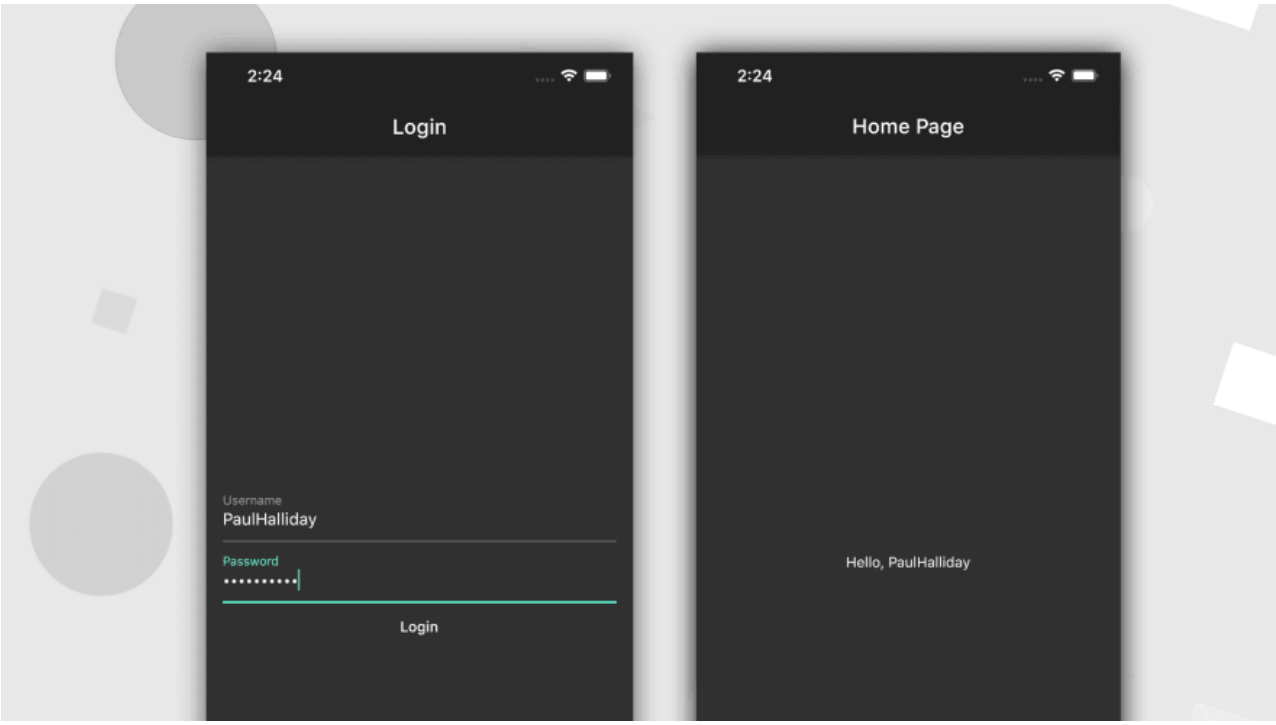
How to Use ProxyProvider with Flutter

dev.to/paulhalliday/how-to-use-proxyprovider-with-flutter-3ifo

In this article we're going to look at how to use `ProxyProvider` to inject data into *other* providers. This is useful if we're wanting to inject an auth token or other piece of dynamic data into another `Provider` at some point in the future.

The `ProxyProvider` has an `update` method which is called whenever one of its dependencies has updated. We'll see this in action in our example application by passing a `GreetingService` a `UserService` which is able to provide the current user.

Here's an example:



Project Setup

Let's create a new Flutter project in the terminal:

```
# New Flutter project
$ flutter create proxyprovider

# Open in VS Code
$ cd proxyprovider && code .
```

We'll then need to add the `provider` dependency to our `pubspec.yaml`:

```
dependencies:
  flutter:
    sdk: flutter

  provider: ^4.0.5
```

That's all the packages we need. You can now open the project up on the platform of your choice.

