

AHMET YESEVİ ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

MOBİL ÖĞRENCİ YOKLAMA UYGULAMASI

GERÇEKLEŞTİRME RAPORU

HAZIRLAYAN

182132062 Çağlar AVŞAROĞLU

DANIŞMAN ÖĞRETİM ÜYESİ

Sami ACAR

2020

İçindekiler

[1. Giriş 3](#_Toc59731363)

[2. Ekran Tasarımları 3](#_Toc59731364)

[2.1. Giriş Ekranı 3](#_Toc59731365)

[2.2. Yoklama Ekranı 6](#_Toc59731366)

[2.3. Öğrenci Listesi Ekranı 8](#_Toc59731367)

[2.4. Öğrenci Ekleme Ekranı 10](#_Toc59731368)

[2.5. Sınav Listesi Ekranı 12](#_Toc59731369)

[2.6. Sınav Ekleme Ekranı 14](#_Toc59731370)

[2.7. Raporlar Ekranı 17](#_Toc59731371)

[3. Veri Tabanı Tasarımı 18](#_Toc59731372)

[ Öğrenci ORM Yapısı 19](#_Toc59731373)

[ Sınav ORM Yapısı 19](#_Toc59731374)

[ Oturum ORM Yapısı 20](#_Toc59731375)

[ DbHelper ORM Yapısı (Veri tabanı işlem yardımcısı) 21](#_Toc59731376)

[3.1. Veri Tabanı Tablolar Arası İlişki 24](#_Toc59731377)

# Giriş

Bu raporda Mobil Öğrenci Yoklama Uygulaması Geliştirme Proje çalışmasına ilişkin gerçekleştirme öğelerine yer verilmektedir. Ödev içeriğinde temel olarak proje de gerçekleştirilen ekran ekranlar ve bu ekranlarda yapılan iş ve işlemler anlatılırken, oluşturulan veri tabanına da yer verilecektir. Veri tabanı oluşturulurken, projede saklanması istenilen verilerin nasıl ve ne şekilde tutulduğuna dair bilgilerde paylaşılacaktır.

# Ekran Tasarımları

Mobil Öğrenci Yoklama Uygulaması için gerçekleştirilen ekranlar şunlardır:

* Giriş Ekranı,
* Yoklama Ekranı,
* Öğrenci Listesi Ekranı,
* Öğrenci Ekleme Ekranı,
* Sınav Listeleme Ekranı,
* Sınav Ekleme Ekranı ve
* Rapor Ekranıdır.

Aşağıda tasarım raporunda bahsedilen ve gerçekleştirilen ekran görüntüleri bulunmaktadır. Bazı ekranlar tümleşik olduğu için iç ekranlar hakkında bilgilendirmelere ekranların alt kısmında yer verilecektir. Ayrıca asıl pencerelere ilaveten alt pencereler ile ilgili bilgilendirmeler de bu ekranların anlatımından sonra yapılacaktır.

## Giriş Ekranı



Kullanıcının uygulamaya girişte karışılacak oluğu ekrandır. Bu ekranda diğer ekranlara geçiş için bir menü bulunmaktadır. Ayrıca aktif olan sınavların listesi de bu ekranda yer alacaktır. Kullanıcı listelenen sınavı kaydırarak tarama ekranına geçiş yaparak sınava ilişkin yoklama yapabilmektedir.

class Anasayfa extends StatefulWidget {  
 @override  
 State<StatefulWidget> createState() => \_AnasayfaState();  
}  
  
class \_AnasayfaState extends State<Anasayfa> {  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: Text("Aktif Sınavlarınız Listelenmektedir")),  
 drawer: MyDrawer(),  
  
 body: new Builder(  
 builder: (BuildContext context) {  
 return Column(  
  
 mainAxisAlignment: MainAxisAlignment.spaceAround,  
 children: <Widget>[  
 Expanded(  
 flex: 10,  
 child: FutureBuilder<List<Sinav>>(  
  
  
 future: DBHelper().getUnblockedSinav(),  
 builder: (BuildContext context,  
 AsyncSnapshot<List<Sinav>> snapshot) {  
 if (snapshot.hasData) {  
 return ListView.builder(  
 itemCount: snapshot.data.length,  
 itemBuilder: (BuildContext context, int index) {  
 Sinav item = snapshot.data[index];  
 return Dismissible(  
 key: UniqueKey(),  
 background: Container(color: Colors.*red*),  
 onDismissed: (direction) {  
 Navigator.*of*(context)  
 .push(MaterialPageRoute(  
 builder: (context) => ScanPage(item)));  
 },  
 child: Container(  
 decoration: BoxDecoration(  
 color: Colors.*amberAccent*,  
 border: Border.all()),  
 margin: const EdgeInsets.all(5.0),  
 padding: const EdgeInsets.all(5.0),  
 child: ListTile(  
  
 title: Text("Sınav Adı : " + item.sinav\_adi,  
 style: TextStyle(  
 fontWeight: FontWeight.*bold*)),  
 leading: Text(  
 "Açıklama : " + item.sinav\_aciklama,  
 textAlign: TextAlign.center,  
  
 overflow: TextOverflow.ellipsis,  
 style: TextStyle(color: Colors.*redAccent*,  
 fontSize: 25,  
 fontWeight: FontWeight.*bold*),),  
  
