

Министерство образования и молодежной политики Свердловской области

ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Отчёт по учебной практике

УП 02.01

Выполнила: Корпушенко Карина Ришатовна

Группа: ПР-31

Преподаватель: Мирошниченко Г.В.

2024

Содержание

[1. **Задание №5 Мобильное приложение «Онлайн-кинотеатр»** 3](#_gjdgxs)

[1.1 Описание задачи 3](#_30j0zll)

[1.2 Структура проекта 5](#_1fob9te)

[1.3 Описание разработанных функций 6](#_3znysh7)

[1.4 Используемые библиотеки 34](#_tyjcwt)

[1.5 Используемые инструменты 3](#_3dy6vkm)4

[1.6 Описание пользовательского интерфейса 3](#_1t3h5sf)4

[1.7 Приложение (pr screen экранов) 36](#_4d34og8)

# 1. **Задание №5 Мобильное приложение «Онлайн-кинотеатр»**

## Описание задачи

1. Cоздать приложение для онлайн-кинотеатра (используя

примеры сайтов в интернете)

На главной странице должна быть реализована авторизация и выбор режима

работы (для работника кинотеатра и для клиента), с проверкой на

правильность введенных данных (проверка на правильность ввода почты,

логина и пароля) и сохранением через Shared Preference в базе данных.

В этой базе заказчик хотел бы хранить информацию:

 О номере билета (является первичным ключом для базы данных), с

автоматической нумерацией на 1.

 о фильмах в прокате (с фото) – информацию запрашивать через API

 о том, к какому жанру относится фильм и возможностью группировки

по жанру

 о наличии конкретных билетов в данный момент

 о клиентах, обслуживаемых фирмой, с возможностью группировки по

клиенту.

Некоторые условия работы онлайн-кинотеатра, существенные для

проектирования базы данных: после выбора фильма, осуществляется

переход на другую страницу

 в онлайн-кинотеатре может быть несколько различных фильмов одного

жанра

 можно составить свой диск-сборник из нескольких фильмов – купить

права на просмотр (с учетом скидки).

 каждый клиент может купить билет на несколько фильмов (с учетом

возрастных ограничений и фильмы по времени не должны пересекаться –

если клиент идет в кинотеатр) – для клиентов должна быть реализована

система скидок с учетом возраста.

В приложении должно быть организовано не менее 4-х – 5 окон

Навигацию осуществить наиболее оптимальным образом для приложения (с

помощью Carousel with MotionLayout, Navigation) – должны быть

использованы не менее 2-х видов навигации.

В приложении должен быть реализован следующий функционал:

Редактирование, удаление, добавление фильмов и клиентов.

На главной странице поиск по наименованию фильма.

2. Создайте и заполните готовую базу и подключите его в проекте через

ROOM.

## Структура проекта

## 

## 1.3 Описание разработанных функций

## Класс AdditionMovieFragment

package com.example.megogo  
  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import android.widget.EditText  
import android.widget.ImageView  
import android.widget.TextView  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import androidx.lifecycle.Observer  
import androidx.lifecycle.ViewModelProvider  
import androidx.lifecycle.*lifecycleScope*import com.android.volley.Request  
import com.android.volley.toolbox.StringRequest  
import com.android.volley.toolbox.Volley  
import com.squareup.picasso.Picasso  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
import org.json.JSONObject  
  
class AdditionMovieFragment : Fragment() {  
  
 private lateinit var searchTitle: EditText  
 private lateinit var searchYear: EditText  
 private lateinit var searchButton: Button  
 private lateinit var addButton: Button  
 private lateinit var posterIm: ImageView  
 private lateinit var titleText: TextView  
 private lateinit var yearText: TextView  
 private lateinit var genreText: TextView  
 private lateinit var plotText: TextView  
 private lateinit var db: CinemaDatabase  
 private var movie: Movie? = null  
 private lateinit var movieViewModel: MovieViewModel  
  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 val view = inflater.inflate(R.layout.*fragment\_addition\_movie*, container, false)  
  
 db = DatabaseClient.getInstance(requireContext())  
 searchTitle = view.findViewById(R.id.*etSearchTitle*)  
 searchYear = view.findViewById(R.id.*etSearchYear*)  
 searchButton = view.findViewById(R.id.*btnSearch*)  
 addButton = view.findViewById(R.id.*btnAdd*)  
 posterIm = view.findViewById(R.id.*tvPoster*)  
 titleText = view.findViewById(R.id.*tvTitle*)  
 yearText = view.findViewById(R.id.*tvYear*)  
 genreText = view.findViewById(R.id.*tvGenre*)  
 plotText = view.findViewById(R.id.*tvPlot*)  
  
 // Инициализация ViewModel  
 movieViewModel = ViewModelProvider(this).get(MovieViewModel::class.*java*)  
  
