LAB 8

MỤC TIÊU

Kết thúc bài thực hành sinh viên có khả năng:

- ✓ Tương tác với API
- ✓ Retrofit

NỘI DUNG

BÀI 1: XÂY DỰNG CHỨC NĂNG HIỂN THỊ DANH SÁCH PHIM ĐƯỢC LẤY TỪ API

Tiếp tục sử dụng project Lab 7, nâng cấp chức năng

Reponse API trả về danh sách phim có dạng

```
[
  {
    "filmId": String,
    "filmName": String,
    "duration": String,
    "releaseDate": String,
    "genre": String,
    "national": String,
    "description": String,
    "image": String
  },
  {
        ...
  }
]
```

Bước 1: Import thư viện retrofit

```
implementation ("com.squareup.retrofit2:retrofit:2.9.0")
implementation ("com.squareup.retrofit2:converter-gson:2.9.0")
```

Bước 2: Tạo **MovieResponse** khai báo các giá trị trả về khi gọi API

```
data class MovieResponse(
    @SerializedName("filmId") val filmId: String,
    @SerializedName("filmName") val filmName: String,
    @SerializedName("duration") val duration: String,
    @SerializedName("releaseDate") val releaseDate: String,
    @SerializedName("genre") val genre: String,
    @SerializedName("national") val national: String,
    @SerializedName("description") val description: String,
    @SerializedName("image") val image: String,
)
```

Bước 3: Tạo **MovieService** quản lý các API gọi trong ứng dụng

```
interface MovieService {
    @GET("list-film.php")
    suspend fun getListFilms(): Response<List<MovieResponse>>
}
```

Bước 4: Tạo **RetrofitService** để cấu hình Retrofit

```
open class RetrofitService() {

    private val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl("<BASE_URL>")
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val movieService: MovieService by lazy {
        retrofit.create(MovieService::class.java)
    }
}
```

Bước 5: Trong class Movie, thêm id vào class Movie

```
data class Movie(
    val id: String,
    val title: String,
    val releaseDate: String,
    val posterUrl: String,
    val shotDescription: String,
    val genre: String,
    val duration: String,
)
```

Bước 6: Tạo class **Transfrom** để convert dữ liệu lấy từ API thành model

```
fun MovieResponse.toMovie(): Movie {
    return Movie(
        id = this.filmId,
        title = this.filmName,
        duration = this.duration,
        releaseDate = this.releaseDate,
        genre = this.genre,
        shotDescription = this.description,
        posterUrl = this.image
    )
}
```

Bước 7: Đổi tên **MainViewModel** thành **MovieViewModel**

Cập nhật lại **MovieViewModel**

```
class MovieViewModel : ViewModel() {
    private val _movies = MutableLiveData<List<Movie>>()
    val movies: LiveData<List<Movie>> = _movies

    init {
        getMovies()
    }

    fun getMovies() {
        viewModelScope.launch {
            try {
                val response = RetrofitService().movieService.getListFilms()
                if (response.isSuccessful) {
                    _movies.postValue(response.body()?.map { it.toMovie() })
                } else {
```

```
                _movies.postValue(emptyList())
            }
        } catch (e: Exception) {
            Log.e("TAG", "getMovies: " + e.message)
            _movies.postValue(emptyList())
        }
    }
  }
}
```

Bước 8: Thay đổi trong class **ScreenNavigation**

```
@Composable
fun ScreenNavigation() {
  val navController = rememberNavController()

  NavHost(
    navController = navController,
    startDestination = Screen.LOGIN.route,
  ) {
    composable(Screen.LOGIN.route) { LoginScreen(navController) }
    composable(Screen.MOVIE_SCREEN.route) { MovieScreen() }
    composable(Screen.SCREEN1.route) { Screen1(navController) }
    composable(Screen.SCREEN2.route) { Screen2(navController) }
    composable(Screen.SCREEN3.route) { Screen3(navController) }

  }
}
```

Bước 9: Trong MainActivity, trong hàm **MovieScreen()** khai báo MovieViewModel và movieState

```
val movieViewModel: MovieViewModel = viewModel()
val moviesState = movieViewModel.movies.observeAsState(initial = emptyList())
val movies = moviesState.value
```

Chạy ứng dụng và xem kết quả.