 ),),  
 );  
 },  
 );  
 } else {  
 return Center(child:  
 Text("Aktif Sınav Bulunmamaktadır"),);  
 }  
 },  
 ),  
 ),  
  
  
 ],  
 );  
 }  
 )  
 );  
 }  
}

* Sayfalar arası geçişi sağlayan menü yapısı

class MyDrawer extends StatefulWidget {  
 @override  
 State<StatefulWidget> createState() => \_MyDrawerState();  
}  
  
class \_MyDrawerState extends State {  
 @override  
 Widget build(BuildContext context) {  
 return Drawer(  
 child: ListView(  
 children: <Widget>[  
 DrawerHeader(  
 child: Align(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: <Widget>[  
 Icon(  
 Icons.*add\_to\_home\_screen*,  
 color: Colors.*white*,  
 size: 100.0,  
 ),  
 Text(  
 "Qr Yoklama Uygulaması",  
 style: TextStyle(color: Colors.*white*, fontSize: 25.0),  
 ),  
 ],  
 ),  
 ),  
 decoration: BoxDecoration(  
 color: Colors.*blue*,  
 ),  
 ),  
 ListTile(  
 leading: Icon(Icons.*home*),  
 title: Text('Anasayfa'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/");  
 },  
 ),  
 ExpansionTile(  
 leading: Icon(Icons.*perm\_device\_information*),  
 title: Text('Öğrenci İşlemleri'),  
 trailing: Icon(Icons.*arrow\_drop\_down*),  
 children: <Widget>[  
 ListTile(  
 title: Text('Öğrenci Listem'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/ogrencilistesi");  
 },  
 ),  
 ListTile(  
 title: Text('Öğrenci Ekle'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/ogrenciekle");  
 },  
 ),  
 ],  
 ),  
 ExpansionTile(  
 leading: Icon(Icons.*perm\_device\_information*),  
 title: Text('Sınav İşlemleri'),  
 trailing: Icon(Icons.*arrow\_drop\_down*),  
 children: <Widget>[  
 ListTile(  
 title: Text('Sınav Listem'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/sinavlistesi");  
 },  
 ),  
 ListTile(  
 title: Text('Sınav Ekle'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/sinavekle");  
 },  
 ),  
 ],  
 ),  
 ListTile(  
 leading: Icon(Icons.*local\_laundry\_service*),  
 title: Text('Sınav Raporlarım'),  
 trailing: Icon(Icons.*arrow\_right*),  
 onTap: () {  
 Navigator.*pushNamed*(context, "/raporlar");  
 },  
 ),  
 ],  
 ),  
 );  
 }  
}

## Yoklama Ekranı

metin, ekran, ekran görüntüsü, telefon içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcı seçmiş olduğu sınav ile ilgili yoklama yapabilecek olduğu ekrandır. Bu ekran aracılığı ile tarayıcı açılarak Qr kod okuması yapmakta, okunan qr ile elde edilen öğrenci bilgisiyle öğrencinin sınavda olup olmadığına dair bilgi edinilmekte ve oturum listesinde ise oturum listesine eklenmekte, yok ise öğrencinin sınav listesinde olmadığına dair bilgi dönülmektedir.

class ScanPage extends StatefulWidget {  
 final Sinav sinav;  
  
 const ScanPage(this.sinav);  
  