Login

The first thing we'll do is create our `LoginForm`. In our example we're using it to gather a `username` to be greeted on the `HomePage`:

```
///lib/presentation/widgets/login_form.dart
import 'package:flutter/material.dart';
import 'package:proxyprovider/domain/entities/user.dart';

class LoginForm extends StatefulWidget {
  final Function(User) onFormSaved;

  const LoginForm({Key key, @required this.onFormSaved}) : super(key: key);

  @override
  _LoginFormState createState() => _LoginFormState();
}

class _LoginFormState extends State<LoginForm> {
  bool _autoValidate;

  GlobalKey<FormState> _formKey;

  TextEditingController _usernameTextEditingController;
  TextEditingController _passwordTextEditingController;

  @override
  void initState() {
    super.initState();

    _autoValidate = false;

    _formKey = GlobalKey<FormState>();

    _usernameTextEditingController = TextEditingController();
    _passwordTextEditingController = TextEditingController();
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      autovalidate: _autoValidate,
      child: Column(
        children: [
          TextFormField(
              controller: _usernameTextEditingController,
              decoration: InputDecoration(labelText: "Username"),
              validator: (String value) =>
                  _validateFormField(value, "Username")),
          TextFormField(
```

```
            controller: _passwordTextEditingController,
            obscureText: true,
            decoration: InputDecoration(labelText: "Password"),
            validator: (String value) =>
                _validateFormField(value, "Password")),
          FlatButton(
            onPressed: _onLoginPressed,
            child: Text("Login"),
          )
        ],
      ),
    );
  }

  _onLoginPressed() {
    setState(() {
      _autoValidate = true;
    });

    if (_formKey.currentState.validate()) {
      widget.onFormSaved(
        User(
          username: _usernameTextEditingController.text,
        ),
      );
    }
  }

  String _validateFormField(String value, String fieldName) {
    if (value.isEmpty) {
      return "$fieldName cannot be empty.";
    }

    return null;
  }

  @override
  void dispose() {
    _usernameTextEditingController.dispose();
    _passwordTextEditingController.dispose();

    super.dispose();
  }
}
```

Our `User` entity will be extremely bare. It'll have one property - `username` :

```
///lib/domain/entities/user.dart
import 'package:flutter/foundation.dart';

class User {
  final String username;

  User({@required this.username});
}
```

We can then create our `LoginPage` which will use the `LoginForm`:

```
///lib/presentation/pages/login_page.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:proxyprovider/application/services/user_service.dart';
import 'package:proxyprovider/domain/entities/user.dart';
import 'package:proxyprovider/presentation/pages/home_page.dart';
import 'package:proxyprovider/presentation/widgets/login_form.dart';

class LoginPage extends StatelessWidget {
  static Route<dynamic> route() => MaterialPageRoute(
        builder: (BuildContext context) => LoginPage(),
      );

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Login"),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            LoginForm(
              onFormSaved: (User user) => _onFormSaved(context, user),
            ),
          ],
        ),
      ),
    );
  }

  _onFormSaved(BuildContext context, User user) {
    Provider.of<UserService>(context, listen: false).setUser(user);
    Navigator.of(context).pushReplacement(HomePage.route());
  }
}
```

A review of what we've got so far:

1. We've got the ability to capture a `User` object from our `LoginForm`
2. Our `LoginPage` shows the form, and when the `onFormSaved` callback is fired we're calling `UserService.setUser(user)` and navigating to the `HomePage`.

