LAB 6

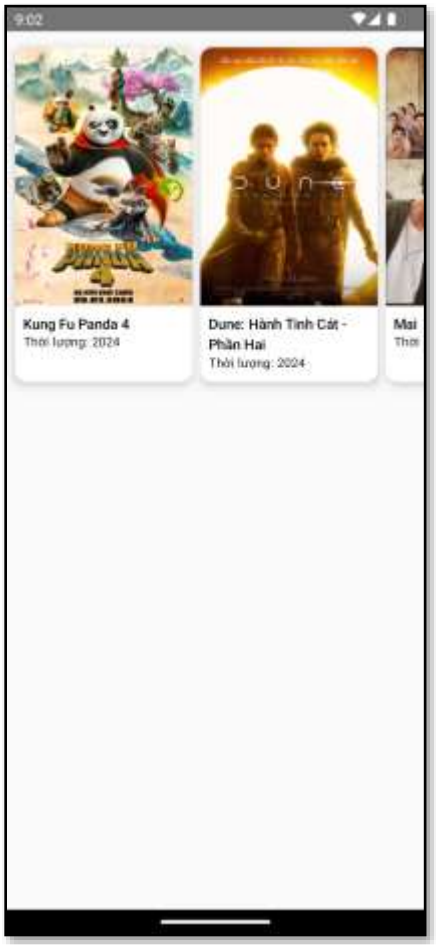
MỤC TIÊU

Kết thúc bài thực hành sinh viên có khả năng:

- ✓ Biết cách sử dụng listview và các dạng listview.
- ✓ Biết cách tạo tùy chỉnh giao diện cho từng dạng listview.
- ✓ Biết cách tạo package và cấu trúc bố cục của project.

NỘI DUNG

BÀI 1: Xây dựng giao diện hiển thị danh sách các bộ phim.



❖ Yêu cầu giao diện:

- Hiển thị danh sách phim theo dạng hàng ngang.
- Hiển thị hình ảnh của phim từ 1 liên kết trong Card.

❖ Hướng dẫn và code tham khảo:

- Tạo 1 data class là Movie và danh sách Movie mẫu như sau:

```
data class Movie(
    val title: String,
    val year: String,
    val posterUrl: String
) {
    companion object {
        fun getSampleMovies() = listOf()
    }
}
```

- Dữ liệu mẫu trong file đính kèm.
- Để hiển thị hình ảnh từ URL cần sử dụng thư viện **coil** thêm thư viện vào trong **build/gradle.kts** (module)

```
implementation("io.coil-kt:coil-compose:2.6.0")
```

- Cấp quyền truy cập internet cho ứng dụng thêm 2 premission sau vào AndroidManifest.xml.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Tạo MovieItem để hiển thị thông tin phim và hình ảnh:

```
@Composable
fun MovieItem(movie: Movie) {
    Card(
        colors = CardDefaults.cardColors(containerColor = Color.White),
        elevation = CardDefaults.cardElevation(defaultElevation = 6.dp),
    ) {
        Column(
            modifier = Modifier.width(175.dp).height(330.dp)
        ) {
            AsyncImage(
                model = movie.posterUrl,
                contentScale = ContentScale.Crop,
                contentDescription = null,
                modifier = Modifier
                    .height(255.dp)
                    .fillMaxWidth()
```

```
                    .clip(RoundedCornerShape(topEnd = 8.dp, topStart =
8.dp)),
            )

            Column(modifier = Modifier.padding(8.dp)) {
                Text(text = movie.title, style =
MaterialTheme.typography.titleSmall, maxLines = 2)
                Text(text = "Thời lượng: ${movie.year}", style =
MaterialTheme.typography.bodySmall)
            }
        }
    }
}
```