 @override  
 \_ScanPageState createState() => \_ScanPageState();  
}  
  
class \_ScanPageState extends State<ScanPage> {  
 String qrCodeResult = "Henüz Bir Tarama Gerçekleşmedi";  
 String durum = "Tarama Yapılmadı";  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text("Sınav Adı : " + widget.sinav.sinav\_adi),  
 centerTitle: true,  
 ),  
 drawer: MyDrawer(),  
 body: Container(  
 padding: EdgeInsets.all(20.0),  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 crossAxisAlignment: CrossAxisAlignment.stretch,  
 children: <Widget>[  
 Text(  
 "Sınav Adı : " + widget.sinav.sinav\_adi,  
 style: TextStyle(fontSize: 25.0, fontWeight: FontWeight.*bold*),  
 textAlign: TextAlign.center,  
 ),  
 Text(  
 "Öğrenci No : " + qrCodeResult,  
 style: TextStyle(  
 fontSize: 20.0,  
 ),  
 textAlign: TextAlign.center,  
 ),  
 SizedBox(  
 height: 20.0,  
 ),  
 Text(  
 durum,  
 style: TextStyle(  
 fontSize: 20.0,  
 ),  
 textAlign: TextAlign.center,  
 ),  
 SizedBox(  
 height: 20.0,  
 ),  
 FlatButton(  
 padding: EdgeInsets.all(15.0),  
 onPressed: () async {  
 String codeSanner = await BarcodeScanner  
 .*scan*(); *//barcode scnner* String sonuc = await DBHelper().oturumKontrol(  
 widget.sinav.sinav\_id, int.*parse*(codeSanner));  
 setState(() {  
 qrCodeResult = codeSanner;  
 durum = sonuc;  
 });  
 },  
 child: Text(  
 "Tarayıcıyı Aç",  
 style:  
 TextStyle(color: Colors.*blue*, fontWeight: FontWeight.*bold*),  
 ),  
 shape: RoundedRectangleBorder(  
 side: BorderSide(color: Colors.*blue*, width: 3.0),  
 borderRadius: BorderRadius.circular(20.0)),  
 )  
 ],  
 ),  
 ),  
 );  
 }  
}

## Öğrenci Listesi Ekranı

metin, ekran, ekran görüntüsü, telefon içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcının sisteme kaydetmiş olduğu öğrencileri listeleyebilecek olduğu ekrandır. Bu ekran ile öğrencileri aktif veya pasif yapabilmenin dışında, öğrenci silme ve toplu öğrenci silme işlemleri de yapılabilmektedir.

class Ogrencilisteleme extends StatefulWidget {  
 @override  
 State<StatefulWidget> createState() => \_OgrencilistelemeState();  
}  
  
class \_OgrencilistelemeState extends State {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: Text("Öğrenci Listesi")),  
 drawer: MyDrawer(),  
 body: FutureBuilder<List<Ogrenci>>(  
 future: DBHelper().getAllOgrenciler(),  
 builder: (BuildContext context, AsyncSnapshot<List<Ogrenci>> snapshot) {  
 if (snapshot.hasData) {  
 return ListView.builder(  
 itemCount: snapshot.data.length,  
 itemBuilder: (BuildContext context, int index) {  
 Ogrenci item = snapshot.data[index];  
 return Dismissible(  
 key: UniqueKey(),  
 background: Container(color: Colors.*red*),  
 onDismissed: (direction) {  
 DBHelper().deleteOgrenci(item.ogrenci\_id);  
 },  
 child: ListTile(  
 title: Text(item.ogrenci\_adi,  
 style: TextStyle(fontWeight: FontWeight.*bold*)),  
 subtitle: RichText(  
 text: TextSpan(  
 text: item.ogrenci\_adi,  
 ),  
 ),  
 leading: Text(item.ogrenci\_id.toString()),  
 trailing: Checkbox(  
 onChanged: (bool value) {  
 DBHelper().blockOrUnblockOgrenci(item);  
 setState(() {});  
 },  
 value: item.blocked,  
 ),  
 ),  
 );  
 },  
 );  
 } else {  
 return Center(child: CircularProgressIndicator());  
 }  
 },  
 ),  
 floatingActionButton: FloatingActionButton(  
 child: Icon(Icons.*settings\_backup\_restore*),  
 tooltip: "Tüm Öğrencileri Sil",  
 onPressed: () async {  
 await DBHelper().deleteAllOgrenci();  
 print("işlem oldu");  
 setState(() {});  
 },  
 ),  
 );  
 }  
}

## Öğrenci Ekleme Ekranı

metin, ekran, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcının sisteme yeni öğrenci eklemek için kullanacak olduğu ekrandır. Eklenecek öğrenci numarası uniqe yani tekil olmak durumundadır. Öğrenci ekleme durumuna göre kullanıcıya bilgi döndürülmektedir.

class Ogrencieklemesayfasi extends StatefulWidget {  
 Ogrencieklemesayfasi({Key key}) : super(key: key);  
  
 @override  
 State<StatefulWidget> createState() => \_OgrencieklemeState();  
}  
  
class \_OgrencieklemeState extends State<Ogrencieklemesayfasi> {  
 @override  
 void initState() {  
 super.initState();  
 }  
  
