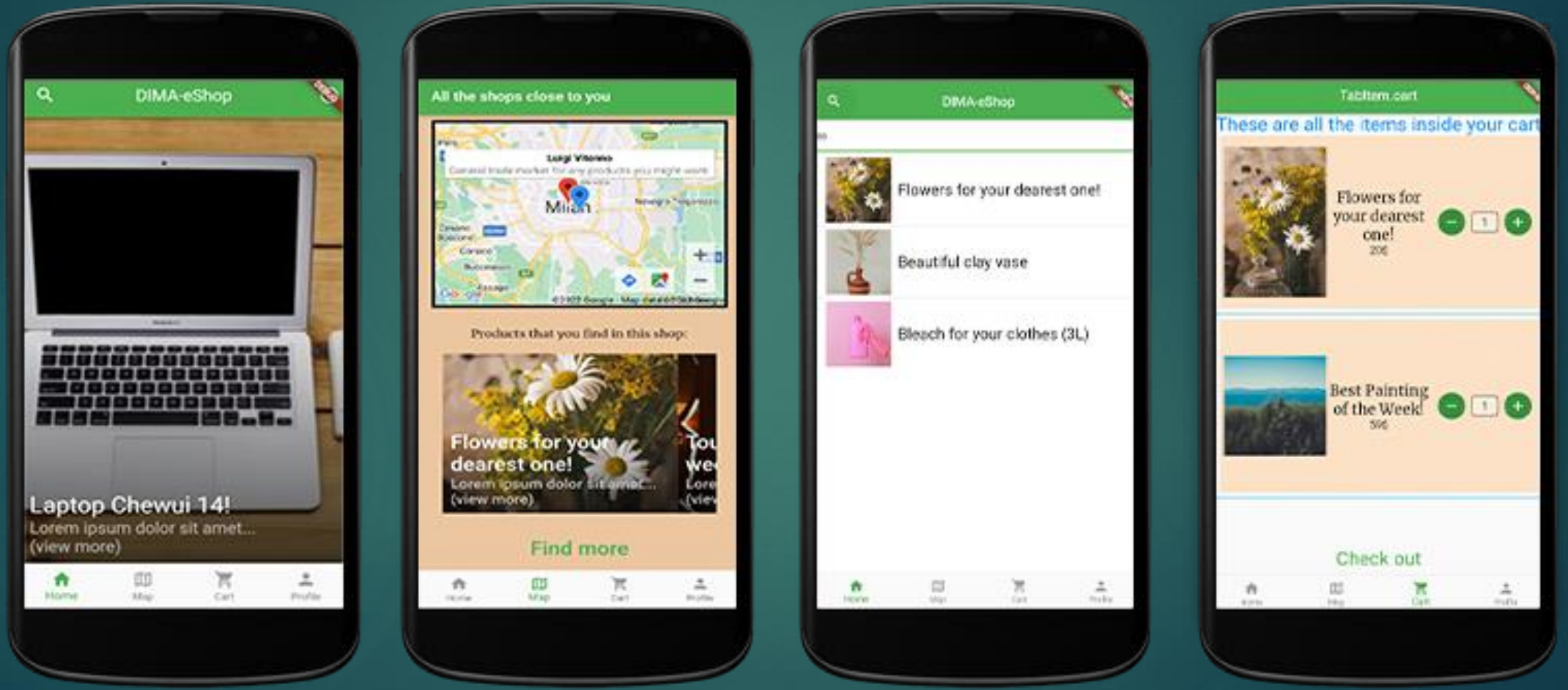
DIMA eShop

ETION PINARI E PIETRO MORONI

Purpose of the Application

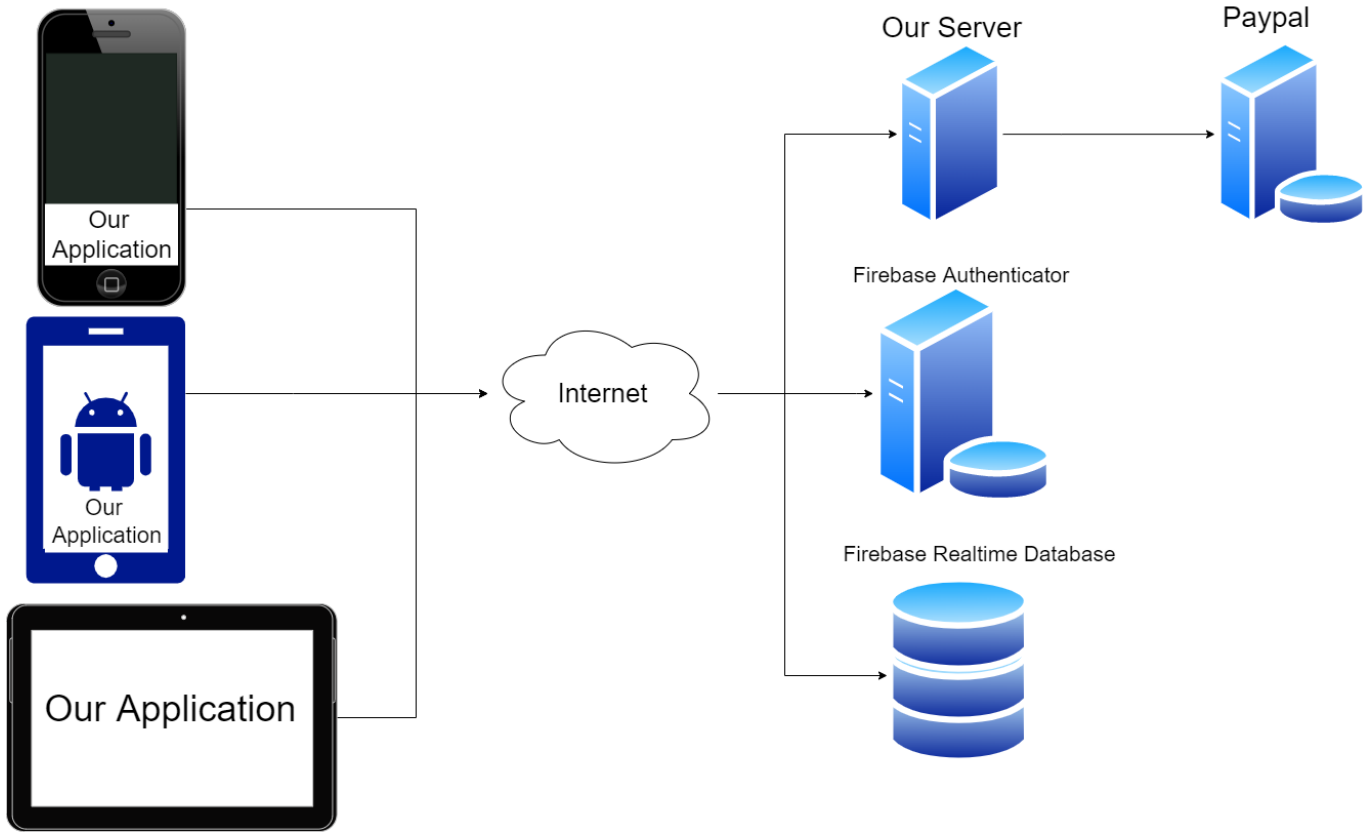
- User-friendly interface which connects clients and the vendors allowing them to respectively buy and sell products.
- It works as an intuitive interface for clients to browse and buy products from our shop.
- Both purchase and keep track of products you like by saving them as "favorites" for later.
- The application is also designed to work for registered and unregistered users, providing most functionalities to both categories.