 // Наблюдаем за результатом добавления фильма  
 movieViewModel.movieAdded.observe(*viewLifecycleOwner*, *Observer* **{** success **->** if (success) {  
 Toast.makeText(requireContext(), "Movie added successfully!", Toast.*LENGTH\_SHORT*).show()  
 } else {  
 Toast.makeText(requireContext(), "There's a error during addition of the movie", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}**)  
  
 searchButton.setOnClickListener **{** if (searchTitle.*text*.*isNotEmpty*()) {  
 val title = searchTitle.*text*.toString()  
 val year = searchYear.*text*.toString()  
 searchMovies(title, year)  
 } else {  
 Toast.makeText(  
 requireContext(),  
 "Fill at least a name of a movie",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** addButton.setOnClickListener **{** addMovieToDatabase(movie)  
 **}** return view  
 }  
  
 private fun searchMovies(title: String, year: String) {  
 val url = "https://www.omdbapi.com/?apikey=8424b5c9&t=$title&y=$year"  
  
 val queue = Volley.newRequestQueue(requireContext())  
 val stringRequest = StringRequest(  
 Request.Method.*GET*,  
 url,  
 **{** response **->** val obj = JSONObject(response)  
 try {  
 movie = Movie(  
 id = 0,  
 title = obj.getString("Title"),  
 year = obj.getString("Year"),  
 poster = obj.getString("Poster"),  
 description = obj.getString("Plot"),  
 genre = obj.getString("Genre")  
 )  
 movie?.*let* **{** loadMovie(**it**) **}** } catch (e: Exception) {  
 Toast.makeText(  
 requireContext(),  
 "The movie wasn't found",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}**,  
 **{** Toast.makeText(requireContext(), "There's some error, we're soon gonna fix it", Toast.*LENGTH\_SHORT*).show()  
 **}** )  
 queue.add(stringRequest)  
 }  
  
 private fun loadMovie(movie: Movie) {  
 titleText.*text* = movie.title  
 yearText.*text* = movie.year  
 plotText.*text* = movie.description  
 genreText.*text* = movie.genre  
 Picasso.get()  
 .load(movie.poster)  
 .placeholder(R.drawable.*holder*)  
 .into(posterIm)  
 }  
  
 private fun addMovieToDatabase(movie: Movie?) {  
 if (movie == null) {  
 Toast.makeText(requireContext(), "Please search and select a movie first", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
  
 *lifecycleScope*.*launch* **{** try {  
 val existingMovie = withContext(Dispatchers.IO) **{** db.movieDao().getMovieByTitle(movie.title)  
 **}** if (existingMovie == null) {  
 withContext(Dispatchers.IO) **{** movieViewModel.addMovieWithTickets(movie)  
 **}** } else {  
 withContext(Dispatchers.Main) **{** Toast.makeText(requireContext(), "The movie is already in the rental", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
 } catch (e: Exception) {  
 withContext(Dispatchers.Main) **{** Toast.makeText(requireContext(), "Error adding movie", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
 **}** }  
}

Данный класс отвечает за реализацию поиска фильма через API-запрос, фильтрацию фильмов по жанру, отображение информации о фильме на экране и добавления его в прокат

Класс Authorisation

package com.example.megogo  
  
import android.content.Intent  
import android.content.SharedPreferences  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.view.View  
import android.widget.EditText  
import android.widget.Toast  
import androidx.room.Room  
import kotlinx.coroutines.GlobalScope  
import kotlinx.coroutines.launch  
  
class Authorisation : AppCompatActivity() {  
 private lateinit var loginOrEmail : EditText  
 private lateinit var pass : EditText  
 private lateinit var db: CinemaDatabase  
 private lateinit var pref : SharedPreferences  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_authorisation*)  
  
 loginOrEmail = findViewById(R.id.*emailOrLogin*)  
 pass = findViewById(R.id.*password*)  
 db = DatabaseClient.getInstance(*applicationContext*)  
 pref = getSharedPreferences("PREF", *MODE\_PRIVATE*)  
  
 if (pref.getBoolean("IsAdminLogged", false))  
 {  
 startActivity(Intent(this@Authorisation, MainActivityForAdm::class.*java*))  
 }  
 else if (pref.getBoolean("IsUserLogged", false))  
 {  
 startActivity(Intent(this@Authorisation, MainActivityForUser::class.*java*))  
 }  
 }  
  
 fun goToRegister(view: View) {  
 val intent = Intent(this, Registration::class.*java*)  
 startActivity(intent)  
 }  
  
 fun Login(view: View) {  
 if (loginOrEmail.*text*.*isNotEmpty*() && pass.*text*.*isNotEmpty*())  
 {  
 loginUser(loginOrEmail.*text*.toString(), pass.*text*.toString())  
 }  
 else  
 Toast.makeText(this@Authorisation, "Fill in the blanks", Toast.*LENGTH\_SHORT*).show()  
 }  
  