 int ogrenci\_id = 0;  
 String ogrenci\_adi = "";  
 bool blocked = true;  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: Text("Yeni Öğrenci Ekleme Sayfası")),  
 drawer: MyDrawer(),  
*// resizeToAvoidBottomPadding: false,* body: ListView(  
 *//mainAxisAlignment: MainAxisAlignment.center,* children: <Widget>[  
 Text("Aşağıdaki Forma Yeni Öğrenci Bilgilerini Yazınız",  
 style: TextStyle(fontSize: 20), textAlign: TextAlign.center),  
 SizedBox(height: 10),  
 *//ilaç adı* TextField(  
 decoration: InputDecoration(  
 labelText: "Öğrenci İd - No", border: OutlineInputBorder()),  
 textAlign: TextAlign.center,  
 *// Her yeni veri girildiğinde çalışır* onChanged: (veri) {  
 ogrenci\_id = int.*parse*(veri);  
 },  
 *// Klavyedeki Gönder(Bitti) tuşuna basınca çalışır* onSubmitted: (veri) {  
 ogrenci\_id = int.*parse*(veri);  
 },  
 ),  
 SizedBox(height: 10),  
 TextField(  
 decoration: InputDecoration(  
 labelText: "Öğrenci Adı - Soyadı",  
 border: OutlineInputBorder()),  
 textAlign: TextAlign.center,  
 *// Her yeni veri girildiğinde çalışır* onChanged: (veri) {  
 ogrenci\_adi = veri;  
 },  
 *// Klavyedeki Gönder(Bitti) tuşuna basınca çalışır* onSubmitted: (veri) {  
 ogrenci\_adi = veri;  
 },  
 ),  
 SizedBox(height: 10),  
  
 Center(  
 child: RaisedButton(  
 child: Text("Yeni Öğrenci Ekle"),  
 *// Navigator.pop ile bir önceki sayfaya dönücek* onPressed: () async {  
 Ogrenci yeni = Ogrenci(  
 ogrenci\_id: ogrenci\_id,  
 ogrenci\_adi: ogrenci\_adi,  
 blocked: true);  
 await DBHelper().newOgrenci(yeni);  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text("Öğrenci Ekleme İşlemi"),  
 content: Text("Başarılı"),  
 actions: <Widget>[  
 FlatButton(  
 child: Text("Kapat"),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 )  
 ]);  
 });  
 }  
 *//onPressed: () => Navigator.pop(context),* )),  
 ],  
 ),  
 );  
 }  
}

## Sınav Listesi Ekranı

metin, ekran, ekran görüntüsü, telefon içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcının sisteme eklediği sınavları görüntüleyebilecek olduğu ekrandır. Bu ekran aracılığı ile sınavları aktif veya pasif hale getirebilmenin yanı sıra, tek tek ve ya toplu sınav silme işlemleri de gerçekleştirilebilmektedir.

class Sinavlisteleme extends StatefulWidget {  
 @override  
 State<StatefulWidget> createState() => \_SinavlistelemeState();  
}  
  
class \_SinavlistelemeState extends State {  
   
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
  
 appBar: AppBar(title: Text("Sinav Listesi")),  
 drawer: MyDrawer(),  
  
 body: FutureBuilder<List<Sinav>>(  
  
 future: DBHelper().getAllSinavlar(),  
 builder: (BuildContext context, AsyncSnapshot<List<Sinav>> snapshot) {  
 if (snapshot.hasData) {  
 return ListView.builder(  
 itemCount: snapshot.data.length,  
 itemBuilder: (BuildContext context, int index) {  
 Sinav item = snapshot.data[index];  
 return Dismissible(  
 key: UniqueKey(),  
 background: Container(color: Colors.*red*),  
 onDismissed: (direction) {  
 DBHelper().deleteOgrenci(item.sinav\_id);  
 },  
 child: ListTile(  
 title: Text(item.sinav\_adi,  
 style: TextStyle(fontWeight: FontWeight.*bold*)),  
 subtitle: RichText(  
 text: TextSpan(  
 text: item.sinav\_aciklama,  
 ),  
 ),  
  
 leading: Text(item.sinav\_id.toString()),  
 trailing: Checkbox(  
 onChanged: (bool value) {  
 DBHelper().blockOrUnblockSinav(item);  
 setState(() {});  
 },  
 value: item.blocked,  
 ),  
 ),  
 );  
 },  
 );  
 *// veri yok ise ekran ortasında dönen progressindicator göster* } else {  
 return Center(child: CircularProgressIndicator());  
 }  
 },  
 ),  
  
 floatingActionButton: FloatingActionButton(  
 child: Icon(Icons.*settings\_backup\_restore*),  
 tooltip: "Tüm Sınavları Sil",  
 onPressed: () async {  
 await DBHelper().deleteAllSinav();  
 print("işlem oldu");  
  
  
 setState(() {});  
 },  
 ),  
 );  
 }  
}

## Sınav Ekleme Ekranı

metin, ekran görüntüsü, ekran içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcının sisteme yeni sınav eklemek için kullanacak olduğu ekrandır. Bu ekran aracılığı ile sisteme yeni sınav tanımlaması yapılabilmekte ve aktif olan öğrenciler bu sınava atanabilmektedir.

class Sinaveklemesayfasi extends StatefulWidget {  
 Sinaveklemesayfasi({Key key}) : super(key: key);  
  
