

برنامه نویسی دستگاه های سیار (CE364)

جلسه شانزدهم:
برداشت اطلاعات از پایگاه داده

سجاد شیرعلی شمرضا

پاییز 1401

شنبه، 3 دی 1401

● بخش مرتبط با این جلسه:

- Unit 5: Data persistence:
 - Pathway 1: Introduction to SQL, Room, and Flow



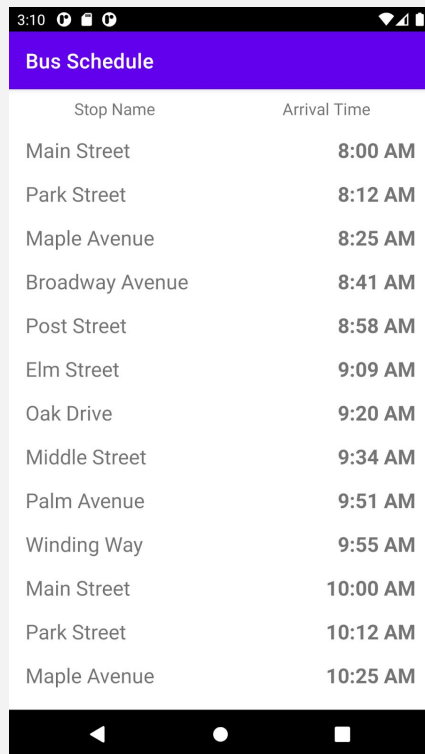
سوال؟

تعریف جدول جهت دسترسی

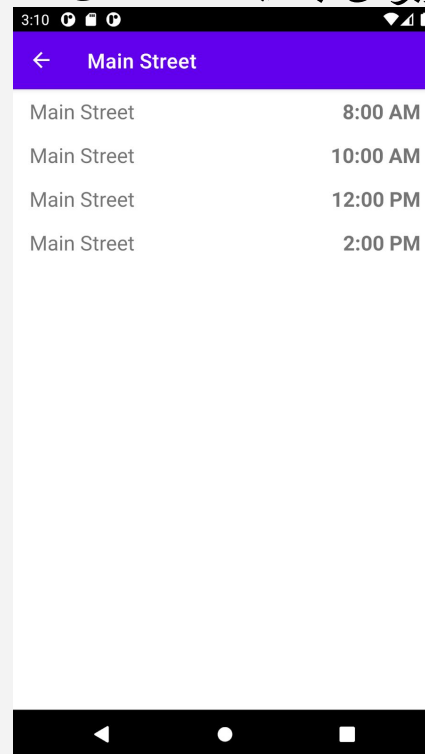
چگونگی تعریف ساختار جدول در برنامه

برنامه نهایی

- ساعت رسیدن اتوبوس به ایستگاه های مختلف



Stop Name	Arrival Time
Main Street	8:00 AM
Park Street	8:12 AM
Maple Avenue	8:25 AM
Broadway Avenue	8:41 AM
Post Street	8:58 AM
Elm Street	9:09 AM
Oak Drive	9:20 AM
Middle Street	9:34 AM
Palm Avenue	9:51 AM
Winding Way	9:55 AM
Main Street	10:00 AM
Park Street	10:12 AM
Maple Avenue	10:25 AM



Main Street	8:00 AM
Main Street	10:00 AM
Main Street	12:00 PM
Main Street	2:00 PM

کتابخانه مورد استفاده

- استفاده از کتابخانه Room برای دسترسی به پایگاه داده
- یک کتابخانه ORM : Object Relational Mapping
- نگاشت جداول پایگاه داده به اشیای قابل استفاده در برنامه

اضافه کردن پیش نیازها به پروژه

```
ext {  
    kotlin_version = "1.3.72"  
    nav_version = "2.3.1"  
    room_version = '2.3.0'  
}
```

```
implementation "androidx.room:room-runtime:$room_version"  
kapt "androidx.room:room-compiler:$room_version"
```

```
// optional - Kotlin Extensions and Coroutines support for Room  
implementation "androidx.room:room-ktx:$room_version"
```