BÀI 2: THỰC HIỆN CHỨC NĂNG THÊM VÀ CHỈNH SỬA PHIM

Vì 2 chức năng thêm phim mới và chỉnh sửa thông tin phim ta xử lý trên cùng một giao diện nên ta sẽ xử lý 2 chức năng này song song với nhau

Bước 1: Thay đổi xử lý trong **MainActivity**

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            //thêm vào
            ScreenNavigation()
        }
    }
}

@Composable
fun MovieScreen() {

    val movieViewModel: MovieViewModel = viewModel()
    val moviesState = movieViewModel.movies.observeAsState(initial = emptyList())
    val movies = moviesState.value

    Column {
        Button(onClick = { }) {
            Text("Thêm")
        }

        MovieColumn(movies)
    }
}

@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],
                onEditClick = {},
                onDeleteClick = {})
        }
    }
}

@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
fun MovieColumnItem(
    movie: Movie,
    onEditClick: (id: String) -> Unit,
    onDeleteClick: (id: String) -> Unit
) {
    Card(
        colors = CardDefaults.cardColors(containerColor = Color.White),
        elevation = CardDefaults.cardElevation(defaultElevation = 6.dp),
    ) {
        Row(
            modifier = Modifier.fillMaxWidth()
        ) {
            AsyncImage(
                model = movie.image,
                contentDescription = null,
                contentScale = ContentScale.FillWidth,
                modifier = Modifier
                    .width(130.dp)
                    .wrapContentHeight()
            )

            Column(
                modifier = Modifier.padding(8.dp)
            ) {
                Text(
                    text = movie.filmName,
                    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.description,
                style = MaterialTheme.typography.bodySmall,
                maxLines = 5,
                overflow = TextOverflow.Ellipsis,
                modifier = Modifier.padding(end = 2.dp)
            )

            Row(
                modifier = Modifier
                    .padding(end = 4.dp)
            ) {
                IconButton(
                    onClick = { onEditClick(movie.id) },
                    modifier = Modifier.size(32.dp)
                ) {
                    Icon(
                        Icons.Filled.Edit,
                        contentDescription = "Edit",
                        tint = MaterialTheme.colorScheme.primary
                    )
                }
                Spacer(Modifier.height(8.dp))
                IconButton(
                    onClick = { onDeleteClick(movie.id) },
                    modifier = Modifier.size(32.dp)
                ) {
                    Icon(
                        Icons.Filled.Delete,
                        contentDescription = "Delete",
                        tint = MaterialTheme.colorScheme.error
                    )
                }
            }
        }
    }
}
```

Bước 2: Thay đổi class **Movie**

```
data class Movie(
    @SerializedName("filmId") val id: String,
    @SerializedName("filmName") val filmName: String,
    @SerializedName("duration") val duration: String,
    @SerializedName("releaseDate") val releaseDate: String,
    @SerializedName("genre") val genre: String,
    @SerializedName("national") val national: String,
    @SerializedName("description") val description: String,
```

```
    @SerializedName("image") val image: String,
)
```

Bước 3: Tạo class **MovieFormScreen** để xây dựng giao diện thêm/chỉnh sửa phim

```
@Composable
fun MovieFormScreen(
    navController: NavController,
    movieViewModel: MovieViewModel,
    filmId: String?,
) {
    val movie = movieViewModel.getMovieById(filmId).observeAsState(initial = null).value
    val isEditing = filmId != null

    var formData by remember(movie) {
        mutableStateOf(movie?.toMovieFormData() ?: MovieFormData())
    }

    MovieForm(formData = formData) { updatedFormData ->
        formData = updatedFormData
        val isValid =
            if (isEditing) movie?.let { validateMovieDataAndEnsureCompletion(updatedFormData, it) }
                ?: false
            else isAllFieldsEntered(updatedFormData)

    }
}

@Composable
fun MovieForm(
    formData: MovieFormData,
    onUpdateFormData: (MovieFormData) -> Unit,
) {
    Column(
        modifier = Modifier
            .padding(horizontal = 16.dp)
            .verticalScroll(rememberScrollState())
            .imePadding()
    ) {

        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier.fillMaxWidth(),
            value = formData.filmName,
            onValueChange = { onUpdateFormData(formData.copy(filmName = it)) },
            label = { Text("Tên Phim *") },
            maxLines = 3,
        )
```