 @override  
 State<StatefulWidget> createState() => \_SinaveklemeState();  
}  
  
class \_SinaveklemeState extends State<Sinaveklemesayfasi> {  
  
 @override  
 void initState() {  
 super.initState();  
 }  
  
 String sinav\_adi = "";  
 String sinav\_aciklama = "";  
 bool blocked = true;  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: Text("Yeni Sınav Ekleme Sayfası")),  
 drawer: MyDrawer(),  
 body: ListView(  
 children: <Widget>[  
 Text("Aşağıdaki Forma Yeni Sınav Bilgilerini Yazınız",  
 style: TextStyle(fontSize: 20), textAlign: TextAlign.center),  
 SizedBox(height: 10),  
  
 TextField(decoration: InputDecoration(labelText: "Sınav Adı",  
 border: OutlineInputBorder()  
 ), textAlign: TextAlign.center,  
 *// Her yeni veri girildiğinde çalışır* onChanged: (veri) {  
 sinav\_adi = veri;  
 },  
 *// Klavyedeki Gönder(Bitti) tuşuna basınca çalışır* onSubmitted: (veri) {  
 sinav\_adi = veri;  
 },  
 ),  
 SizedBox(height: 10),  
 *//ilaç açıklama* TextField(decoration: InputDecoration(labelText: "Sinav Açıklaması",  
 border: OutlineInputBorder()  
 ), textAlign: TextAlign.center,  
  
 onChanged: (veri) {  
 sinav\_aciklama = veri;  
 },  
  
 onSubmitted: (veri) {  
 sinav\_aciklama = veri;  
 },  
 ),  
 SizedBox(height: 10),  
  
  
 Center(  
 child: RaisedButton(  
 child: Text("Yeni Sınav Ekle"),  
 *// Navigator.pop ile bir önceki sayfaya dönücek* onPressed: () async {  
 Sinav yeni = Sinav(sinav\_adi: sinav\_adi,  
 sinav\_aciklama: sinav\_aciklama,  
 blocked: true);  
 await DBHelper().newSinav(yeni);  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text("Sınav Ekleme İşlemi"),  
 content: Text("Başarılı"),  
 actions: <Widget>[ FlatButton(  
 child: Text("Kapat"),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 )  
 ]  
 );  
 }  
 );  
 }  
 *//onPressed: () => Navigator.pop(context),* )),  
 ],  
 ),  
 );  
 }  
  
  
}

## Raporlar Ekranı

metin, ekran, elektronik eşyalar, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kullanıcının sınavlardaki katılım bilgilerini listeleyebilecek olduğu ekrandır. Bu ekran aracılığı ile qr kod okuma işlemi ile oturumda var olan öğrenciler işlenmekte olup ayrıca manuel olaraktan öğrenci sınava eklenip çıkartılabilmektedir. Ayrıca tek tek veya toplu olarak oturum bilgilerini sistemden silinebilmektedir.

class RaporlamaSayfasi extends StatefulWidget {  
 RaporlamaSayfasi({Key key}) : super(key: key);  
  
 @override  
 State<StatefulWidget> createState() => \_RaporlamaSayfasiState();  
}  
  
  
class \_RaporlamaSayfasiState extends State {  
 Future<List<Oturum>> \_oturum = DBHelper().getAllOturumlar();  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
  
 appBar: AppBar(title: Text("Sınav Oturum Raporları")),  
 drawer: MyDrawer(),  
  
 body: FutureBuilder<List<Oturum>>(  
  
 future: DBHelper().getAllOturumlar(),  
 builder: (BuildContext context, AsyncSnapshot<List<Oturum>> snapshot) {  
 if (snapshot.hasData) {  
 return ListView.builder(  
 itemCount: snapshot.data.length,  
 itemBuilder: (BuildContext context, int index) {  
 Oturum item = snapshot.data[index];  
 return Dismissible(  
 key: UniqueKey(),  
 background: Container(color: Colors.*red*),  
 onDismissed: (direction) {  
 DBHelper().blockOrUnblockOturum(item);  
 },  
 child: ListTile(  
 title: Text(  
 item.sinav\_adi + "=> " + item.ogrenci\_id.toString() +  
 " - " + item.ogrenci\_adi,  
 style: TextStyle(fontWeight: FontWeight.*bold*)),  
 leading: Text(item.oturum\_id.toString()),  
 trailing: Checkbox(  
 onChanged: (bool value) {  
 DBHelper().blockOrUnblockOturum(item);  
 setState(() {});  
 },  
 value: item.blocked,  
 ),  
 ),  
 );  
 },  
 );  
 } else {  
 return Center(child: CircularProgressIndicator());  
 }  
 },  
 ),  
  
 floatingActionButton: FloatingActionButton(  
 child: Icon(Icons.*settings\_backup\_restore*),  
 tooltip: "Tüm Oturumları Sil",  
 onPressed: () async {  
 await DBHelper().deleteAllOturum();  
 print("işlem oldu");  
  
  
 setState(() {});  
 },  
 ),  
 );  
 }  
}

# Veri Tabanı Tasarımı

Projede kullanılacak olunan veri tabanı Sqlite3 ile oluşturulmuştur. Dart programlama dili ile standart gelen bu veri tabanı küçük çaplı projelerde etkili olarak çalışmaktadır. Proje de bir adet veri tabanı kullanılmış olup ve bu veri tabanına ait 3 üç adet tablo oluşturulmuştur. Oluşturan tablolar:

* Öğrenci bilgileri için: ogrenci tablosu,
* Sınav bilgileri için: sinav tablosu,
* Oturum bilgileri için: oturum tablosu olarak oluşturulmuştur.

## Öğrenci ORM Yapısı

Ogrenci ogrenciFromJson(String str) {  
 final jsonData = json.decode(str);  
 return Ogrenci.fromMap(jsonData);  
}  
  
String ogrenciToJson(Ogrenci data) {  
 final dyn = data.toMap();  
 return json.encode(dyn);  
}  
  
class Ogrenci {  
 int ogrenci\_id;  
 String ogrenci\_adi;  
 bool blocked;  
  
 Ogrenci({  
 this.ogrenci\_id,  
 this.ogrenci\_adi,  
 this.blocked,  
 });  
   
 factory Ogrenci.fromMap(Map<String, dynamic> json) => new Ogrenci(  
 ogrenci\_id: json["ogrenci\_id"],  
 ogrenci\_adi: json["ogrenci\_adi"],  
 blocked: json["blocked"] == 1,  
 );  
   
 Map<String, dynamic> toMap() => {  
 "ogrenci\_id": ogrenci\_id,  
 "ogrenci\_adi": ogrenci\_adi,  
 "blocked": blocked,  
 };  
}

## Sınav ORM Yapısı

Sinav sinavFromJson(String str) {  
 final jsonData = json.decode(str);  
 return Sinav.fromMap(jsonData);  
}  
  
String sinavToJson(Sinav data) {  
 final dyn = data.toMap();  
 return json.encode(dyn);  
}  
  
class Sinav {  
 int sinav\_id;  
 String sinav\_adi;  
 String sinav\_aciklama;  
 bool blocked;  
  
 Sinav({  
 *// ignore: non\_constant\_identifier\_names* this.sinav\_id,  
 this.sinav\_adi,  
 this.sinav\_aciklama,  
 this.blocked,  
 });  
   
 factory Sinav.fromMap(Map<String, dynamic> json) => new Sinav(  
 sinav\_id: json["sinav\_id"],  
 sinav\_adi: json["sinav\_adi"],  
 sinav\_aciklama: json["sinav\_aciklama"],  
 blocked: json["blocked"] == 1,  
 );  
   
 Map<String, dynamic> toMap() => {  
 "sinav\_id": sinav\_id,  
 "sinav\_adi": sinav\_adi,  
 "sinav\_aciklama": sinav\_aciklama,  
 "blocked": blocked,  
 };  
}

## Oturum ORM Yapısı

Oturum oturumFromJson(String str) {  
 final jsonData = json.decode(str);  
 return Oturum.fromMap(jsonData);  
}  
  
String oturumToJson(Oturum data) {  
 final dyn = data.toMap();  
 return json.encode(dyn);  
}  
  
class Oturum {  
 int oturum\_id;  
 int sinav\_id;  
 String sinav\_adi;  
 int ogrenci\_id;  
 String ogrenci\_adi;  
 bool blocked;  
  
 Oturum({  
 *// ignore: non\_constant\_identifier\_names* this.oturum\_id,  
 this.sinav\_id,  
 this.sinav\_adi,  
 this.ogrenci\_id,  
 this.ogrenci\_adi,  
 this.blocked,  
 });  
  
 *// gelen map verisini json'a dönüştürür* factory Oturum.fromMap(Map<String, dynamic> json) => new Oturum(  
 oturum\_id: json["oturum\_id"],  
 sinav\_id: json["sinav\_id"],  
 sinav\_adi: json["sinav\_adi"],  
 ogrenci\_id: json["ogrenci\_id"],  
 ogrenci\_adi: json["ogrenci\_adi"],  
 blocked: json["blocked"] == 1,  
 );  
  
 *// gelen json' verisini Map'e dönüştürür* Map<String, dynamic> toMap() => {  
 "oturum\_id": oturum\_id,  
 "sinav\_id": sinav\_id,  
 "sinav\_adi": sinav\_adi,  
 "ogrenci\_id": ogrenci\_id,  
 "ogrenci\_adi": ogrenci\_adi,  
 "blocked": blocked,  
 };  
}