 private fun loginUser(name: String, password: String): Boolean {  
 var isAuthenticated = false  
 GlobalScope.*launch* **{** val user = db.userDao().getUserByUsernameOrEmailAndPassword(name, password)  
 isAuthenticated = user != null  
 runOnUiThread **{** if (isAuthenticated) {  
 if (user!!.role == "User") {  
 pref.edit().putBoolean("IsAdminLogged", false).apply()  
 pref.edit().putBoolean("IsUserLogged", true).apply()  
 pref.edit().putString("Online", user.username).apply()  
 startActivity(Intent(this@Authorisation, MainActivityForUser::class.*java*))  
 }  
 else if (user.role == "Admin") {  
 pref.edit().putBoolean("IsUserLogged", false).apply()  
 pref.edit().putBoolean("IsAdminLogged", true).apply()  
 startActivity(Intent(this@Authorisation, MainActivityForAdm::class.*java*))  
 }  
 } else {  
 Toast.makeText(this@Authorisation, "Invalid username or password", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}  
 }** return isAuthenticated  
 }  
}

Отвечает за логику авторизации пользователя в системе

Классы для БД

package com.example.megogo  
  
import android.app.Application  
import android.content.Context  
import androidx.lifecycle.AndroidViewModel  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.*viewModelScope*import androidx.room.Dao  
import androidx.room.Database  
import androidx.room.Delete  
import androidx.room.Entity  
import androidx.room.ForeignKey  
import androidx.room.Index  
import androidx.room.Insert  
import androidx.room.OnConflictStrategy  
import androidx.room.PrimaryKey  
import androidx.room.Query  
import androidx.room.Room  
import androidx.room.RoomDatabase  
import androidx.room.Update  
import androidx.room.migration.Migration  
import androidx.sqlite.db.SupportSQLiteDatabase  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
object DatabaseClient {  
 private var instance: CinemaDatabase? = null  
  
 fun getInstance(context: Context): CinemaDatabase {  
 if (instance == null) {  
 *synchronized*(CinemaDatabase::class.*java*) **{** if (instance == null) {  
 instance = Room.databaseBuilder(  
 context.*applicationContext*,  
 CinemaDatabase::class.*java*,  
 "cinema\_database"  
 )  
 .build()  
 }  
 **}** }  
 return instance!!  
 }  
}  
  
  
  
class MovieViewModel(application: Application) : AndroidViewModel(application) {  
  
 private val movieDao: MovieDao = DatabaseClient.getInstance(application).movieDao()  
 private val ticketDao: TicketDao = DatabaseClient.getInstance(application).ticketDao()  
 private val repository: CinemaRepository = CinemaRepository(movieDao, ticketDao)  
  
 private val \_movieAdded = MutableLiveData<Boolean>()  
 val movieAdded: LiveData<Boolean> get() = \_movieAdded  
  
 // Функция для добавления фильма с билетами  
 fun addMovieWithTickets(movie: Movie) {  
 *viewModelScope*.*launch* **{** try {  
 repository.addMovieWithTickets(movie) // Добавление фильма с билетами  
 \_movieAdded.postValue(true)  
 } catch (e: Exception) {  
 \_movieAdded.postValue(false)  
 }  
 **}** }  
}  
  
  
  
class CinemaRepository(  
 private val movieDao: MovieDao,  
 private val ticketDao: TicketDao  
) {  
 suspend fun addMovieWithTickets(movie: Movie) {  
 withContext(Dispatchers.IO) **{** val movieId = movieDao.insertMovie(movie)  
 val tickets = *mutableListOf*<Ticket>()  
 for (row in 1..5) { // 5 рядов  
 for (seat in 1..10) { // 10 мест в ряду  
 tickets.add(  
 Ticket(  
 row = row,  
 seat = seat,  
 movieId = movieId,  
 ownerUsername = null  
 )  
 )  
 }  
 }  
 ticketDao.insertTickets(tickets)  
 **}** }  
}  
  
  
  
@Database(entities = [Movie::class, User::class, Ticket::class], version = 2)  
abstract class CinemaDatabase : RoomDatabase() {  
 abstract fun movieDao(): MovieDao  
 abstract fun userDao(): UserDao  
 abstract fun ticketDao(): TicketDao  
}  
  
  
  
  
@Dao  
interface MovieDao {  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 fun insertMovie(movie: Movie): Long  
  
 @Delete  
 fun deleteMovie(movie: Movie)  
  
 @Query("SELECT \* FROM movies")  
 fun getAllMovies(): List<Movie>  
  
 @Query("SELECT \* FROM movies WHERE title LIKE :query")  
 fun searchMovies(query: String): List<Movie>  
  
 @Query("SELECT \* FROM movies WHERE title = :title LIMIT 1")  
 fun getMovieByTitle(title: String): Movie?  
  
