

## State Management

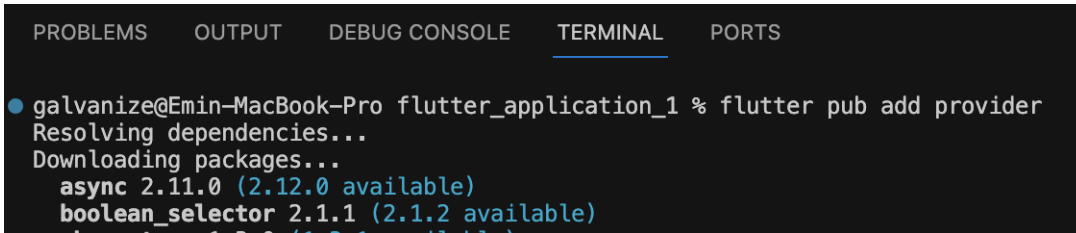
1. **Proje Ortamının Hazırlanması:** Bu derste, Flutter'da Provider kullanarak state yönetimini öğreneceğiz. **Provider**, uygulamada veri yönetimini ve UI (Kullanıcı Arayüzü) güncellemelerini kolaylaştırır. Uygulamanın sağlıklı çalışabilmesi için main.dart dosyasından başka dosyalara da ihtiyacımız olacak. Bu dosyaları lib klasörünün içerisine oluşturun.

**main.dart** → Uygulamanın başlangıç noktası

**home\_screen.dart** → Su içme saatlerini ekleyen ve listeleyen ekranlar

**water\_tracker\_model.dart** → Su içme zamanlarını yöneten model

2. **Provider Paketinin Eklenmesi:** <https://pub.dev/packages/provider/install> sayfasına giderek kütüphaneyi bulun ve VS Code terminal içerisinde `flutter pub add provider` komutunu çalıştırın.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
galvanize@Emin-MacBook-Pro flutter_application_1 % flutter pub add provider
Resolving dependencies...
Downloading packages...
  async 2.11.0 (2.12.0 available)
  boolean_selector 2.1.1 (2.1.2 available)
```

3. **Paketleri Projeye Tanıtma:** Bu dosyalardan biri olan **main.dart** ana kod bloğunu ve gerekli paketleri barındırır. Bunlardan bazıları olan gerekli paketleri projeye dahil ederek dosyayı oluşturmaya başlayalım.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'water_tracker_model.dart';
import 'home_screen.dart';
```

**provider.dart:** Durum yönetimi için kullanılır.

**water\_tracker\_model.dart:** Su takip modelimizi tanımladığımız dosya

**home\_screen.dart:** Uygulamanın başlangıç ekranı

4. **Main Fonksiyonunu Yazma:** Kodlardan biri olan main fonksiyonu yazarak projemize devam edelim. runApp içerisinde **ChangeNotifierProvider** kullanarak bir durum yönetim modeli oluşturalım.

```
void main() {  
  runApp(  
    ChangeNotifierProvider(  
      create: (context) => WaterTrackerModel(),  
      child: const MyApp(),  
    ),  
  );  
}
```

**ChangeNotifierProvider:** **Provider** paketinden gelen bir widget olup, durum değişiklikleri algılandığında widget'ları otomatik olarak yeniden oluşturur.

**Parametreler:**

- **create:** Bu parametre, bir modelin nasıl oluşturulacağını belirtir. Burada, **WaterTrackerModel()** adında bir sınıf oluşturuluyor. Böylece **WaterTrackerModel** adlı model, uygulamanın herhangi bir yerinden erişilebilir hale gelir.
- **child:** Bu parametre, sağlanan modele erişebilecek olan alt widget'ları tanımlar. Bu örnekte, **MyApp** widget'ı tanımlanmıştır.

5. **Ana Uygulama Sınıfını Oluşturma:** MyApp sınıfı için stateless bir widget oluşturun.

```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      home: HomeScreen(),  
      debugShowCheckedModeBanner: false,  
    );  
  }  
}
```

**MaterialApp:** Flutter uygulamasını oluşturur.

- **home:** Başlangıç ekranını belirler. Burada, **HomeScreen** sınıfı başlangıç ekranı olarak ayarlanmıştır.
- **debugShowCheckedModeBanner:** Debug modunda ekranın sağ üst köşesinde görünen etiketi gizler (false yaparak)

6. **Veri Saklama ve Yönetim Modelinin Oluşturulması:** İkinci dosya **water\_tracker\_model.dart** dosyası içerisinde veri saklamak ve yönetmek için bir model oluşturacağımız dosyadır.