- **AsyncImage** trong Jetpack Compose giúp hiển thị hình ảnh từ nguồn như URL hay tài nguyên. Dưới đây là các thuộc tính chính:
  - **model**: Nguồn hình ảnh cần hiển thị.
  - **contentDescription**: Mô tả về hình ảnh, hỗ trợ, trợ năng.
  - **contentScale**: Cách hình ảnh được điều chỉnh để khớp với kích thước **composable**.
  - **modifier**: Hiệu chỉnh **layout** và vẽ, như cắt hoặc thiết lập kích thước.
  - **placeholder**: Hình ảnh hiển thị khi hình chính đang tải.
  - **error**: Hình ảnh hiển thị nếu có lỗi tải hình chính.
  - **onLoading**: **Callback** khi bắt đầu tải hình ảnh.
  - **onSuccess**: **Callback** khi hình ảnh tải và hiển thị thành công.
  - **onError**: **Callback** khi có lỗi trong quá trình tải hoặc hiển thị hình ảnh.
- Thêm MovieItem vào LazyRow().

```
@Composable
fun MovieScreen(movies: List<Movie>) {
    LazyRow(
        state = rememberLazyListState(),
        contentPadding = PaddingValues(horizontal = 8.dp, vertical = 16.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(movies.size) { index ->
            MovieItem(movie = movies[index])
        }
    }
}
```

- **LazyRow** trong Jetpack Compose dùng để hiển thị danh sách các item cuộn ngang. Dưới đây là các thuộc tính chính:
  - **state**: Để kiểm soát và truy xuất vị trí cuộn.
  - **contentPadding**: Padding xung quanh nội dung bên trong.
  - **horizontalArrangement**: Sắp xếp các item theo chiều ngang.
  - **reverseLayout**: Hiển thị item theo thứ tự ngược lại.
  - **flingBehavior**: Tùy chỉnh cách cuộn nhanh phản ứng.
  - **content**: Định nghĩa nội dung bên trong, bao gồm các item.
  - **verticalAlignment**: Căn chỉnh các item theo chiều dọc.
- Thêm hàm MovieScreen vào setContent:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MovieScreen(Movie.getSampleMovies())
        }
    }
}
```

- Chạy ứng dụng và quan xác kết quả.

Bài 2: Tiếp tục sử dụng bài 1, thiết ké giao diện hiển thị danh sách phim theo các dạng: hàng, cột và lưới động.

Hàng

Cột

Lưới

❖ Yêu cầu giao diện:

- Sử dụng **MovieScreen** để tạo giao diện chính, cho phép người dùng chọn giữa các loại danh sách: **Row**, **Column** và **Grid**.
- Hiển thị ba nút để người dùng có thể chuyển đổi giữa các loại danh sách.
- Hiển thị danh sách phim dạng **Row**:
  - Khi người dùng chọn "**Row**", sử dụng **LazyRow** để hiển thị danh sách phim theo hàng ngang.
  - Mỗi phim sẽ được hiển thị trong một **Card** và sử dụng **MovieItem** để hiển thị thông tin cụ thể.

- Hiển thị danh sách phim dạng **Column**:
  - Khi chọn "**Column**", sử dụng **LazyColumn** để hiển thị danh sách phim theo cột.
  - Sử dụng **MovieColumnItem** để hiển thị thông tin của mỗi phim, bao gồm ảnh, tiêu đề, thời lượng, khởi chiếu và tóm tắt.
- Hiển thị danh sách phim dạng **Staggered Grid**:
  - Khi chọn "**Grid**", sử dụng **LazyVerticalStaggeredGrid** với **StaggeredGridCells.Fixed(2)** để tạo lưới có hai cột.
  - Mỗi phim cũng sẽ được hiển thị trong một **Card** sử dụng **MovieItem**.

❖ Hướng dẫn và code tham khảo:

- Tạo enum ListType để quản lý trạng thái của danh sách.

```
enum class ListType {
    ROW, COLUMN, GRID
}
```

- Tạo danh sách các button để chuyển đổi giữa các dạng danh sách:

```
@Composable
fun MovieScreen(movies: List<Movie>) {
    var listType by remember { mutableStateOf(ListType.ROW) }
    Column {
        Row(
            modifier = Modifier.padding(8.dp).fillMaxWidth(),
            horizontalArrangement = Arrangement.Center
        ) {
            Button(onClick = { listType = ListType.ROW }) {
                Text("Row")
            }
            Spacer(modifier = Modifier.width(8.dp))
            Button(onClick = { listType = ListType.COLUMN }) {
                Text("Column")
            }
            Spacer(modifier = Modifier.width(8.dp))
            Button(onClick = { listType = ListType.GRID }) {
                Text("Grid")
            }
        }
        when (listType) {
            ListType.ROW -> MovieRow(movies)
            ListType.COLUMN -> MovieColumn(movies)
            ListType.GRID -> MovieGrid(movies)
        }
    }
}
```

- Tạo hàm MovieRow:

```
@Composable
fun MovieRow(movies: List<Movie>) {
    LazyRow(
        state = rememberLazyListState(),
        contentPadding = PaddingValues(horizontal = 8.dp, vertical = 16.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(movies.size) { index ->
            MovieItem(movie = movies[index], listType = ListType.ROW)
        }
    }
}
```

- Tạo hàm MovieColumn:

```
@Composable
fun MovieColumn(movies: List<Movie>) {
    LazyColumn(
        state = rememberLazyListState(),
        contentPadding = PaddingValues(horizontal = 8.dp, vertical = 16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(movies.size) { index ->
            MovieColumnItem(movie = movies[index], listType =
ListType.COLUMN)
        }
    }
}
```

- Tạo hàm MovieGird:

```
@Composable
fun MovieGrid(movies: List<Movie>) {
    val gridState = rememberLazyStaggeredGridState()

    LazyVerticalStaggeredGrid(
        columns = StaggeredGridCells.Fixed(2),
        state = gridState,
        horizontalArrangement = Arrangement.spacedBy(8.dp),
        verticalItemSpacing = 8.dp,
        contentPadding = PaddingValues(8.dp)
    ) {
        items(movies.size) { index ->
            MovieItem(movie = movies[index], listType = ListType.GRID)
        }
    }
}
```

- **LazyVerticalStaggeredGrid** là một composable trong Jetpack Compose dùng để hiển thị danh sách các item theo dạng lưới (grid) nơi mà các hàng có thể có chiều cao khác nhau, tạo ra một hiệu ứng "**staggered**" hoặc "**masonry**". Dưới đây là một số thuộc tính chính của nó:
  - **columns**: Định nghĩa cách phân chia các cột trong lưới. **StaggeredGridCells.Fixed(count)** xác định số cột cố định, trong khi **StaggeredGridCells.Adaptive(minSize)** cho phép điều chỉnh số cột dựa trên kích thước tối thiểu của item.
  - **state**: Một instance của **LazyStaggeredGridState** giúp quản lý và truy cập trạng thái cuộn của lưới.
  - **contentPadding**: Định nghĩa khoảng cách padding xung quanh nội dung của lưới.
  - **horizontalArrangement**: Xác định cách các item được sắp xếp và khoảng cách giữa chúng theo chiều ngang.
  - **verticalItemSpacing**: Khoảng cách giữa các item theo chiều dọc.
  - **content**: Định nghĩa nội dung bên trong của **LazyVerticalStaggeredGrid**, thường là một danh sách các composable item được hiển thị.
- Cập nhật hàm MovieItem để nó có thể tuỳ chỉnh theo các dạng danh sách:

```
@Composable
fun MovieItem(movie: Movie, listType: ListType) {
    Card(
        colors = CardDefaults.cardColors(containerColor =
Color.White),
        elevation = CardDefaults.cardElevation(defaultElevation =
6.dp),
    ) {
        Column(
            modifier = Modifier.then(getItemSizeModifier(listType))
        ) {
            AsyncImage(
                model = movie.posterUrl,
                contentDescription = null,
                contentScale = ContentScale.FillWidth,
                modifier = Modifier.wrapContentHeight().fillMaxWidth()
            )

            Column(
                modifier = Modifier.padding(8.dp)
            ) {
```

```
                Text(
                    text = movie.title,
                    style = MaterialTheme.typography.titleSmall,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis
                )
                BoldValueText(label = "Thời lượng: ", value =
movie.duration)
                BoldValueText(label = "Khởi chiếu: ", value =
movie.releaseDate)
            }
        }
    }
}
```

- Ở dang Cột thi cấu trúc của các Card khác so với ở dạng Hàng và Lưới nên cần tạo 1 hàm mới dành cho dạng Cột là **MovieItemColumn.**

```
@Composable
fun MovieColumnItem(movie: Movie, listType: ListType) {
    Card(
        colors = CardDefaults.cardColors(containerColor =
Color.White),
        elevation = CardDefaults.cardElevation(defaultElevation =
6.dp),
    ) {
        Row(
            modifier = Modifier.fillMaxWidth()
        ) {
            AsyncImage(
                model = movie.posterUrl,
                contentDescription = null,
                contentScale = ContentScale.FillWidth,
                modifier = Modifier
                    .then(getItemSizeModifier(listType))
                    .wrapContentHeight()
            )