 @Query("SELECT \* FROM movies WHERE id = :movieId LIMIT 1")  
 fun getMovieById(movieId: Long): Movie?  
}  
  
@Entity(tableName = "movies")  
data class Movie(  
 @PrimaryKey(autoGenerate = true)  
 val id: Long = 0,  
 val title: String,  
 val genre: String,  
 val description: String,  
 val poster: String,  
 val year: String  
)  
  
  
  
@Dao  
interface UserDao {  
  
 @Insert  
 fun insertUser(user: User)  
  
 @Update  
 fun updateUser(user: User)  
  
 @Delete  
 fun deleteUser(user: User)  
  
 @Query("SELECT \* FROM users WHERE (username = :name OR email = :name) AND password = :password LIMIT 1")  
 fun getUserByUsernameOrEmailAndPassword(name: String, password: String): User?  
  
 @Query("SELECT \* FROM users WHERE role = :role")  
 fun getUsersByRole(role: String): List<User?>  
  
 @Query("SELECT \* FROM users WHERE username = :username OR email = :email LIMIT 1")  
 fun getUserByUsernameOrEmail(username: String, email: String): User?  
}  
  
  
  
@Entity(tableName = "users")  
data class User(  
 @PrimaryKey(autoGenerate = true)  
 val id: Long = 0,  
 val username: String,  
 val email: String,  
 val password: String,  
 val role: String // Роль: "User" или "Admin"  
)  
  
  
  
@Entity(  
 tableName = "tickets",  
 foreignKeys = [  
 ForeignKey(  
 entity = Movie::class,  
 parentColumns = ["id"],  
 childColumns = ["movieId"],  
 onDelete = ForeignKey.CASCADE  
 )  
 ],  
 indices = [Index(value = ["movieId"])]  
)  
  
data class Ticket(  
 @PrimaryKey(autoGenerate = true) val id: Long = 0,  
 val row: Int,  
 val seat: Int,  
 val movieId: Long,  
 var ownerUsername: String? = null // Логин владельца или null, если билет не куплен  
)  
  
  
  
@Dao  
interface TicketDao {  
 @Insert  
 fun insertTickets(tickets: List<Ticket>)  
  
 @Delete  
 fun deleteTicket(ticket: Ticket)  
  
 @Query("UPDATE tickets SET ownerUsername = :username WHERE id = :ticketId")  
 fun updateTicket(ticketId: Long, username: String)  
  
 @Query("DELETE FROM tickets WHERE movieId = :movieId")  
 fun deleteTicketsByMovieId(movieId: Long)  
  
 @Query("SELECT \* FROM tickets WHERE movieId = :movieId")  
 fun getTicketsByMovieId(movieId: Long): List<Ticket>  
  
 @Query("UPDATE tickets SET ownerUsername = :username WHERE id = :ticketId")  
 fun updateTicketOwner(ticketId: Long, username: String)  
  
 @Query("SELECT \* FROM tickets WHERE ownerUsername = :username")  
 fun getTicketsByOwner(username: String): List<Ticket>  
  
 @Query("SELECT \* FROM tickets WHERE movieId = :movieId")  
 fun getTicketsByMovie(movieId: Long): List<Ticket>  
  
 @Query("SELECT \* FROM tickets WHERE ownerUsername = :username")  
 fun getBoughtTicketsByUser(username: String): List<Ticket>  
  
 @Query("UPDATE tickets SET ownerUsername = NULL WHERE id = :ticketId")  
 fun refundTicket(ticketId: Long)  
}

Данные классы формируют единую базу данных, состоящую из таблиц: пользователи, фильмы и билеты, а также содержат готовые запросы в БД, которыми можно воспользоваться в последующем коде через готовый интерфейс.

Класс MainActivityForAdm

package com.example.megogo  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import com.example.megogo.R  
import com.google.android.material.bottomnavigation.BottomNavigationView  
  
class MainActivityForAdm : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main\_for\_adm*)  
  
 // Получаем BottomNavigationView из разметки  
 val bottomNavigationView: BottomNavigationView = findViewById(R.id.*bottom\_nav*)  
  
 // Устанавливаем слушатель для выбора пунктов меню  
 bottomNavigationView.setOnNavigationItemSelectedListener **{** item **->** when (item.*itemId*) {  
 R.id.*nav\_addition* -> {  
 // Заменяем фрагмент для Home  
 replaceFragment(AdditionMovieFragment())  
 true  
 }  
 R.id.*nav\_removing* -> {  
 // Заменяем фрагмент для Search  
 replaceFragment(RemovingMovieFragment())  
 true  
 }  
 else -> false  
 }  
 **}** // Устанавливаем фрагмент по умолчанию (например, HomeFragment)  
 if (savedInstanceState == null) {  
 bottomNavigationView.*selectedItemId* = R.id.*nav\_addition* }  
 }  
  
 // Функция для замены фрагмента  
 private fun replaceFragment(fragment: Fragment) {  
 *supportFragmentManager*.beginTransaction()  
 .replace(R.id.*container*, fragment)  
 .commit()  
 }  
}

Данный класс предназначен для отображения фрагментов для администратора в контейнере, а так же меню для навигации между данными фрагментами.