- a. **Gerekli Kütüphanelerin Dahil Edilmesi:** Flutter'ın Material Design bileşenlerini kullanmak için gerekli olan kütüphaneyi projeye ekliyoruz.

```
dart

import 'package:flutter/material.dart';
```

- b. **WaterTrackerModel Sınıfını Tanımlama:** **ChangeNotifier**

olarak tanımlanan **WaterTrackerModel** sınıfı

- Dinleyicilere (listeners) haber vermek için kullanılır.

```
class WaterTrackerModel with ChangeNotifier {
```

- c. **Su Takip Verilerini Tutmak için Bir Liste Tanımlayın:** Su tüketim zamanlarını kaydeden **\_waterLog** isminde bir liste oluşturun.

```
class WaterTrackerModel with ChangeNotifier {
  final List<String> _waterLog = [];
```

**final:** Listenin bellekteki referansını değiştirmeyiz, ancak içeriğini güncelleyebiliriz.

- d. **Listeyi Erişime Açmak için Getter Tanımlayın:** Bir getter tanımlayarak sınıf içindeki bir değişkeni (**\_waterLog**) dışarıdan kontrollü bir şekilde erişilebilir hale getirelim.

```
final List<String> _waterLog = [];

List<String> get waterLog => _waterLog;
```

- **get** anahtar kelimesi, bir getter'in tanımlandığını belirtir.
- **waterLog**, getter'in adıdır. Bu getter bir değişken gibi görünür.
- **\_waterLog**, sınıf içinde tanımlanmış **özel bir değişkendir**:  
Dart'ta **\_** (alt çizgi) ile başlayan değişkenler veya metotlar, sınıf dışından erişilemez (**kapsülleme/encapsulation**). Bu, veri güvenliğini ve kontrolünü artırır.

e. **Listeye Eleman Ekleyen Bir Fonksiyon Tanımlayın:** Su tüketim bilgilerini listeye ekleyen ve bu değişikliği dinleyenlere bildiren **addLog** isminde bir metot oluşturalım.

```
List<String> get waterLog => _waterLog;  
  
void addLog(String time) {  
    _waterLog.add(time);  
    notifyListeners(); // UI'ı günceller  
}
```

- **addLog(String time):** Listeye yeni bir zaman bilgisi eklemek için fonksiyon tanımlar.
- **\_waterLog.add(time):** Listeye yeni bir öğe ekler.
- **notifyListeners():** Dinleyiciyi tüm UI bileşenlerini değişiklik olduğu konusunda bilgilendirir.

7. **home\_screen.dart Dosyasını Oluşturun:** Bu dosya, uygulamanın kullanıcı arayüzünü oluşturur. Kullanıcı, ana ekranda su içme zamanını kaydedebilir ve başka bir ekranda su içme geçmişini görüntüleyebilir. Aşağıda dosyanın oluşturulma adımları ve kod parçalarının açıklamaları verilmiştir.

a. **Gerekli Kütüphaneleri Dahil Etme:** Flutter'ın temel UI bileşenlerini kullanmak için **material.dart** kütüphanesini, state yönetimi için **Provider** paketini içeren **provider.dart** kütüphanesini, model sınıfını kullanmak üzere **water\_tracker\_model.dart** dosyalarını dosyamıza dahil edelim.

```
import 'package:flutter/material.dart';  
import 'package:provider/provider.dart';  
import 'water_tracker_model.dart';
```

- b. **Ana Ekran (HomeScreen) Sınıfını Tanımlama:** Sabit bir kullanıcı arayüzü sağlayan bir stateless widget oluşturun. Unutmayın, uygulamamız statefull widget ile state yönetimi gerektirmez çünkü state değişiklikleri zaten **Provider** üzerinden yönetilir.

```
import 'water_tracker_model.dart';  
  
class HomeScreen extends StatelessWidget {  
  const HomeScreen({super.key});
```

- c. **Scaffold ile Ana Sayfa Yapısını Tanımlama:** Öncelikle uygulamanın üst çubuğuna bir başlık ekler.

```
const HomeScreen({super.key});  
  
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: const Text("Su Takip Uygulamsı"),  
    ), // AppBar  
  ); // Scaffold  
}
```

Ardından actions property'si kullanılarak sağ üst köşeye bir liste simgesi ekleyin.

```
title: const Text("Su Takip Uygulamsı"),  
actions: [IconButton(onPressed: () {}, icon: const Icon(Icons.list))],  
), // AppBar
```

Kullanıcı bu düğmeye tıkladığında, su içme geçmişini gösteren başka bir sayfaya (LogScreen) yönlendirilir. Bu sayfayı oluşturmak için de boş stateless bir widget ekleyin.

```
IconButton(  
  onPressed: () {  
    Navigator.push(  
      context,  
      MaterialPageRoute(  
        builder: (context) => const logScreen(),  
      )); // MaterialPageRoute  
  },  
  icon: const Icon(Icons.list)) // IconButton
```

```
class logScreen extends StatelessWidget {  
  const logScreen({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

**d. Body: Ana İçeriğin Oluşturulması:** Center widget'ını kullanarak ekrana bir buton ekleyelim.