            Column(
                modifier = Modifier.padding(8.dp)
            ) {
                Text(
                    text = movie.title,
                    style = MaterialTheme.typography.titleSmall,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis
                )
                BoldValueText(label = "Thời lượng: ", value =
movie.duration)
                BoldValueText(label = "Khởi chiếu: ", value =
movie.releaseDate)
                BoldValueText(label = "Thể loại: ", value =
```

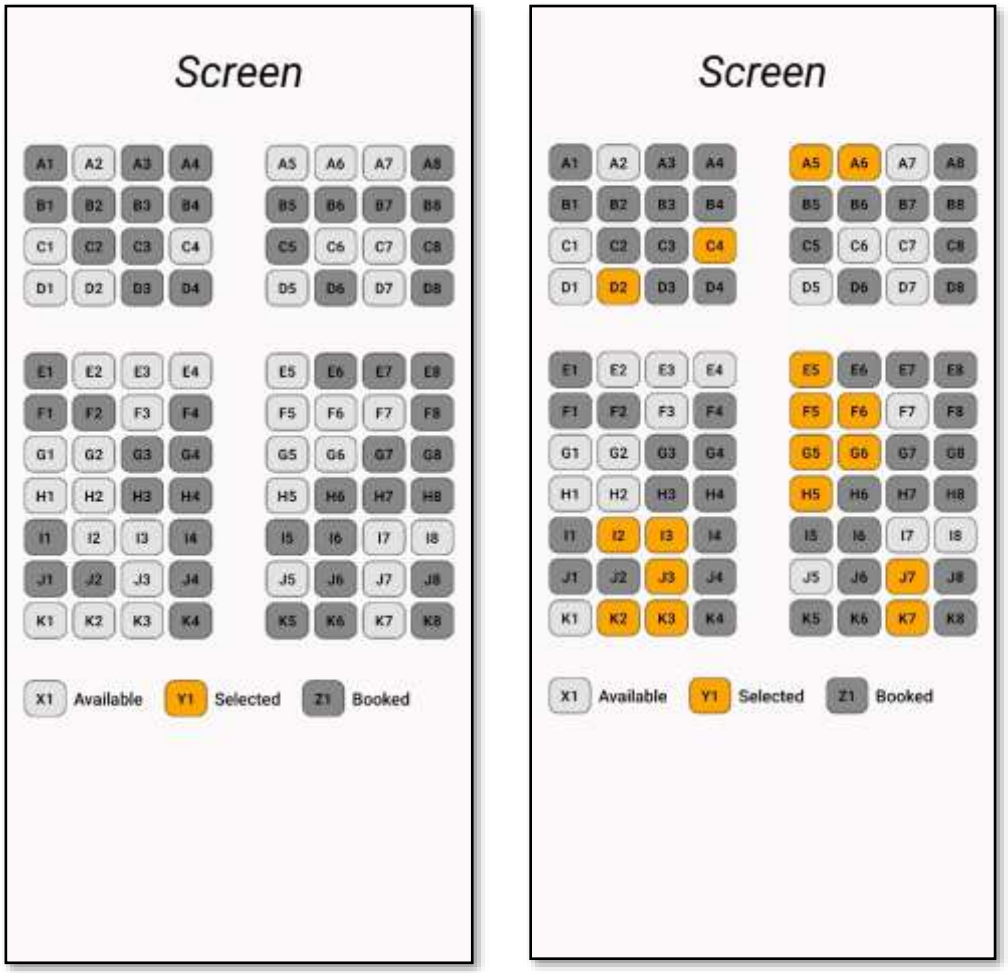
```
movie.genre)
                Text(
                    text = "Tóm tắt nội dung",
                    style = MaterialTheme.typography.bodySmall,
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier.padding(top = 4.dp, bottom =
2.dp)
                )
                Text(
                    text = movie.shotDescription,
                    style = MaterialTheme.typography.bodySmall,
                    maxLines = 5,
                    overflow = TextOverflow.Ellipsis,
                    modifier = Modifier.padding(end = 2.dp)
                )
            }
        }
    }
}
```

- Và các hàm hỗ trợ khác:

```
@Composable
fun BoldValueText(label: String, value: String, style: TextStyle =
MaterialTheme.typography.bodySmall
) {
    Text(buildAnnotatedString {
        append(label)
        withStyle(style = SpanStyle(fontWeight = FontWeight.Bold)) {
            append(value)
        }
    }, style = style)
}