## DbHelper ORM Yapısı (Veri tabanı işlem yardımcısı)

class DBHelper {  
 static Future<Database> *database*() async {  
 final dbPath = await getDatabasesPath();  
 const ogrenciSQL = "CREATE TABLE Ogrenci ("  
 "ogrenci\_id INTEGER PRIMARY KEY,"  
 "ogrenci\_adi TEXT,"  
 "blocked BIT"  
 ")";  
 const sinavSQL = "CREATE TABLE Sinav ("  
 "sinav\_id INTEGER PRIMARY KEY,"  
 "sinav\_adi TEXT,"  
 "sinav\_aciklama TEXT,"  
 "blocked BIT"  
 ")";  
 const oturumSQL = "CREATE TABLE Oturum ("  
 "oturum\_id INTEGER PRIMARY KEY,"  
 "sinav\_id INTEGER,"  
 "sinav\_adi TEXT,"  
 "ogrenci\_id INTEGER,"  
 "ogrenci\_adi TEXT,"  
 "blocked BIT,"  
 "UNIQUE(sinav\_id,ogrenci\_id),FOREIGN KEY(sinav\_id) REFERENCES Sinav(sinav\_id)"  
 ")";  
 return openDatabase(join(dbPath, "mobil\_yoklama\_uygulamasi.db"),  
 onCreate: (db, version) async {  
 await db.execute(ogrenciSQL);  
 db.execute(sinavSQL);  
 db.execute(oturumSQL);  
 },  
 version: 1,  
 );  
 }  
  
 newOgrenci(Ogrenci newOgrenci) async {  
 final db = await *database*();  
 var sonuc = await db.insert("Ogrenci", newOgrenci.toMap());  
 return sonuc;  
 }  
  
 Future<List<Ogrenci>> getAllOgrenciler() async {  
 final db = await *database*();  
 var sonuc = await db.query("Ogrenci");  
 List<Ogrenci> list = sonuc.isNotEmpty ? sonuc.map((c) => Ogrenci.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 Future<List<Ogrenci>> getBlockedOgrenci() async {  
 final db = await *database*();  
 var sonuc = await db.query(  
 "Ogrenci", where: "blocked = ? ", whereArgs: [1]);  
 List<Ogrenci> list = sonuc.isNotEmpty ? sonuc.map((c) => Ogrenci.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 updateOgrenci(Ogrenci newOgrenci) async {  
 final db = await *database*();  
 var sonuc = await db.update(  
 "Ogrenci", newOgrenci.toMap(), where: "ogrenci\_id = ?",  
 whereArgs: [newOgrenci.ogrenci\_id]);  
 return sonuc;  
 }  
  
 blockOrUnblockOgrenci(Ogrenci ogrenci) async {  
 final db = await *database*();  
 Ogrenci blocked = Ogrenci(ogrenci\_id: ogrenci.ogrenci\_id,  
 ogrenci\_adi: ogrenci.ogrenci\_adi, blocked: !ogrenci.blocked);  
 var sonuc = await db.update(  
 "Ogrenci", blocked.toMap(), where: "ogrenci\_id = ?",  
 whereArgs: [ogrenci.ogrenci\_id]);  
 return sonuc;  
 }  
  
 deleteOgrenci(int ogrenci\_id) async {  
 final db = await *database*();  
 db.delete("Ogrenci", where: "ogrenci\_id = ?", whereArgs: [ogrenci\_id]);  
 db.delete("Oturum", where: "ogrenci\_id = ?", whereArgs: [ogrenci\_id]);  
 }  
  
 deleteAllOgrenci() async {  
 final db = await *database*();  
 db.rawDelete("Delete from Ogrenci");  
 db.rawDelete("Delete from Oturum");  
 }  
  
 newSinav(Sinav newSinav) async {  
 final db = await *database*();  
 var sonuc = await db.insert("Sinav", newSinav.toMap());  
 await oturumEkle(sonuc);  
 return sonuc;  
 }  
  
 Future<List<Sinav>> getAllSinavlar() async {  
 final db = await *database*();  
 var sonuc = await db.query("Sinav");  
 List<Sinav> list = sonuc.isNotEmpty ? sonuc.map((c) => Sinav.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 Future<List<Sinav>> getBlockedSinav() async {  
 final db = await *database*();  
 var sonuc = await db.query("Sinav", where: "blocked = ? ", whereArgs: [0]);  
 List<Sinav> list = sonuc.isNotEmpty ? sonuc.map((c) => Sinav.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 Future<List<Sinav>> getUnblockedSinav() async {  
 final db = await *database*();  
 var sonuc = await db.query("Sinav", where: "blocked = ? ", whereArgs: [1]);  
 List<Sinav> list = sonuc.isNotEmpty ? sonuc.map((c) => Sinav.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 updateSinav(Sinav newSinav) async {  
 final db = await *database*();  
 var sonuc = await db.update(  
 "Sinav", newSinav.toMap(), where: "sinav\_id = ?",  
 whereArgs: [newSinav.sinav\_id]);  
 return sonuc;  
 }  
  