Interface Design



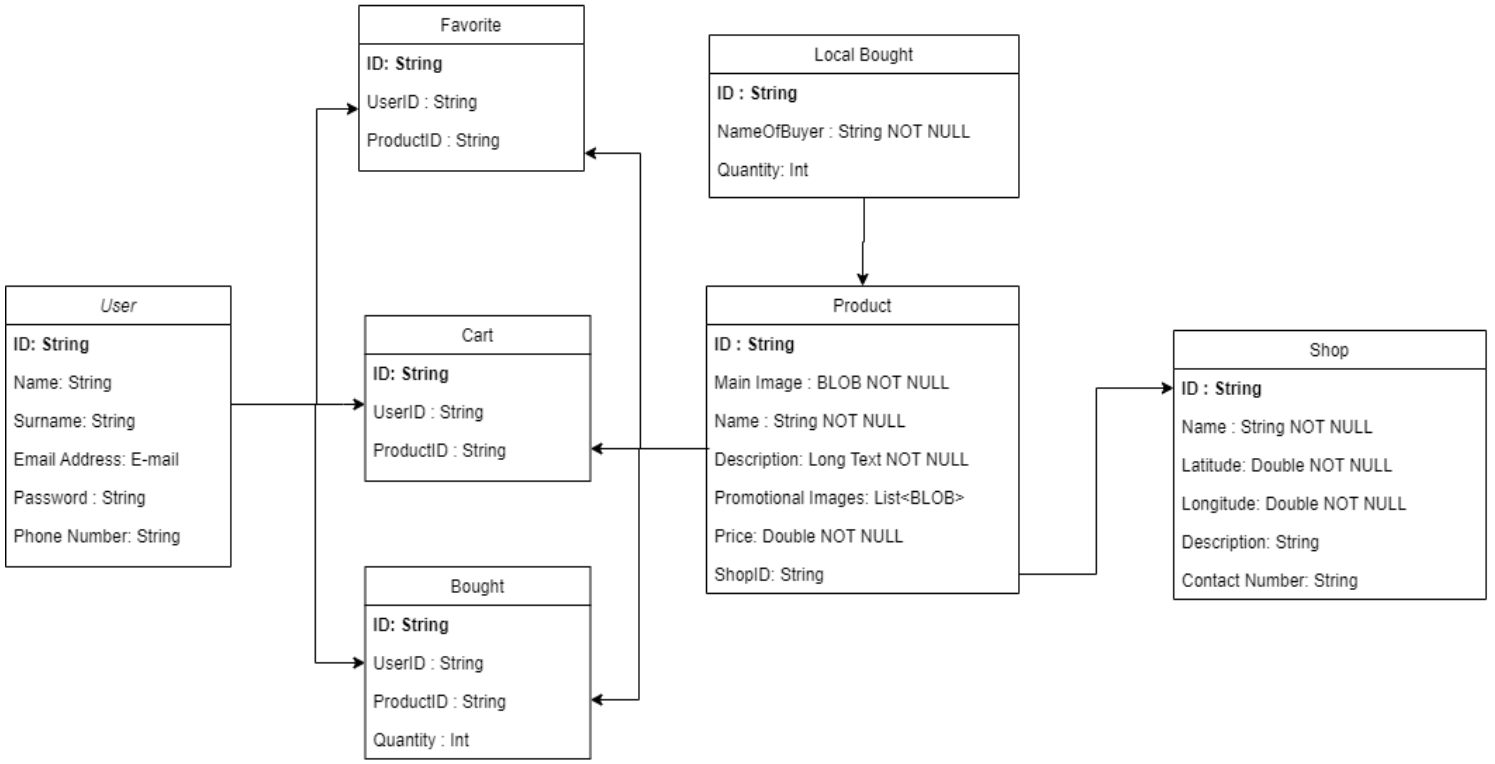
System Architecture

- Cross platform mobile application
  - Using the flutter dart framework
- Cloud backend



Database Design

- We identified the three main entities
  - Users
  - Products
  - Shops
- We use a special entity to keep track of unauthenticated users' orders in the database



External Services and APIs

- Firebase Authentication
  - Sign up
  - Sign in
- Firebase Realtime Database
  - Shop and product data
  - User data
- Google Maps Services
  - Nearest shops
- Payment API
- Flutter packages from pub.dev