@Composable
private fun getItemSizeModifier(listType: ListType): Modifier {
    return when (listType) {
        ListType.ROW -> Modifier.width(175.dp)
        ListType.COLUMN -> Modifier
            .width(130.dp)

        ListType.GRID -> Modifier
            .fillMaxWidth()
    }
}
```

- Cập nhật lại lớp Movie và dữ liệu mẫu theo file đính kèm.
- Chạy ứng dụng và quan xác kết quả

Bài 3: Tạo danh sách các chỗ ngồi trong 1 rạp chiếu phim từ 1 sơ đồ chỗ ngồi.



❖ Yêu cầu giao diện:

- Hiển thị vị trí của màn chiếu nằm ở trên cùng.
- Phía dưới là sơ đồ danh sách chỗ ngồi.
- Chỗ ngồi có 3 trạng thái là trống, đang chọn, và đã được đặt.
- Nếu ghế còn trống thì có thể chọn, khi ghế đã được chọn khi chọn lại sẽ trở về trống, còn ghế đã được đặt thì không thể chọn.
- Ghế còn trống có màu xám sáng opacity 0.5, đã chọn có màu cam, còn đã được đặt thì là màu xám.
- Hiển thị vị trí lối đi và ghế nằm xác lối đi.

❖ Yêu cầu bố cục thư mục theo cấu trúc sau:

```
com/example/lab6_ps23456_3/
├── activity/
│   └── MainActivity.kt
├── ui/
│   ├── components/
│   │   └── SeatComposable.kt
│   ├── screens/
│   │   └── CinemaSeatBooking.kt
│   └── theme/
│       ├── Theme.kt
│       ├── Type.kt
│       └── Color.kt
├── model/
│   └── entities/
│       ├── Seat.kt
│       └── SeatStatus.kt
└── utils/
    └── TheaterSeatingUtil.kt
```

- Trong đó:
  - **ui** - Thư mục này chứa tất cả các thành phần giao diện người dùng:
    - **components** - Chứa các **Composable** tái sử dụng như **SeatComposable**.
    - **screens** - Chứa các màn hình hoặc phần lớn của UI, ví dụ như **CinemaSeatBooking**.
    - **theme** - Chứa các định nghĩa cho **theme**, **style**, và các tài nguyên giao diện người dùng khác.
  - **model** - Thư mục cho các đối tượng dữ liệu:
  - **entities** - Định nghĩa các đối tượng như **Seat** và **SeatStatus**.
  - **util** - Chứa các hàm tiện ích, ví dụ như **createTheaterSeating**.
  - **activity** - Chứa các **activity** của ứng dụng, như **MainActivity**.

❖ Hướng dẫn và code tham khảo:

- Tạo Thư Mục Mới:
  - Click chuột phải vào package (ví dụ: **com.example.lab6_ps23456_3**) nơi bạn muốn tạo thư mục mới.

- Trong menu ngữ cảnh, chọn **New** > **Package**.
- Nhập tên cho **package** mới và nhấn **Enter**. Ví dụ: **ui**, **model**, **utils**, v.v.

- Tạo Các **Subpackage**:
  - Lặp lại quá trình tương tự để tạo các **subpackage** như **ui/components**, **ui/screens**, **model/entities**, v.v. Bạn cần click chuột phải vào **package** ui để tạo **components** và **screens**, và làm tương tự với **model** để tạo **entities**.

- Tạo 2 model class là Seat và StatusSeat:

```
// in /model/entieies/Seat.kt
data class Seat(var row: Char, val number: Int, var status:
SeatStatus)

// in /model/entieies/SeatStatus.kt
enum class SeatStatus { EMPTY, SELECTED, BOOKED, AISLE }
```

- Tiến hành tạo giao diện cho từng Seat:
  - Tạo file SeatComposable.kt và hàm SeatComposable:

```
@Composable
fun SeatComposable(seat: Seat, clickable: Boolean = true) {
    var status by remember { mutableStateOf(seat.status) }

    val backgroundColor = when (status) {
        SeatStatus.EMPTY -> Color.LightGray.copy(alpha = 0.5f)
        SeatStatus.SELECTED -> Color(0xFFFFA500)
        SeatStatus.BOOKED -> Color.Gray
        SeatStatus.AISLE -> Color.Transparent
    }

    val borderModifier = if (status != SeatStatus.AISLE) {
        Modifier.border(
            BorderStroke(1.dp, Color.DarkGray.copy(alpha =
0.8f)),
            shape = RoundedCornerShape(8.dp)
        )
    } else Modifier