 blockOrUnblockSinav(Sinav sinav) async {  
 final db = await *database*();  
 Sinav blocked = Sinav(sinav\_id: sinav.sinav\_id, sinav\_adi: sinav.sinav\_adi,  
 sinav\_aciklama: sinav.sinav\_aciklama, blocked: !sinav.blocked);  
 var sonuc = await db.update("Sinav", blocked.toMap(), where: "sinav\_id = ?",  
 whereArgs: [sinav.sinav\_id]);  
 return sonuc;  
 }  
  
 deleteSinav(int sinav\_id) async {  
 final db = await *database*();  
 db.delete("Sinav", where: "sinav\_id = ?", whereArgs: [sinav\_id]);  
 db.delete("Oturum", where: "sinav\_id = ?", whereArgs: [sinav\_id]);  
 }  
  
 deleteAllSinav() async {  
 final db = await *database*();  
 db.rawDelete("Delete from Sinav");  
 db.rawDelete("Delete from Oturum");  
 }  
  
 oturumEkle(int sinav\_id) async {  
 final db = await *database*();  
 var ogrencilistesi = await getAllOgrenciler();  
 var ysinav = await db.query(  
 "Sinav", where: "sinav\_id = ?", whereArgs: [sinav\_id]);  
 Sinav sinav = Sinav.fromMap(ysinav.first);  
 for (var ogrenci in ogrencilistesi) {  
 Oturum oturum = Oturum(sinav\_id: sinav.sinav\_id,  
 sinav\_adi: sinav.sinav\_adi,  
 ogrenci\_id: ogrenci.ogrenci\_id,  
 ogrenci\_adi: ogrenci.ogrenci\_adi,  
 blocked: false);  
 try {  
 await newKullanim(oturum);  
 }  
 catch (e) {  
 print('Hatalı işlem');  
 }  
 }  
 }  
  
 newKullanim(Oturum newOturum) async {  
 final db = await *database*();  
 var sonuc = await db.insert("Oturum", newOturum.toMap());  
 return sonuc;  
 }  
  
 Future<List<Oturum>> getAllOturumlar() async {  
 final db = await *database*();  
 var sonuc = await db.query("Oturum");  
 List<Oturum> list = sonuc.isNotEmpty ? sonuc.map((c) => Oturum.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 getSinavOturum(int sinav\_id) async {  
 final db = await *database*();  
 var sonuc = await db.query(  
 "Oturum", where: "sinav\_id = ?", whereArgs: [sinav\_id]);  
 List<Oturum> list = sonuc.isNotEmpty ? sonuc.map((c) => Oturum.fromMap(c))  
 .toList() : [];  
 return list;  
 }  
  
 updateOturum(Oturum newOturum) async {  
 final db = await *database*();  
 var sonuc = await db.update(  
 "Oturum", newOturum.toMap(), where: "oturum\_id = ?",  
 whereArgs: [newOturum.oturum\_id]);  
 return sonuc;  
 }  
  
 blockOrUnblockOturum(Oturum oturum) async {  
 final db = await *database*();  
 Oturum blocked = Oturum(oturum\_id: oturum.oturum\_id,  
 sinav\_id: oturum.sinav\_id,  
 sinav\_adi: oturum.sinav\_adi,  
 ogrenci\_id: oturum.ogrenci\_id,  
 ogrenci\_adi: oturum.ogrenci\_adi,  
 blocked: !oturum.blocked);  
 var sonuc = await db.update(  
 "Oturum", blocked.toMap(), where: "oturum\_id = ?",  
 whereArgs: [oturum.oturum\_id]);  
 return sonuc;  
 }  
  
 oturumKontrol(int sinav\_id, int ogrenci\_id) async {  
 final db = await *database*();  
 var sonuc = await db.query(  
 "Oturum", where: "(sinav\_id = ? and ogrenci\_id = ?)",  
 whereArgs: [sinav\_id, ogrenci\_id]);  
 print(sonuc);  
 if (sonuc.isEmpty) {  
 print("deneme");  
 return "Öğrenci Sınavda Tanımlı Değil";  
 }  
 else {  
 Oturum oturum = Oturum.fromMap(sonuc.first);  
 oturum.blocked = true;  
 var sonuc2 = await db.update(  
 "Oturum", oturum.toMap(), where: "oturum\_id = ?",  
 whereArgs: [oturum.oturum\_id]);  
 return "Öğrenci Sınava Eklendi";  
 }  
 }  
  
 deleteAllOturum() async {  
 final db = await *database*();  
 db.rawDelete("Delete from Oturum");  
 }  
   
}

# Veri Tabanı Tablolar Arası İlişki