- تعریف نسخه مورد استفاده از کتابخانه
○ در فایل build.gradle
-

تعریف موجودیت (Entity)

- کلاس: قالب تعریف شی
- جدول: قالب تعریف هر شی یا هر سطر
- نمایش هر جدول در قالب یک کلاس
 - در مدل ORM : Object Relational Mapping
 - معروف به موجودیت (Entity) و همچنین کلاس نمونه (Model Class)
- ستون های جدول زمانبندی (schedule) مورد استفاده در برنامه نمونه
 - شناسه (id) : یک عدد صحیح به عنوان شناسه یکتا برای نمایش هر سطر
 - نام ایستگاه (stop_name) : یک رشته برای نمایش نام ایستگاه
 - زمان ورود (arrival_time) : یک عدد صحیح برای نمایش زمان ورود اتوبوس به ایستگاه

تعریف کلاس

- تعریف یک بسته (package) به نام پایگاه داده (database)
- تعریف یک بسته داخل آن برای جدول زمانبندی (schedule)
- تعریف یک کلاس داده در داخل آن برای جدول
 - فایل Schedule.kt
- مشخص کردن کلید اصلی جدول

```
data class Schedule(  
    )
```

```
@PrimaryKey val id: Int
```

- مشخص کردن نام ستون متناظر و اجباری بودن وجود مقدار برای آن

```
@NonNull @ColumnInfo(name = "stop_name") val stopName: String,
```

کلاس زمانبندی تعریف شده

- امکان مشخص کردن کلاس متناظر با
- `@Entity(tableName="table_name_in_db")`

```
@Entity
data class Schedule(
    @PrimaryKey val id: Int,
    @NonNull @ColumnInfo(name = "stop_name") val stopName: String,
    @NonNull @ColumnInfo(name = "arrival_time") val arrivalTime: Int
)
```

کلاس دسترسی به داده

- هدف کلاس دسترسی به داده: تعریف چگونگی دسترسی و تغییر داده
- DAO : Data Access Object
 - برای پایگاه داده: شامل دستورات SQL که باید اجرا شوند
 - لایه انتزاعی بین جزئیات دسترسی به داده و استفاده کننده داده
 - تعریف کلاس برای جدول زمانبندی در بسته پایگاه داده (database.schedule)

```
@Dao  
interface ScheduleDao {  
}
```

تعریف توابع دسترسی به داده

- دریافت زمان ورود اتوبوس برای تمامی ایستگاه ها

```
@Query("SELECT * FROM schedule ORDER BY arrival_time ASC")  
fun getAll(): List<Schedule>
```

- دریافت زمان ورود اتوبوس برای یک ایستگاه خاص
○ ارجاع به متغیر ورودی تابع با اضافه کردن ":" قبل از اسم آن

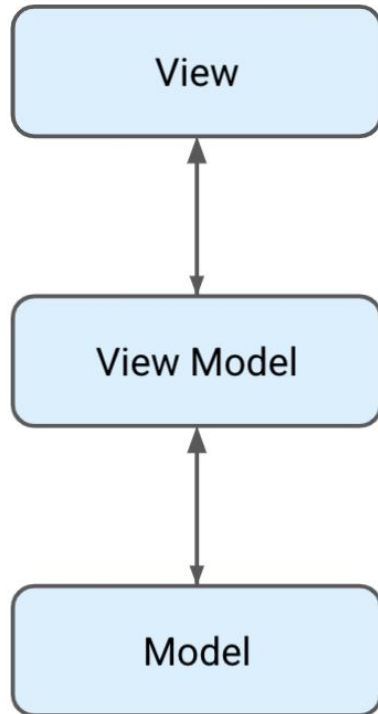
```
@Query("SELECT * FROM schedule WHERE stop_name = :stopName ORDER BY arrival_time ASC")  
fun getByStopName(stopName: String): List<Schedule>
```



سوال؟

مدل نما

مزیت مدل نما (View Model)



- یک الگوی رایج در طراحی برنامه دستگاه سیار
- تفکیک کد مربوط به رابط کاربری (UI) از مدل داده
- کمک به تست مجزای قسمت های مختلف برنامه
- استفاده از مزایای کلاس ViewModel
 - پشتیبانی از تغییرات چرخه زندگی برنامه
 - از بین رفتن به خاطر تغییرات رابط کاربری

