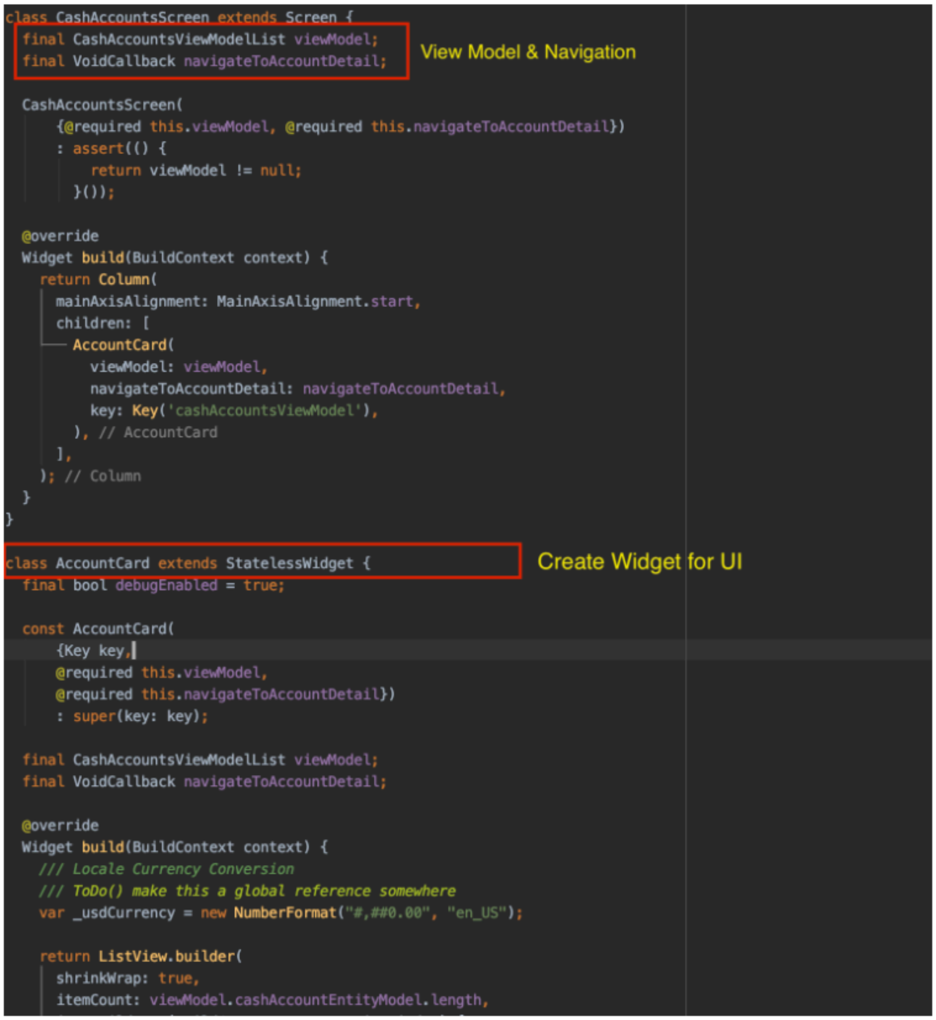
```
class CashAccountsPresenter extends Presenter<CashAccountsBloc, CashAccountsViewModelList, CashAccountsScreen> {
  @override
  Stream<CashAccountsViewModelList> getViewModelStream(CashAccountsBloc bloc) {
    return bloc.cashAccountsViewModelListPipe.receive;
  }

  @override
  CashAccountsScreen buildScreen(
      BuildContext context,
      CashAccountsBloc bloc,
      CashAccountsViewModelList viewModel) {
    return CashAccountsScreen(
      viewModel: viewModel,
      navigateToAccountDetail: () {
        _navigateToAccountDetail(context);
      },
    );
  }

  void _navigateToAccountDetail(BuildContext context) {
    Navigator.push(
      context,
      MaterialPageRoute(
        settings: RouteSettings(name: 'AccountDetailScreen'),
        builder: (context) => AccountDetailScreen(),
      ), // MaterialPageRoute
    );
  }
}
```

Subscribe to View Model Pipe Updates

Build UI Screen within Context, Bloc, and View Model

Screen Navigation Method

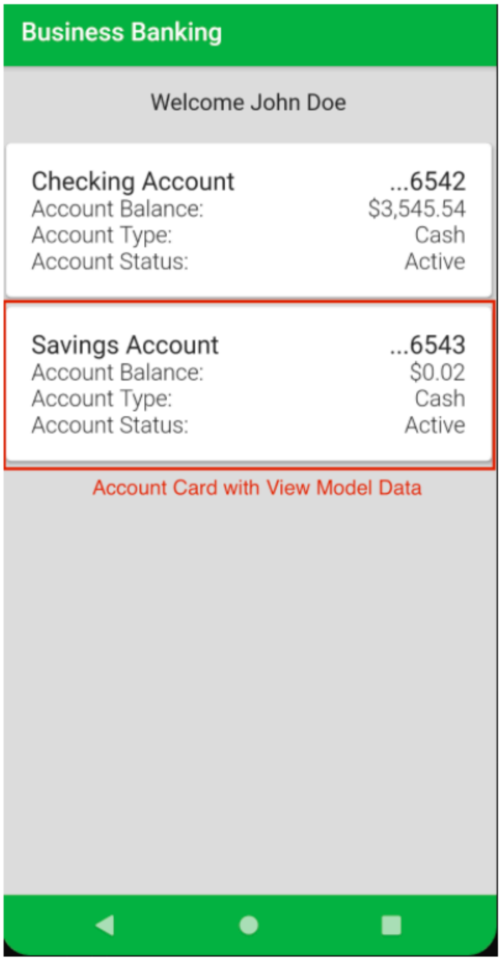
18. Cash Accounts - Screen -> UI



```
class CashAccountsScreen extends Screen {
  final CashAccountsViewModelList viewModel;
  final VoidCallback navigateToAccountDetail;

  CashAccountsScreen(
      {@required this.viewModel, @required this.navigateToAccountDetail})
      : assert(() {
          return viewModel != null;
        }());

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.start,
      children: [
        AccountCard(
          viewModel: viewModel,
          navigateToAccountDetail: navigateToAccountDetail,
          key: Key('cashAccountsViewModel'),
        ), // AccountCard
      ],
    ); // Column
  }
}

class AccountCard extends StatelessWidget {
  final bool debugEnabled = true;

  const AccountCard(
      {Key key,
      @required this.viewModel,
      @required this.navigateToAccountDetail})
      : super(key: key);

  final CashAccountsViewModelList viewModel;
  final VoidCallback navigateToAccountDetail;

  @override
  Widget build(BuildContext context) {
    /// Locale Currency Conversion
    /// ToDo() make this a global reference somewhere
    var _usdCurrency = new NumberFormat("#,##0.00", "en_US");

    return ListView.builder(
      shrinkWrap: true,
      itemCount: viewModel.cashAccountEntityModel.length,
```

View Model & Navigation

Create Widget for UI



Business Banking

Welcome John Doe

Checking Account	...6542
Account Balance:	$3,545.54
Account Type:	Cash
Account Status:	Active

Savings Account	...6543
Account Balance:	$0.02
Account Type:	Cash
Account Status:	Active

Account Card with View Model Data

19. References

AndroidPub - Milhay Nagy

Uncle Bob - Clean Coder

geeksforgeeks - SOLID Principles