Services

We haven't created the `UserService` or the `HomePage` to support this use case. Let's do that now:

UserService

Our `UserService` will be a simple class that is able to `set` and `get` the current `user`:

```
///lib/application/services/user_service.dart
import 'package:proxyprovider/domain/entities/user.dart';

class UserService {
  User _user;
  User get user => _user;

  setUser(User user) {
    _user = user;
  }
}
```

We can update `main.dart` to add our `UserService` as a `Provider`:

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:proxyprovider/application/services/user_service.dart';
import 'package:proxyprovider/presentation/pages/login_page.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        Provider(
          create: (_) => UserService(),
        ),
      ],
      child: MaterialApp(
        title: 'ProxyProvider',
        debugShowCheckedModeBanner: false,
        theme: ThemeData(
          brightness: Brightness.dark,
          visualDensity: VisualDensity.adaptivePlatformDensity,
        ),
        home: LoginPage(),
      ),
    );
  }
}
```

GreetingService

Our `GreetingService` will take a `UserService` in as a parameter and we'll use `ProxyProvider` to inject this with the latest value from our `UserService`:

```
///lib/application/services/greeting_service.dart
import 'package:flutter/foundation.dart';
import 'package:proxyprovider/application/services/user_service.dart';

class GreetingService {
  GreetingService({@required UserService userService})
      : _userService = userService;

  UserService _userService;

  String get greeting => "Hello, ${_userService.user.username}";
}
```

ProxyProvider

Now that we've got both our services, we can update our `providers` list inside of `main.dart` to return the `GreetingService` as a `Provider` with the latest value from `UserService`:

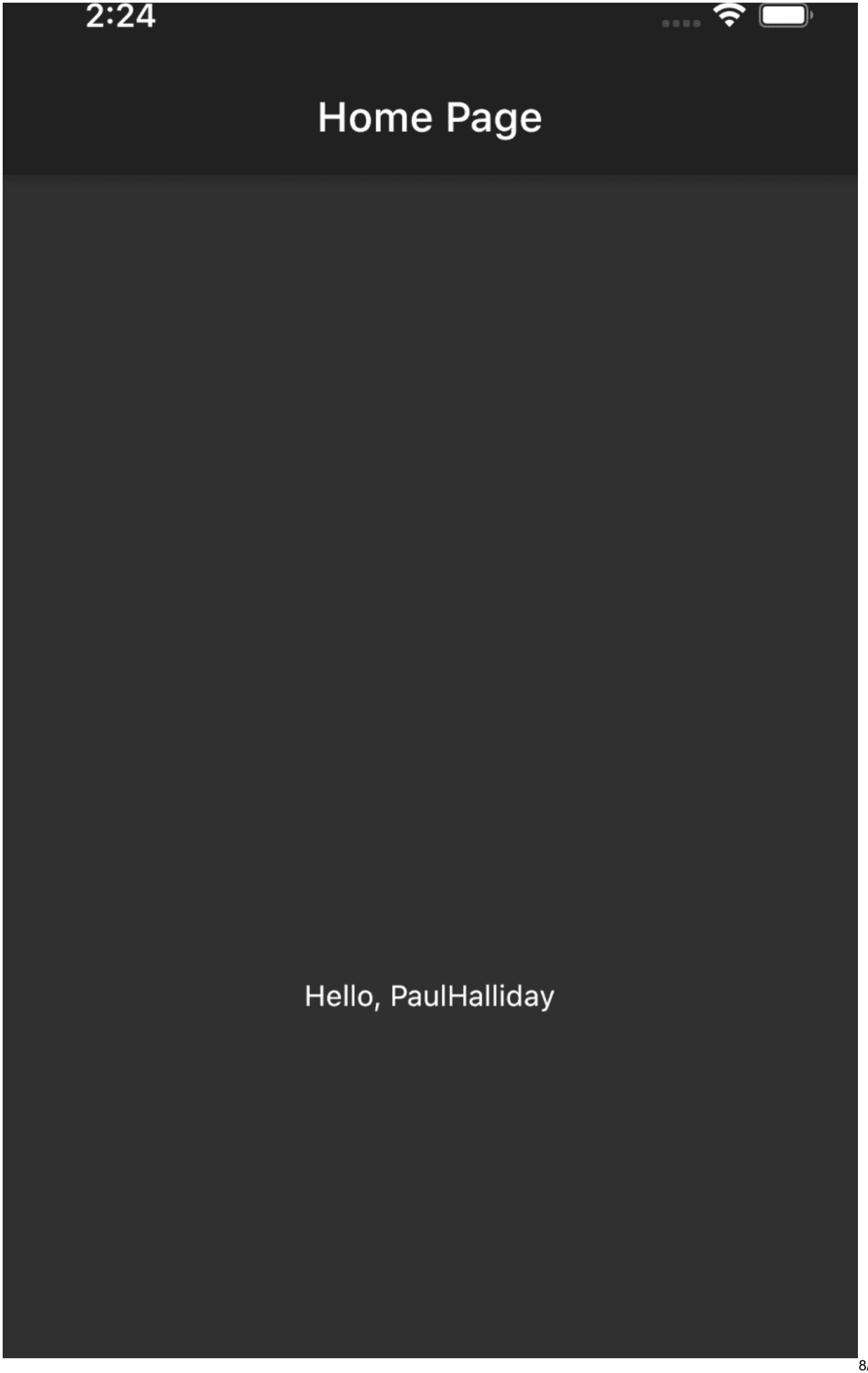
```
MultiProvider(
  providers: [
    Provider(
      create: (_) => UserService(),
    ),
    ProxyProvider<UserService, GreetingService>(
      update: (BuildContext context, UserService userService,
            GreetingService greetingService) =>
          GreetingService(userService: userService),
    ),
  ],
 //
```

This means that we're now able to access the value of `GreetingService` as a `Provider` and we can be assured that any time our `UserService` updates, our `GreetingService` will be updated to match.

We can see this in our `HomePage`:

```
///lib/presentation/pages/home_page.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:proxyprovider/application/services/greeting_service.dart';

class HomePage extends StatelessWidget {
  static Route<dynamic> route() => MaterialPageRoute(
        builder: (BuildContext context) => HomePage(),
      );

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Home Page"),
      ),
      body: Center(
        child: Text(
          Provider.of<GreetingService>(context).greeting,
        ),
      ),
    );
  }
}
```

Whatever we typed inside of our `LoginForm` as a `username` will now appear in the `Center` of our `HomePage`:


2:24
Home Page
Hello, PaulHalliday

Multiple Injections

What if we have more than one item that we want to inject as a `ProxyProvider`? As of now we're only injecting the `UserService`, but there may be times when we want to add more than one object.

For this we have to do the same as before, but use `ProxyProvider2`, `ProxyProvider2`, `ProxyProvider3`, and so on.

Here's an example of how this may look with the use of `ProxyProvider2`:

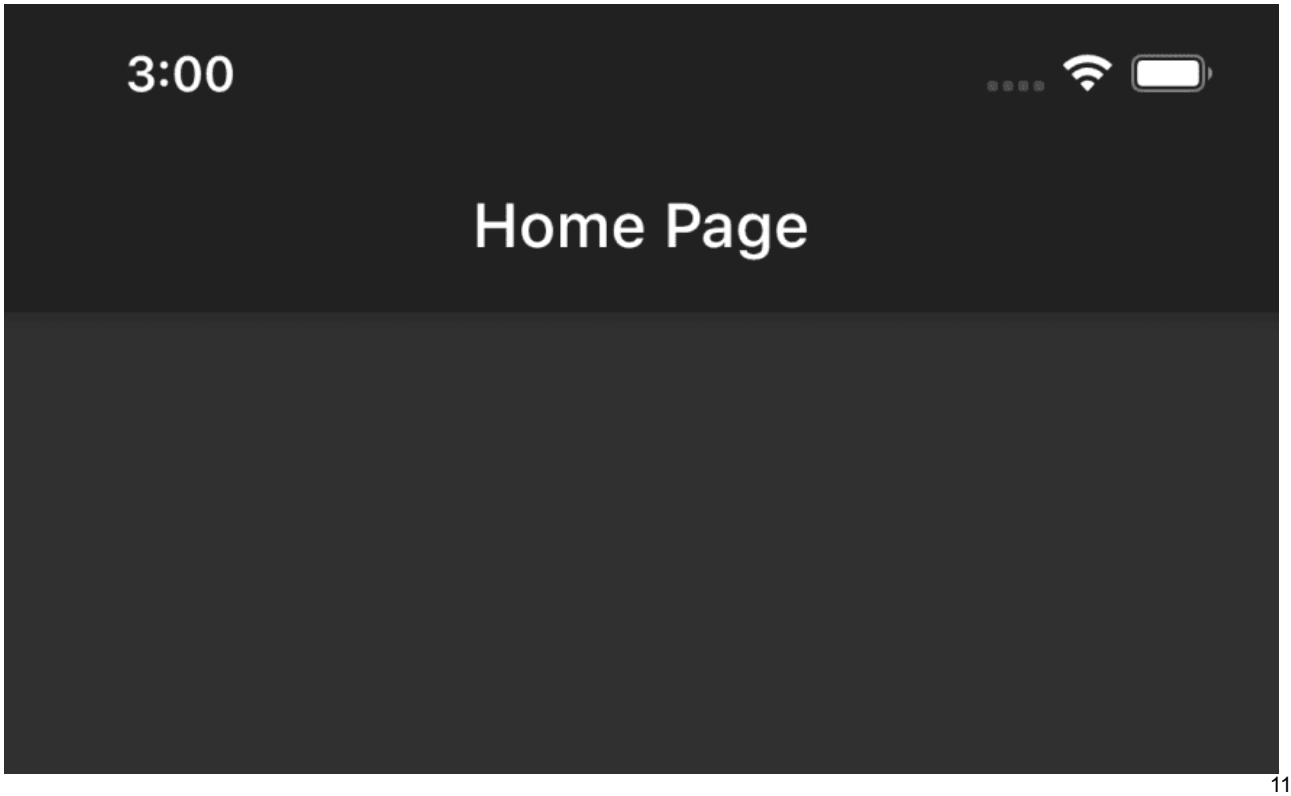
```
MultiProvider(
  providers: [
    Provider(
      create: (_) => UserService(),
    ),
    ProxyProvider<UserService, GreetingService>(
      update: (BuildContext context, UserService userService,
            GreetingService greetingService) =>
          GreetingService(userService: userService),
    ),
    ProxyProvider2<UserService, GreetingService, CartService>(
      update: (BuildContext context, UserService userService,
            GreetingService greetingService, CartService cartService) =>
          CartService(
        userService: userService,
        greetingService: greetingService,
      ),
    ),
  ],
```

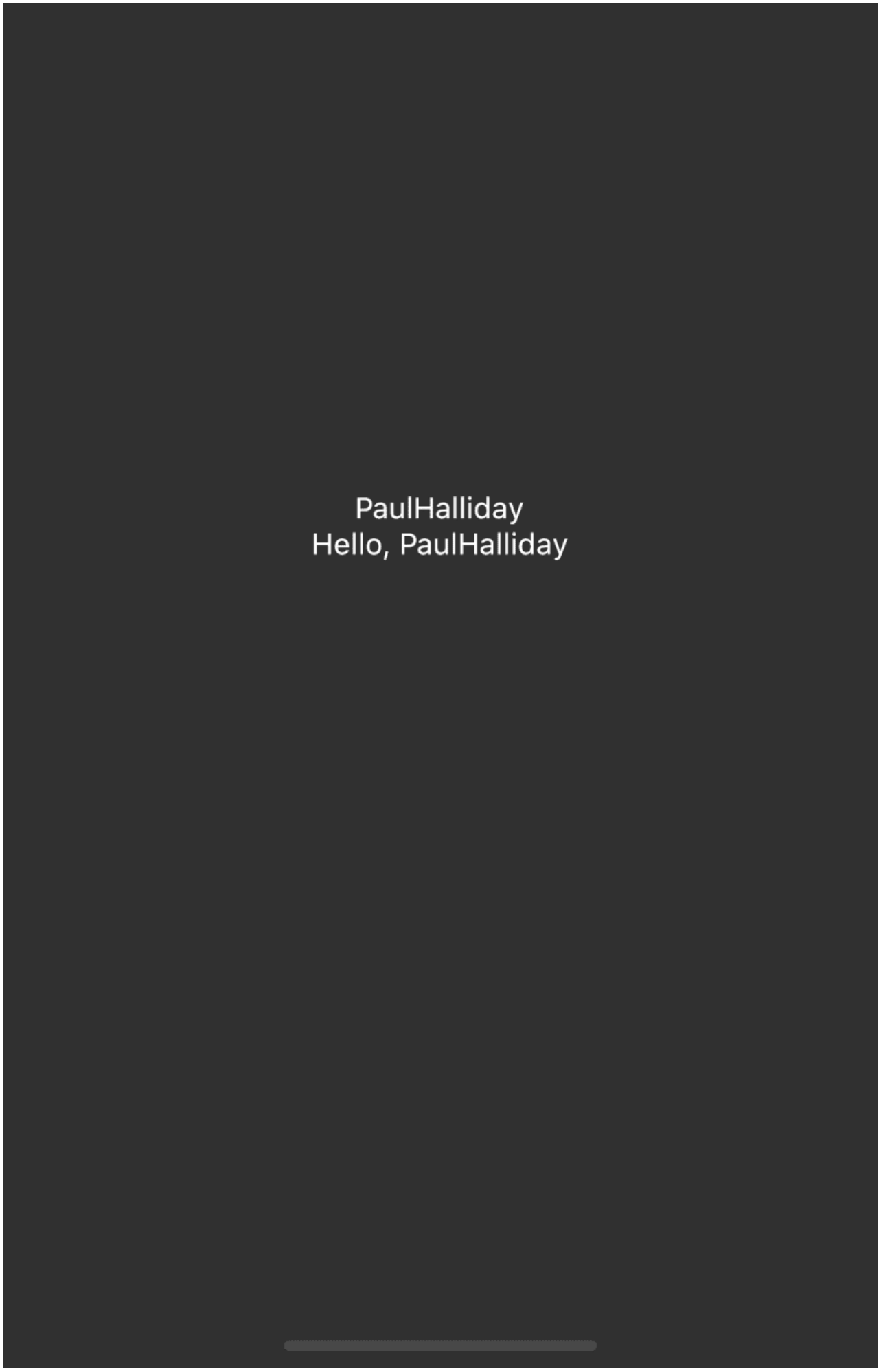
Our `CartService` does nothing interesting:

```
import 'package:flutter/foundation.dart';
import 'package:proxyprovider/application/services/greeting_service.dart';
import 'package:proxyprovider/application/services/user_service.dart';
import 'package:proxyprovider/domain/entities/user.dart';

class CartService {
  CartService({
    @required GreetingService greetingService,
    @required UserService userService,
  }) : _greetingService = greetingService,
       _userService = userService;

  GreetingService _greetingService;
  UserService _userService;

  String get cartGreeting => _greetingService.greeting;
  User get user => _userService.user;
}
```

If we were to update our `HomePage` to instead use our `CartService`, it'd look like this:

```
///lib/presentation/pages/home_page.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:proxyprovider/application/services/cart_service.dart';

class HomePage extends StatelessWidget {
  static Route<dynamic> route() => MaterialPageRoute(
        builder: (BuildContext context) => HomePage(),
      );

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Home Page"),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              Provider.of<CartService>(context).user.username,
            ),
            Text(
              Provider.of<CartService>(context).cartGreeting,
            ),
          ],
        ),
      ),
    );
  }
}
```




PaulHalliday
Hello, PaulHalliday

Summary

In this article we looked at how to get started with `ProxyProvider` to inject values that can be provided across our widget tree. I hope you found it useful!

I'd love to hear your thoughts in the comments section below!

Code for this article: https://github.com/PaulHalliday/flutter_proxyprovider