تعریف مدل نما

- اضافه کردن یک بسته جدید viewmodels
- اضافه کردن کلاس جدید BusScheduleViewModel

```
class BusScheduleViewModel(private val scheduleDao: ScheduleDao): ViewModel()
```

- تعریف توابع گرفتن اطلاعات

```
fun fullSchedule(): List<Schedule> = scheduleDao.getAll()
```

```
fun scheduleForStopName(name: String): List<Schedule> = scheduleDao.getByStopName(name)
```


تعریف کلاس کارخانه برای آن

```
class BusScheduleViewModelFactory(  
    private val scheduleDao: ScheduleDao  
) : ViewModelProvider.Factory {  
}
```

- گرفتن مدل نما از طریق کلاس کارخانه

- تعریف تابع گرفتن مدل نما

```
override fun <T : ViewModel> create(modelClass: Class<T>): T {  
    if (modelClass.isAssignableFrom(BusScheduleViewModel::class.java)) {  
        @Suppress("UNCHECKED_CAST")  
        return BusScheduleViewModel(scheduleDao) as T  
    }  
    throw IllegalArgumentException("Unknown ViewModel class")  
}
```

مشخص کردن ارتباط بین این کلاس ها

- اهداف تعریف کلاس پایگاه داده (AppDatabase)
 - مشخص کردن موجودیت های تعریف شده در پایگاه داده
 - فراهم آوردن دسترسی به یک کلاس دسترسی داده برای هر نوع داده
 - انجام مراحل اضافه از قبیل مقداردهی اولیه به پایگاه داده

تعریف کلاس پایگاه داده

- تعریف کلاس در بسته پایگاه داده

```
abstract class AppDatabase: RoomDatabase() {  
}
```

- تعریف تابع گرفتن کلاس دسترسی به داده

```
abstract fun scheduleDao(): ScheduleDao
```

پیاده سازی تنها یک نمونه از کلاس

- تعریف یک شی همراه (Companion Object) برای اینکه تنها یک نمونه از آن به وجود آید

```
companion object {  
}
```

```
@Volatile  
private var INSTANCE: AppDatabase? = null
```

- تعریف یک نمونه از شی
○ در ابتدا، مقدار ندارد

پیاده سازی تنها یک نمونه از کلاس (ادامه)

```
fun getDatabase(context: Context): AppDatabase {  
    return INSTANCE ?: synchronized(this) {  
        val instance = Room.databaseBuilder(  
            context,  
            AppDatabase::class.java,  
            "app_database")  
            .createFromAsset("database/bus_schedule.db")  
            .build()  
        INSTANCE = instance  
  
        instance  
    }  
}
```

- تابع گرفتن پایگاه داده
- مقدار دهی اولیه به پایگاه داده

تعریف موجودیت ها و نسخه پایگاه داده

- تعریف لیست موجودیت های مورد استفاده

```
@Database(entities = arrayOf(Schedule::class), version = 1)
```

- تعریف نسخه پایگاه داده
 - افزایش آن با تغییر ساختار پایگاه داده (schema change)
 - تعیین چگونگی به روز رسانی پایگاه داده

تعریف متغیر پایگاه داده در برنامه

- تعریف یک زیر کلاس جدید از کلاس Application
- تعریف یک متغیر از نوع تاخیری (lazy) در آن

```
class BusScheduleApplication : Application() {  
    val database: AppDatabase by lazy { AppDatabase.getDatabase(this) }  
}
```

- استفاده از این کلاس برای برنامه اصلی در فایل AndroidManifest.xml

```
<application  
    android:name="com.example.busschedule.BusScheduleApplication"  
    ...  
>
```

تعریف مبدل

- تعریف مبدل (Adapter) برای RecyclerView
- استفاده از ListAdapter

```
class BusStopAdapter(private val onItemClick: (Schedule) -> Unit) :  
    ListAdapter<Schedule, BusStopAdapter.BusStopViewHolder>(DiffCallback) {  
}
```