```
Spacer(modifier = Modifier.height(12.dp))
Row(modifier = Modifier.fillMaxWidth()) {
    OutlinedTextField(
        value = formData.duration,
        onValueChange = { onUpdateFormData(formData.copy(duration = it)) },
        label = { Text("Thời Lượng *") },
        keyboardOptions = KeyboardOptions.Default.copy(keyboardType = KeyboardType.Number),
        singleLine = true,
        modifier = Modifier.weight(2f)
    )

    Spacer(modifier = Modifier.width(8.dp))
    DatePickerField(
        label = "Ngày Chiếu *",
        selectedDate = formData.releaseDate,
        onDateSelected = { newDate ->
            onUpdateFormData(formData.copy(releaseDate = newDate))
        },
        modifier = Modifier.weight(3f)
    )
}
Spacer(modifier = Modifier.height(12.dp))
OutlinedTextField(
    modifier = Modifier.fillMaxWidth(),
    value = formData.genre,
    onValueChange = { onUpdateFormData(formData.copy(genre = it)) },
    label = { Text("Thể Loại *") },
    singleLine = true
)
Spacer(modifier = Modifier.height(12.dp))
OutlinedTextField(
    modifier = Modifier.fillMaxWidth(),
    value = formData.national,
    onValueChange = { onUpdateFormData(formData.copy(national = it)) },
    label = { Text("Quốc Gia *") },
    singleLine = true
)
Spacer(modifier = Modifier.height(12.dp))
OutlinedTextField(
    modifier = Modifier.fillMaxWidth(),
    value = formData.imageUrl,
    onValueChange = { onUpdateFormData(formData.copy(imageUrl = it)) },
    label = { Text("Liên kết ảnh minh hoạ *") },
    singleLine = true
)
Spacer(modifier = Modifier.height(12.dp))
OutlinedTextField(
    modifier = Modifier
        .fillMaxWidth()
        .defaultMinSize(minHeight = 200.dp),
```

```
            value = formData.description,
            onValueChange = { onUpdateFormData(formData.copy(description = it)) },
            label = { Text("Mô Tả *") }
        )
        Spacer(modifier = Modifier.height(12.dp))
    }
}

@Composable
fun DatePickerField(
    label: String,
    selectedDate: String,
    onDateSelected: (String) -> Unit,
    modifier: Modifier
) {
    val context = LocalContext.current
    var showDialog by remember { mutableStateOf(false) }

    val dateFormat = SimpleDateFormat("dd/MM/yyyy", Locale.getDefault())

    if (showDialog) {
        val calendar = Calendar.getInstance()
        try {
            calendar.time = dateFormat.parse(selectedDate) ?: Date()
        } catch (_: Exception) {
        }

        val year = calendar.get(Calendar.YEAR)
        val month = calendar.get(Calendar.MONTH)
        val day = calendar.get(Calendar.DAY_OF_MONTH)

        DatePickerDialog(
            context, { _, selectedYear, selectedMonth, dayOfMonth ->
                val newCalendar = Calendar.getInstance().apply {
                    set(selectedYear, selectedMonth, dayOfMonth)
                }
                onDateSelected(dateFormat.format(newCalendar.time))
            }, year, month, day
        ).apply {
            show()
        }

        LaunchedEffect(Unit) {
            showDialog = false
        }
    }

    OutlinedTextField(
        modifier = modifier,
        value = selectedDate,
```

```
        onValueChange = { },
        readOnly = true,
        label = { Text(label) },
        trailingIcon = {
            Icon(
                Icons.Default.DateRange,
                contentDescription = "Chọn ngày",
                modifier = Modifier.clickable { showDialog = true },
            )
        }
    )
}

fun isAllFieldsEntered(formData: MovieFormData): Boolean {
    return with(formData) {
        filmName.isNotEmpty() && duration.isNotEmpty() && releaseDate.isNotEmpty() &&
                genre.isNotEmpty() && national.isNotEmpty() && description.isNotEmpty() && imageUrl.isNotEmpty()
    }
}

fun validateMovieDataAndEnsureCompletion(formData: MovieFormData, movie: Movie): Boolean {
    if (!isAllFieldsEntered(formData)) return false
    if (formData.filmName != movie.filmName) return true
    if (formData.duration != movie.duration) return true
    if (formData.releaseDate != movie.releaseDate) return true
    if (formData.genre != movie.genre) return true
    if (formData.national != movie.national) return true
    if (formData.description != movie.description) return true
    if (formData.imageUrl != movie.image) return true

    return false
}


data class MovieFormData(
    var id: String? = "",
    var filmName: String = "",
    var duration: String = "",
    var releaseDate: String = "",
    var genre: String = "",
    var national: String = "",
    var description: String = "",
    var imageUrl: String = ""
)
```

Bước 4: Trong class Transform, ta thay đổi như sau:

```
fun MovieResponse.toMovie(): Movie {
    return Movie(
        id = this.filmId,
        filmName = this.filmName,
        duration = this.duration,
        releaseDate = this.releaseDate,
        genre = this.genre,
        national = this.national,
        description = this.description,
        image = this.image
    )
}

fun MovieFormData.toMovieRequest(): MovieRequest {
    val filmIdInt = try {
        this.id?.toIntOrNull()
    } catch (e: NumberFormatException) {
        null
    }

    val durationInt = try {
        this.duration.toInt()
    } catch (e: NumberFormatException) {
        0
    }

    return MovieRequest(
        filmId = filmIdInt,
        filmName = this.filmName,
        duration = durationInt,
        releaseDate = this.releaseDate,
        genre = this.genre,
        national = this.national,
        description = this.description,
        image = this.imageUrl
    )
}

fun Movie?.toMovieFormData() = this?.let {
    MovieFormData(
        this.id,
        this.filmName,
        this.duration,
        this.releaseDate,
        this.genre,
        this.national,
        this.description,
```

```
        this.image
    )
}
```

Bước 5: Tạo class **MovieRequest** để quản lý các thông tin gửi lên server khi thực hiện gọi API thêm/chỉnh sửa

```
data class MovieRequest(
    val filmId: Int? = null,
    val filmName: String,
    val duration: Int,
    val releaseDate: String,
    val genre: String,
    val national: String,
    val description: String,
    val image: String
)

data class StatusResponse(
    val status: Int,
    val message: String
)
```

Bước 6: Trong **MovieService** khai báo thêm các API cần xử lý

```
@GET("film-detail.php")
suspend fun getFilmDetail(@Query("id") id: String): Response<MovieResponse>


@POST("add-film.php")
suspend fun addFilm(@Body filmRequest: MovieRequest): Response<StatusResponse>

@POST("update-film.php")
suspend fun updateFilm(@Body filmRequest: MovieRequest): Response<StatusResponse>
```

Bước 7: Trong **MovieViewModel** tạo thêm hàm **getMovieById** để lấy thông tin chi tiết phim

```
fun getMovieById(filmId: String?): LiveData<Movie?> {
    val liveData = MutableLiveData<Movie?>()
    filmId?.let {
        viewModelScope.launch {
            try {
                val response = RetrofitService().movieService.getFilmDetail(filmId)
                if (response.isSuccessful) {
                    liveData.postValue(response.body()?.toMovie())
                } else {
                    liveData.postValue(null)
                }
            } catch (e: Exception) {
                liveData.postValue(null)
            }
        }
    }
    return liveData
}
```

Bước 8: Trong class Screen khai báo thêm 2 route thêm và chỉnh sửa

```
enum class Screen(val route: String) {
    LOGIN("Login"),
    ADD("Add"),
    EDIT("Edit"),
    MOVIE_SCREEN("MovieScreen"),
    SCREEN1("Screen1"),
    SCREEN2("Screen2"),
    SCREEN3("Screen3"),
}
```

Bước 9: Trong class ScreenNavigation xử lý chức năng khi người dùng nhấn thêm/chỉnh sửa phim

```
@Composable
fun ScreenNavigation() {
    val navController = rememberNavController()
    val movieViewModel: MovieViewModel = viewModel()