Класс MainActivityForUser

package com.example.megogo  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import com.example.megogo.R  
import com.google.android.material.bottomnavigation.BottomNavigationView  
  
class MainActivityForUser : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main\_for\_user*)  
  
 // Получаем BottomNavigationView из разметки  
 val bottomNavigationView: BottomNavigationView = findViewById(R.id.*bottom\_nav\_user*)  
  
 // Устанавливаем слушатель для выбора пунктов меню  
 bottomNavigationView.setOnNavigationItemSelectedListener **{** item **->** when (item.*itemId*) {  
 R.id.*nav\_searching* -> {  
 replaceFragment(SearchingMovieFragment())  
 true  
 }  
 R.id.*nav\_myTickets* -> {  
 replaceFragment(MyTicketsFragment())  
 true  
 }  
 else -> false  
 }  
 **}** // Устанавливаем фрагмент по умолчанию  
 if (savedInstanceState == null) {  
 bottomNavigationView.*selectedItemId* = R.id.*nav\_searching* }  
 }  
  
 // Функция для замены фрагмента  
 private fun replaceFragment(fragment: Fragment) {  
 *supportFragmentManager*.beginTransaction()  
 .replace(R.id.*container\_user*, fragment)  
 .commit()  
 }  
}

Выполняет те же функции, что и MainActivityForAdm, только учитывая то, что для пользователя будет другое меню и другие фрагменты.

Классы кастомизированных адаптеров

package com.example.megogo  
  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import android.widget.ImageView  
import android.widget.TextView  
import androidx.recyclerview.widget.RecyclerView  
import com.example.megogo.R  
import com.example.megogo.CinemaDatabase  
import com.squareup.picasso.Picasso  
  
//MovieAdapterAdm для экрана удаления фильмов  
class MovieAdapterAdm(  
 private val movies: MutableList<Movie>,  
 private val onDeleteClick: (Movie) -> Unit  
) : RecyclerView.Adapter<MovieAdapterAdm.MovieViewHolder>() {  
  
 class MovieViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 val tvMovieTitle: TextView = view.findViewById(R.id.*tvMovieTitle*)  
 val tvMovieGenre: TextView = view.findViewById(R.id.*tvMovieGenre*)  
 val tvMovieYear: TextView = view.findViewById(R.id.*tvMovieYear*)  
 val ivPoster: ImageView = view.findViewById(R.id.*ivPoster*)  
 val btnDelete: Button = view.findViewById(R.id.*btnDelete*)  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder {  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_movie\_adm*, parent, false)  
 return MovieViewHolder(view)  
 }  
  
 override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {  
 val movie = movies[position]  
 holder.tvMovieTitle.*text* = movie.title  
 holder.tvMovieGenre.*text* = movie.genre  
 holder.tvMovieYear.*text* = movie.year  
  
 // Загружаем изображение постера с помощью Picasso  
 Picasso.get().load(movie.poster).placeholder(R.drawable.*holder*).into(holder.ivPoster)  
  
 holder.btnDelete.setOnClickListener **{** onDeleteClick(movie) **}** }  
  
 override fun getItemCount(): Int = movies.size  
}  
  
  
//MovieAdapter для экрана пользователя  
data class MovieItem(val id: Long, val title: String, val genre: String, val posterUrl: String, val year: String, val plot: String)  
  
class MovieAdapter(private var movieList: List<MovieItem>, private val onMovieClick: (MovieItem) -> Unit) :  
 RecyclerView.Adapter<MovieAdapter.MovieViewHolder>() {  
  
 inner class MovieViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 val title: TextView = view.findViewById(R.id.*movie\_title*)  
 val genre: TextView = view.findViewById(R.id.*movie\_genre*)  
 val poster: ImageView = view.findViewById(R.id.*movie\_poster*)  
 val yearText: TextView = view.findViewById(R.id.*movie\_year*)  
 val descText: TextView = view.findViewById(R.id.*movie\_desc*)  
  
 init {  
 // Обработка клика по элементу фильма  
 view.setOnClickListener **{** onMovieClick(movieList[*adapterPosition*])  
 **}** }  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder {  
 val view = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_movie*, parent, false)  
 return MovieViewHolder(view)  
 }  
  
 override fun onBindViewHolder(holder: MovieViewHolder, position: Int) {  
 val movie = movieList[position]  
 holder.title.*text* = movie.title  
 holder.genre.*text* = movie.genre  
 holder.yearText.*text* = movie.year  
 holder.descText.*text* = movie.plot  
 Picasso.get().load(movie.posterUrl).placeholder(R.drawable.*holder*).into(holder.poster)  
 }  
  
 override fun getItemCount(): Int = movieList.size  
  
 fun updateMovies(newMovies: List<MovieItem>) {  
 movieList = newMovies  
 notifyDataSetChanged()  
 }  
}  
  