تعریف کلاس نگه دارنده نما

```
class BusStopViewHolder(private var binding: BusStopItemBinding):  
    RecyclerView.ViewHolder(binding.root) {  
  
    @SuppressWarnings("SimpleDateFormat")  
    fun bind(schedule: Schedule) {  
        binding.stopNameTextView.text = schedule.stopName  
        binding.arrivalTimeTextView.text = SimpleDateFormat(  
            "h:mm a").format(Date(schedule.arrivalTime.toLong() * 1000)  
        )  
    }  
}
```

تابع ایجاد نگه دارنده نما

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BusStopViewHolder {  
    val viewHolder = BusStopViewHolder(  
        BusStopItemBinding.inflate(  
            LayoutInflater.from( parent.context),  
            parent,  
            false  
        )  
    )  
    viewHolder.itemView.setOnClickListener {  
        val position = viewHolder.adapterPosition  
        onItemClick(getItem(position))  
    }  
    return viewHolder  
}  
  
override fun onBindViewHolder(holder: BusStopViewHolder, position: Int) {  
    holder.bind(getItem(position))  
}
```

مقایسه دو زمان ورود اتوبوس

```
companion object {  
    private val DiffCallback = object : DiffUtil.ItemCallback<Schedule>() {  
        override fun areItemsTheSame(oldItem: Schedule, newItem: Schedule): Boolean {  
            return oldItem.id == newItem.id  
        }  
  
        override fun areContentsTheSame(oldItem: Schedule, newItem: Schedule): Boolean {  
            return oldItem == newItem  
        }  
    }  
}
```

ایجاد کردن مدیر نما در صفحه اصلی

- ایجاد و آماده سازی مدیر نمای RecyclerView در تابع onViewCreated

```
recyclerView = binding.recyclerView  
recyclerView.layoutManager = LinearLayoutManager(requireContext())
```

- متصل کردن مبدل

```
val busStopAdapter = BusStopAdapter({  
    val action = FullScheduleFragmentDirections  
        .actionFullScheduleFragmentToStopScheduleFragment(  
            stopName = it.stopName  
        )  
    view.findNavController().navigate(action)  
})  
recyclerView.adapter = busStopAdapter
```

ایجاد کردن مدیر نما در صفحه اصلی (ادامه)

- دریافت زمان ورود اتوبوس ها از پایگاه داده
 - انجام آن در یک کوروتین برای جلوگیری از قفل شدن رابط کاربری در حین آن

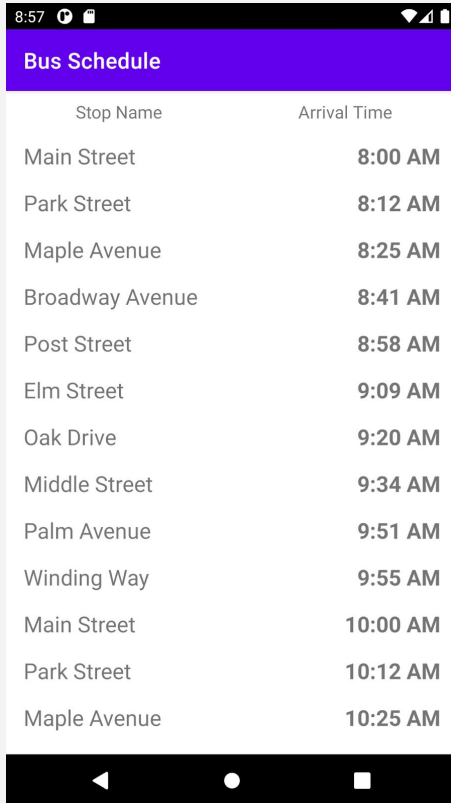
```
// submitList() is a call that accesses the database. To prevent the
// call from potentially locking the UI, you should use a
// coroutine scope to launch the function. Using GlobalScope is not
// best practice, and in the next step we'll see how to improve this.
GlobalScope.launch(Dispatchers.IO) {
    busStopAdapter.submitList(viewModel.fullSchedule())
}
```


ایجاد کردن مدیر نما در صفحه دوم

- عدم نیاز به انجام کاری در صورت کلیک کردن در صفحه دوم