    NavHost(
        navController = navController,
        startDestination = Screen.LOGIN.route,
    ) {
        composable(Screen.LOGIN.route) { LoginScreen(navController) }
        composable(Screen.ADD.route) { MovieFormScreen(navController, movieViewModel, null) }
        composable(
            "${Screen.EDIT.route}/{filmId}",
            arguments = listOf(navArgument("filmId") { type = NavType.StringType }),
        ) { backStackEntry ->
            backStackEntry.arguments?.getString("filmId")?.let { filmId ->
                MovieFormScreen(navController, movieViewModel, filmId)
            }
        }
        composable(Screen.MOVIE_SCREEN.route) { MovieScreen(navController, movieViewModel) }
        composable(Screen.SCREEN1.route) { Screen1(navController) }
        composable(Screen.SCREEN2.route) { Screen2(navController) }
        composable(Screen.SCREEN3.route) { Screen3(navController) }

    }
}
```

Bước 10: Trong MainActivity, thay đổi MovieColumn, thêm sự kiện click

```
@Composable
fun MovieColumn(movies: List<Movie>,
            onEditClick: (id: String) -> Unit,
            onDeleteClick: (id: String) -> Unit) {

   LazyColumn(
      state = rememberLazyListState(),
      contentPadding = PaddingValues(horizontal = 8.dp, vertical = 16.dp),
      verticalArrangement = Arrangement.spacedBy(8.dp)
   ) {
      items(movies.size) { index ->
         MovieColumnItem(
            movie = movies[index],
            onEditClick = onEditClick,
            onDeleteClick = onDeleteClick)
      }
   }
}
```

Bước 11: Thay đổi trong class MovieScreen, thêm nút Thêm phim mới và sự kiện nhấn nút chỉnh sửa phim

```
@Composable
fun MovieScreen(navigationController: NavController, movieViewModel: MovieViewModel) {
   val moviesState = movieViewModel.movies.observeAsState(initial = emptyList())
   val movies = moviesState.value

   Column {
      Button(onClick = {
         navigationController.navigate(Screen.ADD.route)
      }) {
         Text("Thêm")
      }

      MovieColumn(movies, onEditClick = {
         navigationController.navigate("${Screen.EDIT.route}/${it}")
      }, onDeleteClick = {})
   }
}
```

Bước 12: Trong MovieViewModel thêm 2 hàm xử lý chức năng thêm/chỉnh sửa phim

```
private val _isSuccess = MutableLiveData<Boolean>()
val isSuccess: LiveData<Boolean> = _isSuccess
fun addFilm(movieRequest: MovieRequest) {
    viewModelScope.launch {
        _isSuccess.value = try {
            val response = RetrofitService().movieService.addFilm(movieRequest)
            if (response.isSuccessful) {
                response.body()?.let {
                    if (it.status == 1) {
                        getMovies()
                        true
                    } else {
                        false
                    }
                } ?: false
            } else {
                false
            }
        } catch (e: Exception) {
            false
        }
    }
}

fun updateMovie(movieRequest: MovieRequest) {
    viewModelScope.launch {
        _isSuccess.value = try {
            val response = RetrofitService().movieService.updateFilm(movieRequest)
            if (response.isSuccessful) {
                response.body()?.let {
                    if (it.status == 1) {
                        getMovies()
                        true
                    } else {
                        false
                    }
                } ?: false
            } else {
                false
            }
        } catch (e: Exception) {
            false
        }
    }
}
```

Bước 13: Trong class **MovieFormScreen** tạo sự kiện khi người dùng nhấn nút Save

```
@Composable
fun MovieForm(
    formData: MovieFormData,
    onSave: () -> Unit,
    onUpdateFormData: (MovieFormData) -> Unit,
) {
    Column(
        modifier = Modifier
            .padding(horizontal = 16.dp)
            .verticalScroll(rememberScrollState())
            .imePadding()
    ) {

        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier.fillMaxWidth(),
            value = formData.filmName,
            onValueChange = { onUpdateFormData(formData.copy(filmName = it)) },
            label = { Text("Tên Phim *") },
            maxLines = 3,
        )
        Spacer(modifier = Modifier.height(12.dp))
        Row(modifier = Modifier.fillMaxWidth()) {
            OutlinedTextField(
                value = formData.duration,
                onValueChange = { onUpdateFormData(formData.copy(duration = it)) },
                label = { Text("Thời Lượng *") },
                keyboardOptions = KeyboardOptions.Default.copy(keyboardType = KeyboardType.Number),
                singleLine = true,
                modifier = Modifier.weight(2f)
            )