  
  
class TicketAdapter(private var tickets: List<Ticket>, private val onTicketClick: (Ticket) -> Unit) :  
 RecyclerView.Adapter<TicketAdapter.TicketViewHolder>() {  
  
 inner class TicketViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 val row: TextView = view.findViewById(R.id.*ticketRow*)  
 val seat: TextView = view.findViewById(R.id.*ticketSeat*)  
 val button: Button = view.findViewById(R.id.*ticketButton*)  
  
 init {  
 button.setOnClickListener **{** onTicketClick(tickets[*adapterPosition*])  
 **}** }  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TicketViewHolder {  
 val view = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_ticket*, parent, false)  
 return TicketViewHolder(view)  
 }  
  
 override fun onBindViewHolder(holder: TicketViewHolder, position: Int) {  
 val ticket = tickets[position]  
 holder.row.*text* = "Row: ${ticket.row}"  
 holder.seat.*text* = "Seat: ${ticket.seat}"  
 holder.button.*text* = if (ticket.ownerUsername == null) "Buy" else "Taken"  
 holder.button.*isEnabled* = ticket.ownerUsername == null  
 }  
  
 override fun getItemCount(): Int = tickets.size  
  
 fun updateTickets(newTickets: List<Ticket>) {  
 tickets = newTickets  
 notifyDataSetChanged()  
 }  
}  
  
  
  
// Класс для объединения билета и фильма в один объект  
data class TicketWithMovie(  
 val ticket: Ticket, // Информация о билете  
 val movie: Movie? // Информация о фильме  
)  
  
class MyTicketAdapter(  
 private var ticketList: List<TicketWithMovie>,  
 private val onRefundTicketClicked: (TicketWithMovie) -> Unit  
) : RecyclerView.Adapter<MyTicketAdapter.TicketViewHolder>() {  
  
 inner class TicketViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 val ticketInfo: TextView = view.findViewById(R.id.*ticket\_info*)  
 val movieTitle: TextView = view.findViewById(R.id.*movie\_title*)  
 val movieYear: TextView = view.findViewById(R.id.*movie\_year*)  
 val refundButton: Button = view.findViewById(R.id.*refund\_button*)  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TicketViewHolder {  
 // Теперь используем my\_item\_ticket.xml  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*my\_item\_ticket*, parent, false)  
 return TicketViewHolder(view)  
 }  
  
 override fun onBindViewHolder(holder: TicketViewHolder, position: Int) {  
 val ticketWithMovie = ticketList[position]  
  
 // Отображаем информацию о билете  
 holder.ticketInfo.*text* = "Row: ${ticketWithMovie.ticket.row}, Seat: ${ticketWithMovie.ticket.seat}"  
 holder.movieTitle.*text* = ticketWithMovie.movie!!.title  
 holder.movieYear.*text* = ticketWithMovie.movie.year  
  
 // Обработчик нажатия на кнопку "Вернуть средства"  
 holder.refundButton.setOnClickListener **{** onRefundTicketClicked(ticketWithMovie)  
 **}** }  
  
 override fun getItemCount(): Int = ticketList.size  
  
 fun updateTickets(newTickets: List<TicketWithMovie>) {  
 ticketList = newTickets  
 notifyDataSetChanged()  
 }  
}

Выполняют функции отрисовки и логики отдельных элементов, отображаемых в recyclerView

Класс MovieDetailsActivity

package com.example.megogo  
  
import android.content.SharedPreferences  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.text.StaticLayout  
import android.view.View  
import android.widget.Button  
import android.widget.ImageView  
import android.widget.TextView  
import android.widget.Toast  
import androidx.lifecycle.*lifecycleScope*import androidx.recyclerview.widget.LinearLayoutManager  
import androidx.recyclerview.widget.RecyclerView  
import com.squareup.picasso.Picasso  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class MovieDetailsActivity : AppCompatActivity() {  
  
 private lateinit var pref : SharedPreferences  
 private lateinit var ticketAdapter: TicketAdapter  
 private lateinit var db: CinemaDatabase  
 private var movieId: Long = -1  
 private var selectedTicket: Ticket? = null  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_movie\_details*)  
  
 pref = getSharedPreferences("PREF", *MODE\_PRIVATE*)  
 db = DatabaseClient.getInstance(this)  
  
 // Получаем данные фильма  
 movieId = *intent*.getLongExtra("movieId", -1)  
 if (movieId == -1L) {  
 Toast.makeText(this, "Movie not found", Toast.*LENGTH\_SHORT*).show()  
 finish()  
 return  
 }  
  