```
body: Center(  
  child: ElevatedButton(  
    onPressed: () {  
      final currentTime = TimeOfDay.now().format(context);  
      Provider.of<WaterTrackerModel>(context, listen: false).addLog(currentTime);  
    },  
    child: const Text('Su İçtim'),  
  ),  
),  
  
  child: const Text('Su İçtim'),  
),  
);  
}
```

- **ElevatedButton:** Kullanıcı, bu butona tıklayarak su içtiğini işaretler.

- **onPressed:** Butona tıklanıldığında çağrılan işlemdir.
  - **TimeOfDay.now().format(context):** O anki saat bilgisini alır ve kullanıcıya okunabilir bir formatta döndürür.
  - **Provider.of<WaterTrackerModel>(context, listen: false).addLog(currentTime):** Model üzerinden su içme saatini ekler ve değişikliği UI'ya bildirir.

**Provider.of:**

- daha önce uygulamanın ChangeNotifierProvider ile bağlandığı (provided) bir modeli almak için kullanılır.
- Bu satır, WaterTrackerModel sınıfının bir örneğine (instance) erişir.

**context Nedir?**

- context, Flutter'da widget ağacındaki konumu temsil eder.
- Provider.of çağrıldığında, context kullanılarak ilgili WaterTrackerModel örneği bulunur.

**listen: false:**

- Bu kodun **UI değişikliklerini dinlemesine gerek olmadığını** belirtir.
- Yalnızca modele erişmek ve bir işlem yapmak için kullanılır.

Dinlemeyi kapatmanın avantajı:

- Performansı artırır: WaterTrackerModel'de bir değişiklik olduğunda, bu kodla ilgili olan widget'lar yeniden oluşturulmaz (rebuild edilmez).

**addLog(currentTime)**

- Erişilen modelin (WaterTrackerModel) addLog metodu çağrılır.
- currentTime bir String parametredir ve su içme zamanını temsil eder.
- Ne Yapar?
- addLog metodu, currentTime değerini \_waterLog adlı listeye ekler:

**e. LogScreen Sınıfını Tanımlama:**

- LogScreen Widget'ı StatelessWidget olarak sadece su içme saatlerinin bir listesini göstermek için tasarlanmıştır.
- Kullanıcı burada geçmiş su içme saatlerini görebilir.

```
class LogScreen extends StatelessWidget {  
  const LogScreen({super.key});  
}
```

**f. Su İçme Kayıtlarını İçeren Listeyi Alma:**

- **Provider.of** metodu, widget ağacından belirli bir modelin (**WaterTrackerModel**) örneğini alır.
- **waterLog**: **WaterTrackerModel** içinde tanımlı olan bir getterdir. Bu getter, su içme kayıtlarını içeren bir listeyi döndürür.

```
@override
Widget build(BuildContext context) {
  final waterLog = Provider.of<WaterTrackerModel>(context).waterLog;
```

**g. Scaffold ile Listeleme Sayfasını Oluşturma:**

```
final waterLog = Provider.of<WaterTrackerModel>(context).waterLog;

return Scaffold(
  appBar: AppBar(
    title: const Text('İçilen Su Saatleri'),
  ),
```

- **title: Text('İçilen Su Saatleri')**: Su içme geçmişini listelemek için üst çubuk başlığını ayarlar.

**h. Body: Su İçme Saatleri Listesini Görüntüleme:** Bir **ListView**.

**builder** kullanarak dinamik bir liste oluşturun. Bu liste, kullanıcının içtiği su saatlerini göstermek için her elemanı bir **ListTile** olarak yapılandırır.



```
return Scaffold(  
  appBar: AppBar(  
    title: const Text('İçilen Su Saatleri'),  
  ),  
  body: ListView.builder(  
    itemCount: waterLog.length,  
    itemBuilder: (context, index) {  
      return ListTile(  
        leading: const Icon(Icons.local_drink),  
        title: Text('Saat: ${waterLog[index]}'),  
      );  
    },  
  ),  
);  
}
```

**itemCount: waterLog.length**

- itemCount, listenin kaç eleman içereceğini belirtir.
- Burada waterLog.length, su içme kayıtlarının bulunduğu listenin uzunluğunu temsil eder.

**itemBuilder: (context, index)**

- itemBuilder, liste elemanlarını nasıl oluşturacağınızı tanımlayan bir fonksiyondur.
- ListView.builder, her eleman için bu fonksiyonu çağırır.

Parametreler:

- context:
  - Elemanın widget ağacındaki konumunu temsil eder.
  - Flutter'ın widget'larla ilgili işlemleri yürütmesine yardımcı olur.
- index:
  - O an oluşturulan liste elemanının sıra numarasıdır (0'dan başlar).

İşleyiş:

- Her çağrıda, index numarasına göre bir eleman oluşturulur.

**return ListTile(...)**

ListTile, Flutter'da basit bir liste elemanıdır. İçinde bir simge, başlık, alt başlık ve diğer unsurları barındırabilir.

**leading: const Icon(Icons.local\_drink)**

- leading:
  - Elemanın sol tarafına bir widget eklemek için kullanılır.

- Burada, sabit bir **su şişesi simgesi** (Icons.local\_drink) eklenmiştir.  
**title: Text('Saat: \${waterLog[index]}')**
- **title**, elemanın ana metin alanıdır.
- Dinamik olarak waterLog[index] değerini alır ve metni oluşturur.