TDD Demonstration

1. Have a new functionality to add

In this case, we want to add a new functionality to our app which is to fetch the expenses from our server.

2. Implement a test that validates the functionality

We write a test that makes the request needed to the server and expects a certain result in order to pass the test.

```dart
test(
  "fetchExpenses. Returns a list of Expenses if the http call does succesfully",
  () async {
    final String url = 'http://10.0.2.2:3000';
    final String db = 'db';
    final client = MockClient();
    when(
      client.get(Uri.parse('$url/$db')),
    ).thenAnswer(
      (_) async => http.Response(
        '''
        {"expenses": [
          {"id": 0,"title": "initial balance", "amount": 120},
          {"id": 1,"title": "groceries", "amount": -24.3}
          ],
          "savings": []
        }''',
        200,
      ),
    );
    final apiClient = APIClient();
    expect(
      await apiClient.fetchExpenses(client: client),
      isA<List<ExpenseEntry>>(),
    );
  },
);
```

3. Check that our code does not pass the test

This is a trivial step which is not needed, but in order to demonstrate the procedure, we can run the test and check that obviously our code does not meet the requirements since the functionality has not been implemented yet.

```
FormatException: Unexpected end of input (at character 1)

^

dart:convert                                        JsonCodec.decode
new ExpensesList.fromJson
APIClient.fetchExpenses
main.<fn>
main.<fn>
2
✕ fetchExpenses. Returns a list of Expenses if the http call does succesfully
```

4. Write the new functionality

In our case, a function called fetchExpenses is needed to achieve our goal, in which we get from the server the required information.

```dart
Future<List<ExpenseEntry>> fetchExpenses({Client? client}) async {
  if (client == null) client = Client();
  final response = await client.get(Uri.parse('$url/$db'));

  if (response.statusCode == 200) {
    return ExpensesList.fromJson(response.body).expenses;
  } else {
    throw Exception(
      'Failed to load Expenses. Check the server is running',
    );
  }
}
```

5. Run the test

When we think our new function is completed and correct, we run the test in order to validate our work is properly done

```
✓ fetchExpenses. Returns a list of Expenses if the http call does succesfully
Exited
```

6. Iterate

In case the test fails, we repeat steps 4 and 5 until our test succeeds. We have to keep an eye on the test code itself, and double check that it is well implemented.