    Box(
        modifier = Modifier
            .padding(2.dp)
            .size(width = 35.dp, height = 30.dp)
            .then(borderModifier)
            .clip(RoundedCornerShape(8.dp))
            .background(backgroundColor)
            .padding(if (seat.status != SeatStatus.AISLE) 3.dp
else 0.dp)
            .clickable(enabled = clickable && (status ==
```

```
SeatStatus.EMPTY || status == SeatStatus.SELECTED)) {
                status = if (status == SeatStatus.EMPTY)
SeatStatus.SELECTED else SeatStatus.EMPTY
            },
        contentAlignment = Alignment.Center
    ) {
        if (seat.status != SeatStatus.AISLE) {
            Text(
                text = "${seat.row}${seat.number}",
                style = MaterialTheme.typography.bodySmall,
                fontWeight = FontWeight.SemiBold
            )
        }
    }
}
```

- Tạo PreviewSeat để quan xác tran thái của Seat:

```
@Preview
@Composable
fun PreviewSeat() {
    Row {
        SeatComposable(Seat('X', 1, SeatStatus.EMPTY))
        SeatComposable(Seat('Y', 1, SeatStatus.SELECTED))
        SeatComposable(Seat('Z', 1, SeatStatus.BOOKED))
    }
}
```

- Tiến hành tạo danh sách ghế sử dụng **LazyVerticalGird**:
  - Tạo file và hàm **CinemaSeatBookingScreen**:

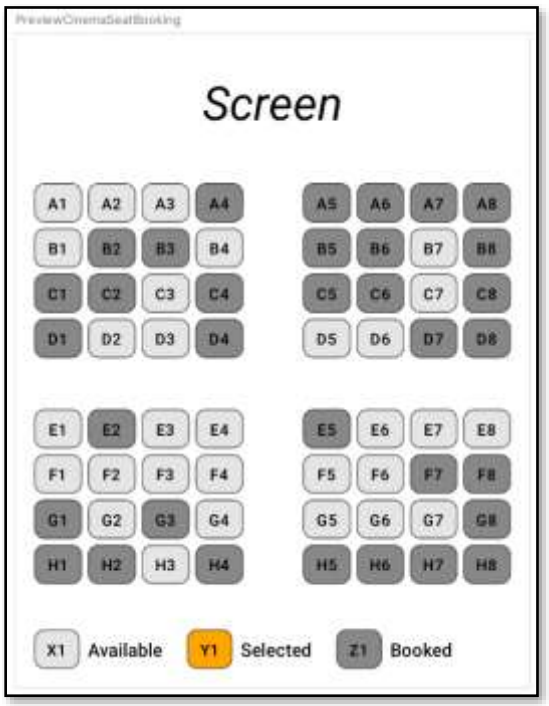
```
@Composable
fun CinemaSeatBookingScreen(seats: List<Seat>, totalSeatsPerRow:
Int) {
    val textModifier = Modifier.padding(end = 16.dp, start =
4.dp)

    Column(horizontalAlignment = Alignment.CenterHorizontally,
modifier = Modifier.padding(12.dp)) {
        Text(
            "Screen",
            modifier = Modifier.padding(16.dp),
            fontStyle = FontStyle.Italic,
            style = MaterialTheme.typography.displaySmall
```

```
        )

        Spacer(modifier = Modifier.height(20.dp))

        LazyVerticalGrid(columns =
GridCells.Fixed(totalSeatsPerRow)) {
            items(seats.size) { index ->
                SeatComposable(seat = seats[index])
            }
        }

        Spacer(modifier = Modifier.height(30.dp))

        Row(modifier = Modifier.fillMaxWidth(),
verticalAlignment = Alignment.CenterVertically) {
            val exampleEmptySeat = remember { Seat('X', 1,
SeatStatus.EMPTY) }
            val exampleSelectedSeat = remember { Seat('Y', 1,
SeatStatus.SELECTED) }
            val exampleBookedSeat = remember { Seat('Z', 1,
SeatStatus.BOOKED) }

            SeatComposable(exampleEmptySeat, false)
            Text(
                text = "Available",
                style = MaterialTheme.typography.titleSmall,
                modifier = textModifier
            )