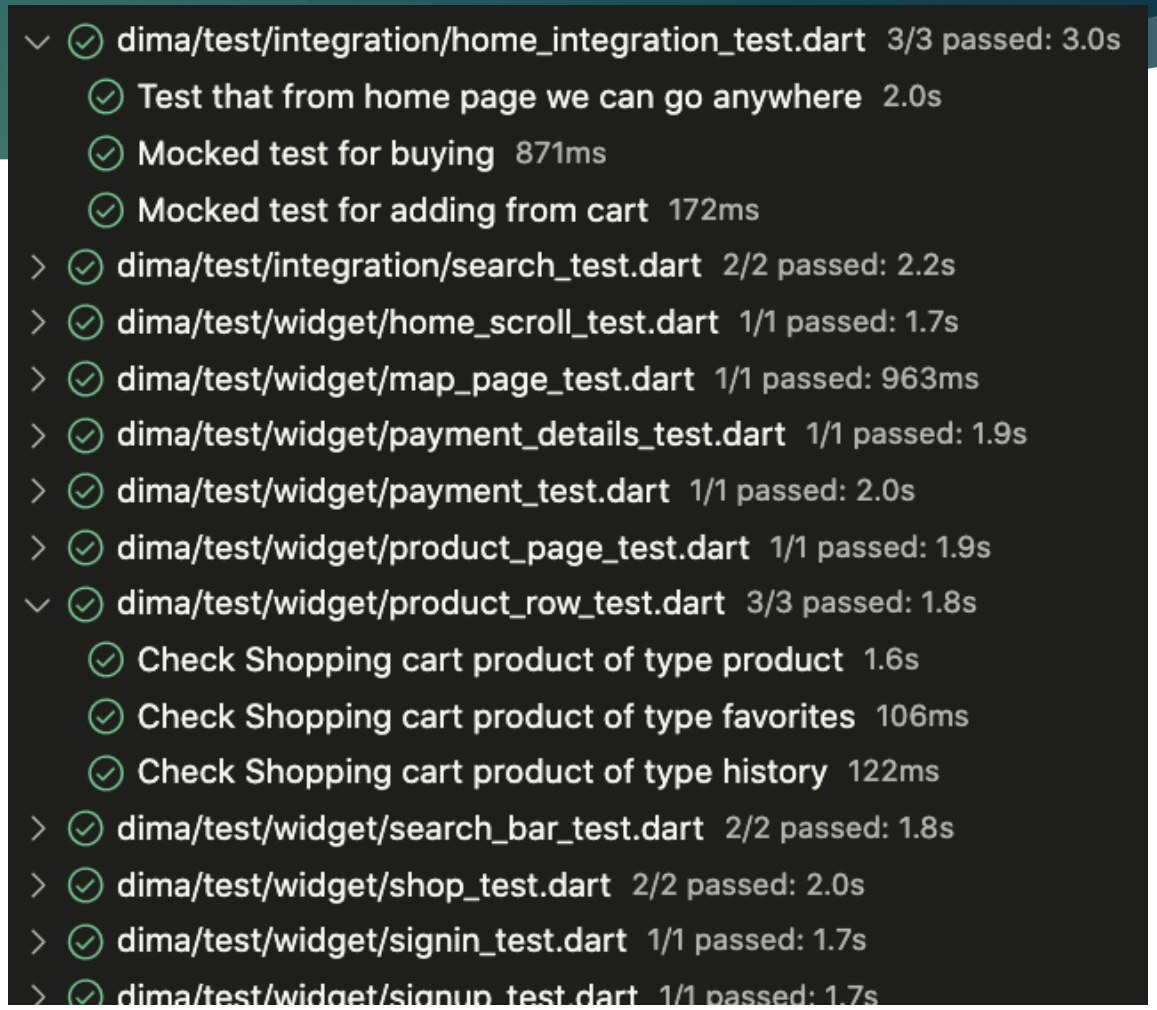
Application Use Cases

- Browsing the catalogue
  - Scrolling through the home page
  - Searching the catalogue through the search bar
- Purchasing a product
  - Adding products to the cart
  - Modifying the cart's content
  - Checking out
- Looking up the nearest shops in the map

Testing Campaign

- Widget Tests
- Integration Tests

- Tests are mocked with hard-coded data to avoid the logic being strictly dependent on the external services



Widget Tests

- We designed tests for widgets that are then used inside larger-scope pages of the application
- Widgets are tested by drawing them with mock data and asserting that each of their sub-components is rendered correctly and displays the correct information

```
testWidgets('Check that the product page is correctly instantiated',
    (WidgetTester tester) async {
  Product product = Product(
    id: '0', name: 'Laptop Chewui 14!',
    price: '259', imageLink: 'https://picsum.photos/250?image=9',
    image: Image.asset('images/twoMenShakingHands.jpg',scale: 0.1),
  ); // Product
  Widget _widget = ChangeNotifierProvider(
      create: (context) => ApplicationState(initializer: () {
            return;
          }), // ApplicationState
      builder: (context, _) => MaterialApp(
          home: SizedBox.expand(
              child: Container(color: Colors.black, child: ProductFromID(
                  product: product, productId: product.id, ))))));
  await mockNetworkImagesFor(() async => await tester.pumpWidget(_widget));
  expect(find.byIcon(Icons.favorite, skipOffstage: false), findsOneWidget);
  expect(find.byIcon(Icons.add_shopping_cart_rounded, skipOffstage: false),
      findsOneWidget);
  expect(find.byType(Image, skipOffstage: false), findsOneWidget);
  expect(find.textContaining('Buy', skipOffstage: false), findsOneWidget);
});
```

Integration Tests

- Integration tests are designed around an expected flow of actions that simulates an expected interaction of a user with the real application.
- The interactions are also designed around the mock data since we decided to decouple database communication from the simulation.

```
testWidgets('Searching for a product that exists in the [redacted]',
    (WidgetTester tester) async {
  DatabaseManager.updateProductTester();
  MyApp _myApp = const MyApp();
  await tester.runAsync(() async {
    await tester.pumpWidget(ChangeNotifierProvider(
      create: (context) => ApplicationState(initializer: () {
        return;
      }),
      builder: (context, _) => _myApp,
    ));
  });
  await tester.tap(find.byIcon(Icons.search_rounded));
  await tester.pump(const Duration(seconds: 1));

  TextField? widget;
  for (final element
      in find.byType(TextField, skipOffstage: false).evaluate()) {
    widget = element.widget as TextField;
    await tester.enterText(find.byWidget(widget), "non existing product");
  }
  await tester.pump(const Duration(seconds: 1));
  expect(find.textContaining("Oops", skipOffstage: false), findsOneWidget);

  widget!.controller!.text = "";

  for (final element
      in find.byType(TextField, skipOffstage: false).evaluate()) {
    widget = element.widget as TextField;
    await tester.enterText(find.byWidget(widget), "Chu");
  }
  await tester.pump(const Duration(seconds: 1));
  expect(find.textContaining("Chuwi", skipOffstage: false), findsOneWidget);
});
```