 // Настройка RecyclerView для билетов  
 ticketAdapter = TicketAdapter(*emptyList*()) **{** ticket **->** selectedTicket = ticket  
 onBuyTicketClicked()  
 **}** findViewById<RecyclerView>(R.id.*ticketRecyclerView*).*apply* **{** *layoutManager* = LinearLayoutManager(this@MovieDetailsActivity)  
 *adapter* = ticketAdapter  
 **}** // Загружаем данные о фильме и билетах  
 loadMovieDetails()  
 loadTickets()  
 }  
  
 private fun loadMovieDetails() {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** val movie = db.movieDao().getMovieById(movieId)  
 withContext(Dispatchers.Main) **{** findViewById<TextView>(R.id.*movieTitle*).*text* = movie!!.title  
 findViewById<TextView>(R.id.*movieGenre*).*text* = movie.genre  
 findViewById<TextView>(R.id.*movieYear*).*text* = movie.year  
 findViewById<TextView>(R.id.*moviePlot*).*text* = movie.description  
 Picasso.get().load(movie.poster).into(findViewById<ImageView>(R.id.*moviePoster*))  
 **}  
 }** }  
  
 private fun loadTickets() {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** val tickets = db.ticketDao().getTicketsByMovieId(movieId).*filter* **{ it**.ownerUsername == null **}** withContext(Dispatchers.Main) **{** ticketAdapter.updateTickets(tickets)  
 **}  
 }** }  
  
 private fun onBuyTicketClicked() {  
 selectedTicket?.*let* **{** if (**it**.ownerUsername == null) {  
 // Купить билет  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** val username = pref.getString("Online", "User008")  
 db.ticketDao().updateTicket(**it**.id, username.*toString*())  
 withContext(Dispatchers.Main) **{** Toast.makeText(this@MovieDetailsActivity, "Ticket bought", Toast.*LENGTH\_SHORT*).show()  
 loadTickets() // Обновим список билетов  
 **}  
 }** } else {  
 Toast.makeText(this, "This ticket is already taken", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** ?: *run* **{** Toast.makeText(this, "Select a ticket first", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
}

Данный класс позволяет пользователю по выбранному фильму выбрать и купить билет на фильм, на этой активности присутствуют функции: отображения информации о фильме на экране, отображении информации о билетах, доступных в продаже, покупка билета, благодаря чему, билет будет более не числиться доступным к продаже

Класс MyTicketsFragment

package com.example.megogo  
  
import android.content.Context  
import android.content.SharedPreferences  
import android.os.Bundle  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import androidx.lifecycle.*lifecycleScope*import androidx.recyclerview.widget.LinearLayoutManager  
import com.example.megogo.databinding.FragmentMyTicketsBinding  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class MyTicketsFragment : Fragment() {  
  
 private var \_binding: FragmentMyTicketsBinding? = null  
 private val binding get() = \_binding!!  
  
 private lateinit var ticketAdapter: MyTicketAdapter // Используем MyTicketAdapter  
 private lateinit var db: CinemaDatabase  
 private lateinit var username: String  
 private lateinit var pref: SharedPreferences  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?  
 ): View {  
 \_binding = FragmentMyTicketsBinding.inflate(inflater, container, false)  
 return binding.*root* }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 pref = requireContext().getSharedPreferences("PREF", Context.*MODE\_PRIVATE*)  
 db = DatabaseClient.getInstance(requireContext())  
 username = pref.getString("Online", null).*toString*()  
  
 // Используем MyTicketAdapter вместо TicketAdapter  
 ticketAdapter = MyTicketAdapter(*emptyList*()) **{** ticket **->** // Обработчик нажатия на кнопку "Вернуть средства"  
 onRefundTicketClicked(ticket)  
 **}** binding.recyclerViewTickets.*layoutManager* = LinearLayoutManager(requireContext())  
 binding.recyclerViewTickets.*adapter* = ticketAdapter  
  
 loadTickets()  
 }  
  
 private fun loadTickets() {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** // Получаем список билетов, купленных текущим пользователем  
 val tickets = db.ticketDao().getBoughtTicketsByUser(username)  
  
 // Получаем подробную информацию о фильмах, связанных с билетами  
 val ticketsWithMovies = tickets.*map* **{** ticket **->** val movie = db.movieDao().getMovieById(ticket.movieId)  
 TicketWithMovie(ticket, movie)  
 **}** withContext(Dispatchers.Main) **{** // Обновляем адаптер с новыми данными  
 ticketAdapter.updateTickets(ticketsWithMovies)  
 **}  
 }** }  
  
 private fun onRefundTicketClicked(ticket: TicketWithMovie) {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** // Отменяем покупку билета (обнуляем ownerUsername)  
 db.ticketDao().refundTicket(ticket.ticket.id)  
  
 withContext(Dispatchers.Main) **{** Toast.makeText(requireContext(), "Ticket refunded", Toast.*LENGTH\_SHORT*).show()  
 loadTickets() // Перезагружаем список билетов  
 **}  
 }** }  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
}

Данный класс обеспечивает отображение купленных билетов у пользователя, а также возможность возврата средств и последующем возвращении билета в оборот продаж