```
recyclerView = binding.recyclerView
recyclerView.layoutManager = LinearLayoutManager(requireContext())
val busStopAdapter = BusStopAdapter({})
recyclerView.adapter = busStopAdapter
// submitList() is a call that accesses the database. To prevent the
// call from potentially locking the UI, you should use a
// coroutine scope to launch the function. Using GlobalScope is not
// best practice, and in the next step we'll see how to improve this.
GlobalScope.launch(Dispatchers.IO) {
    busStopAdapter.submitList(viewModel.scheduleForStopName(stopName))
}
```

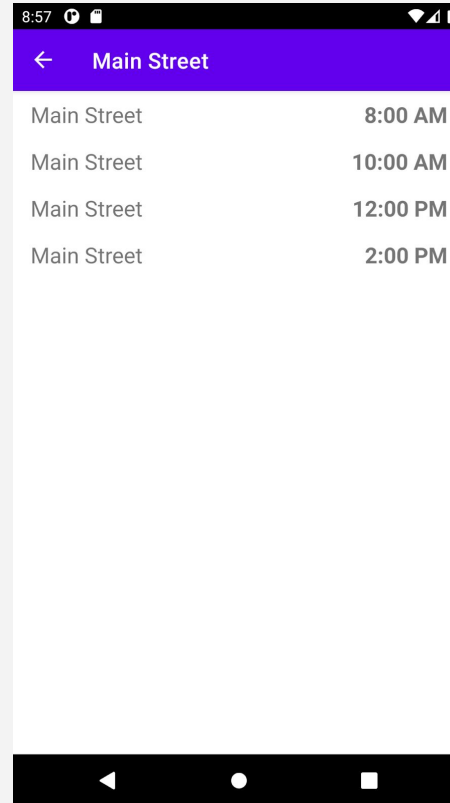
برنامه نهایی



8:57

Bus Schedule

Stop Name	Arrival Time
Main Street	8:00 AM
Park Street	8:12 AM
Maple Avenue	8:25 AM
Broadway Avenue	8:41 AM
Post Street	8:58 AM
Elm Street	9:09 AM
Oak Drive	9:20 AM
Middle Street	9:34 AM
Palm Avenue	9:51 AM
Winding Way	9:55 AM
Main Street	10:00 AM
Park Street	10:12 AM
Maple Avenue	10:25 AM



8:57

← Main Street

Main Street	8:00 AM
Main Street	10:00 AM
Main Street	12:00 PM
Main Street	2:00 PM



سوال؟

به روزرسانی برنامه با تغییر داده

مشکل فعلی

- عدم به روزرسانی برنامه در صورت تغییر داده در پایگاه داده
- مثال: اجرای دستور زیر در Database Inspector

```
INSERT INTO schedule  
VALUES (null, 'Winding Way', 1617202500)
```

- مشکل؟ لیست ورود اتوبوس ها تنها یکبار گرفته می شود
- برای به روزرسانی نیاز به اجرای مجدد برنامه است
- راه حل: استفاده از ویژگی Flow در زبان کاتلین
- ارسال پیوسته اطلاعات از DAO

استفاده از Flow

- تغییر خروجی تابع `getAll` و `getByStopName`

```
fun getAll(): Flow<List<Schedule>>
```

```
fun getByStopName(stopName: String): Flow<List<Schedule>>
```

- تغییر تابع استفاده کننده از آنها

```
class BusScheduleViewModel(private val scheduleDao: ScheduleDao): ViewModel() {  
  
    fun fullSchedule(): Flow<List<Schedule>> = scheduleDao.getAll()  
  
    fun scheduleForStopName(name: String): Flow<List<Schedule>> = scheduleDao.get  
}
```

تغییر نما هنگام تغییر داده

- نیاز به استفاده از کوروتین
- جایگزینی

```
busStopAdapter.submitList(viewModel.fullSchedule())
```

- با

```
lifecycle.coroutineScope.launch {  
    viewModel.fullSchedule().collect() {  
        busStopAdapter.submitList(it)  
    }  
}
```



سوال؟