            SeatComposable(exampleSelectedSeat, false)
            Text(
                text = "Selected",
                style = MaterialTheme.typography.titleSmall,
                modifier = textModifier
            )

            SeatComposable(exampleBookedSeat, false)
            Text(
                text = "Booked",
                style = MaterialTheme.typography.titleSmall,
                modifier = textModifier
            )
        }
    }
}
```

- Tạo hàm PreviewCinemaBookingSeatScreen:

```
const val totalRows = 9
const val totalSeatsPerRow = 9
const val aislePositionInRow = 4
const val aislePositionInColumn = 5
```

```
@Preview
@Composable
fun PreviewCinemaSeatBooking() {
    CinemaSeatBookingScreen(
        createTheaterSeating(
            totalRows,
            totalSeatsPerRow,
            aislePositionInRow,
            aislePositionInColumn
        ), totalSeatsPerRow
    )
}
```



- Viết hàm tạo ra danh sách Seat có trạng thái Seat ngẫu nhiên:
    - Tạo file và hàm createTheaterSeating:

```
fun createTheaterSeating(
    totalRows: Int,
    totalSeatsPerRow: Int,
    aislePositionInRow: Int,
    aislePositionInColumn: Int
): List<Seat> {
    val seats = mutableListOf<Seat>()
    for (rowIndex in 0 until totalRows) {
        for (seatIndex in 1..totalSeatsPerRow) {
            val adjustedRowIndex = if (rowIndex >=
aislePositionInRow) rowIndex - 1 else rowIndex
            val adjustedSeatIndex =
```

```
                if (seatIndex >= aislePositionInColumn)
seatIndex - 1 else seatIndex

            val isAisleRow = rowIndex == aislePositionInRow
            val isAisleColumn = seatIndex ==
aislePositionInColumn

            val status = when {
                isAisleRow || isAisleColumn -> SeatStatus.AISLE
                else -> if (Random.nextInt(0, 99) % 2 == 0)
SeatStatus.BOOKED else SeatStatus.EMPTY
            }

            seats.add(
                Seat(
                    row = 'A' + adjustedRowIndex,
                    number = adjustedSeatIndex,
                    status = status
                )
            )
        }
    }
    return seats
}
```

- Sử dụng hàm **CinemaSeatBookingScreen** trong **MainActivity.kt** sử dụng hàm **createTheaterSeating** để tạo ra danh sách ghế mẫu.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            CinemaSeatBookingScreen(
                createTheaterSeating(
                    totalRows = 12,
                    totalSeatsPerRow = 9,
                    aislePositionInRow = 4,
                    aislePositionInColumn = 5
                ), totalSeatsPerRow = 9
            )
        }
    }
}
```

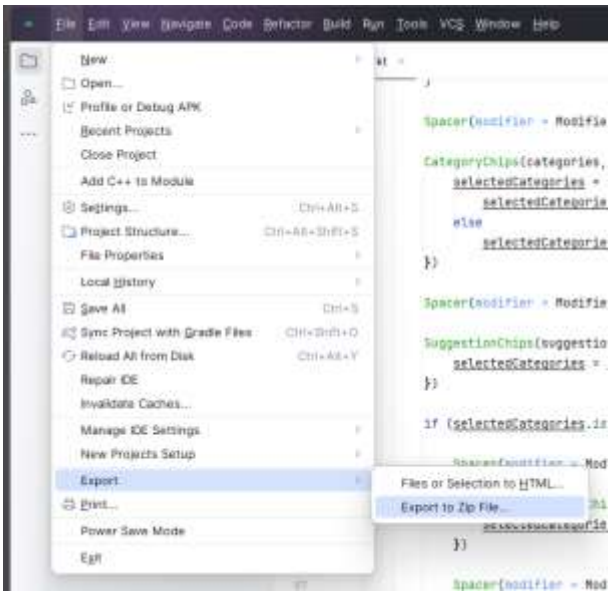
- Chạy ứng dụng và quan xác kế quả.

BÀI 4: GV CHO THÊM

*** YÊU CẦU NỘP BÀI:

Sv nén file bao gồm các yêu cầu đã thực hiện trên, nộp lms đúng thời gian quy định của giảng viên. Không nộp bài coi như không có điểm.

Hướng dẫn nén project: **File** > **Export** > **Export to Zip File**.



--- Hết ---