            Spacer(modifier = Modifier.width(8.dp))
            DatePickerField(
                label = "Ngày Chiếu *",
                selectedDate = formData.releaseDate,
                onDateSelected = { newDate ->
                    onUpdateFormData(formData.copy(releaseDate = newDate))
                },
                modifier = Modifier.weight(3f)
            )
        }
        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier.fillMaxWidth(),
            value = formData.genre,
            onValueChange = { onUpdateFormData(formData.copy(genre = it)) },
            label = { Text("Thể Loại *") },
            singleLine = true
        )
```

```
        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier.fillMaxWidth(),
            value = formData.national,
            onValueChange = { onUpdateFormData(formData.copy(national = it)) },
            label = { Text("Quốc Gia *") },
            singleLine = true
        )
        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier.fillMaxWidth(),
            value = formData.imageUrl,
            onValueChange = { onUpdateFormData(formData.copy(imageUrl = it)) },
            label = { Text("Liên kết ảnh minh hoạ *") },
            singleLine = true
        )
        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            modifier = Modifier
                .fillMaxWidth()
                .defaultMinSize(minHeight = 200.dp),
            value = formData.description,
            onValueChange = { onUpdateFormData(formData.copy(description = it)) },
            label = { Text("Mô Tả *") }
        )
        Spacer(modifier = Modifier.height(12.dp))
        Button(onClick = {
            onSave()
        }) {
            Text(text = "Save")
        }
    }
}
```

Chạy ứng dụng và kiểm tra kết quả

BÀI 3: XÂY DỰNG CHỨC NĂNG XÓA PHIM

Bước 1: Trong **MovieViewModel** thêm hàm xử lý chức năng xóa

```
fun deleteMovieById(id: String) {
    viewModelScope.launch {
        _isSuccess.value = try {
            val response = RetrofitService().movieService.deleteFilm(id)
            if (response.isSuccessful) {
                response.body()?.let {
```

```
                if (it.status == 1) {
                    getMovies()
                    true
                } else {
                    false
                }
            } ?: false
        } else {
            false
        }
    } catch (e: Exception) {
        false
    }
  }
}
```

Bước 2: Trong MainActivity gọi hàm xóa khi người dùng nhấn nút xóa

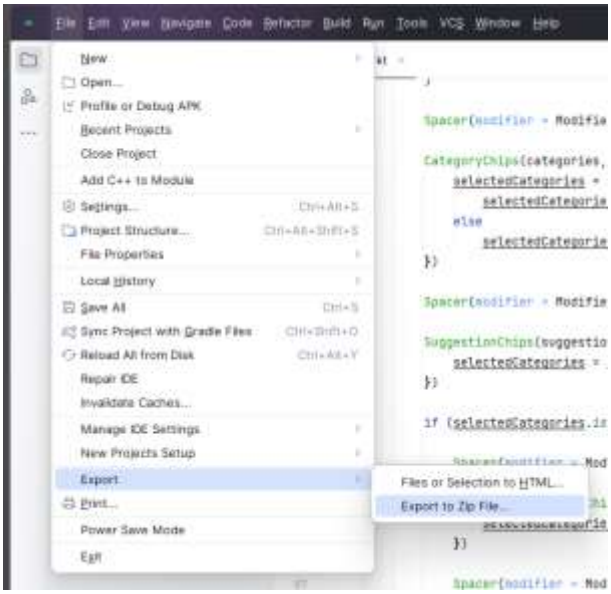
```
@Composable
fun MovieScreen(navigationController: NavController, movieViewModel: MovieViewModel) {
    val moviesState = movieViewModel.movies.observeAsState(initial = emptyList())
    val movies = moviesState.value

    Column {
        Button(onClick = {
            navigationController.navigate(Screen.ADD.route)
        }) {
            Text("Thêm")
        }

        MovieColumn(movies, onEditClick = {
            navigationController.navigate("${Screen.EDIT.route}/${it}")
        }, onDeleteClick = {
            movieViewModel.deleteMovieById(it);
        })
    }
}
```

Chạy ứng dụng và kiểm tra kết 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**.


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
New
Open...
Profile or Debug APK
Recent Projects
Close Project
Add C++ to Module
Settings...
Project Structure...
File Properties
Local History
Save All
Sync Project with Gradle Files
Reload All from Disk
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Export
Print...
Power Save Mode
Exit
Files or Selection to HTML...
Export to Zip File...

--- Hết ---