Класс Registration

package com.example.megogo  
  
import android.content.SharedPreferences  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.view.View  
import android.widget.EditText  
import android.widget.Spinner  
import android.widget.Toast  
import androidx.lifecycle.*lifecycleScope*import androidx.room.Room  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.GlobalScope  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class Registration : AppCompatActivity() {  
 private lateinit var email : EditText  
 private lateinit var login : EditText  
 private lateinit var pass : EditText  
 private lateinit var repPass : EditText  
 private lateinit var role : Spinner  
 private lateinit var db: CinemaDatabase  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_registration*)  
  
 email = findViewById(R.id.*EmailEdit*)  
 login = findViewById(R.id.*LoginEdit*)  
 pass = findViewById(R.id.*PassEdit*)  
 repPass = findViewById(R.id.*RepPassEdit*)  
 role = findViewById(R.id.*RoleSpin*)  
  
 db = DatabaseClient.getInstance(*applicationContext*)  
 }  
  
 // Функция для проверки email  
 fun isValidEmail(email: String): Boolean {  
 val emailRegex = "^[A-Za-z0-9+\_.-]+@[A-Za-z0-9.-]+\$"  
 return email.*matches*(Regex(emailRegex))  
 }  
  
 // Функция для проверки логина  
 fun isValidLogin(login: String): Boolean {  
 // Логин должен быть длиной от 3 до 20 символов и содержать только латинские буквы и цифры  
 val loginRegex = "^[A-Za-z0-9]{3,20}\$"  
 return login.*matches*(Regex(loginRegex))  
 }  
  
 // Функция для проверки пароля  
 fun isValidPassword(password: String): Boolean {  
 // Пароль должен содержать хотя бы одну заглавную букву, одну цифру и иметь длину от 8 до 20 символов  
 val passwordRegex = "^(?=.\*[A-Z])(?=.\*\\d)[A-Za-z\\d]{8,20}\$"  
 return password.*matches*(Regex(passwordRegex))  
 }  
  
 private fun registerUser(username: String, email: String, password: String, role: String) {  
 // Создаём нового пользователя  
 val newUser = User(username = username, email = email, password = password, role = role)  
  
 // Вставляем пользователя в базу данных  
 GlobalScope.*launch* **{** db.userDao().insertUser(newUser)  
 runOnUiThread **{** Toast.makeText(this@Registration, "Registration successful!", Toast.*LENGTH\_SHORT*).show()  
 **}  
 }** }  
  
 fun Register(view: View) {  
 if (email.*text*.*isNotEmpty*() && login.*text*.*isNotEmpty*() && pass.*text*.*isNotEmpty*() && repPass.*text*.*isNotEmpty*()) {  
 if (isValidEmail(email.*text*.toString())) {  
 if (isValidLogin(login.*text*.toString())) {  
 if (isValidPassword(pass.*text*.toString())) {  
 if (pass.*text*.toString() == repPass.*text*.toString()) {  
 // Используем lifecycleScope для работы в фоновом потоке  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** // Проверка существования пользователя с таким логином или email  
 val user = db.userDao().getUserByUsernameOrEmail(login.*text*.toString(), email.*text*.toString())  
  
 // Проверка, если пользователь не найден (то есть можно зарегистрировать)  
 if (user == null) {  
 // Если аккаунт не существует, регистрируем пользователя  
 registerUser(login.*text*.toString(), email.*text*.toString(), pass.*text*.toString(), role.*selectedItem*.toString())  
  
 // Очистка полей  
 withContext(Dispatchers.Main) **{** email.*text*.clear()  
 login.*text*.clear()  
 pass.*text*.clear()  
 repPass.*text*.clear()  
 **}** } else {  
 // Если пользователь уже существует с таким логином или email  
 withContext(Dispatchers.Main) **{** Toast.makeText(  
 this@Registration,  
 "An account with such email or such login already exists",  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** }  
 **}** } else {  
 Toast.makeText(this@Registration, "The passwords aren't matching", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 Toast.makeText(this@Registration, "Invalid Password", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 Toast.makeText(this@Registration, "Invalid Login", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 Toast.makeText(this@Registration, "Invalid Email", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 Toast.makeText(this@Registration, "Fill in the blanks", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
}

Данный класс позволяет пользователю зарегистрироваться в системе, то есть, внести данные о себе в БД, чтобы программа понимала, каким образом взаимодействовать с пользователем, например, как с клиентов или как с администратором кинотеатра

Класс RemovingMovieFragment

package com.example.megogo  
  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import androidx.lifecycle.*lifecycleScope*import androidx.recyclerview.widget.LinearLayoutManager  
import com.example.megogo.databinding.FragmentRemovingMovieBinding  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class RemovingMovieFragment : Fragment() {  
  
 private var \_binding: FragmentRemovingMovieBinding? = null  
 private val binding get() = \_binding!!  
  
 private lateinit var adapter: MovieAdapterAdm  
 private val movies = *mutableListOf*<Movie>()  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View {  
 \_binding = FragmentRemovingMovieBinding.inflate(inflater, container, false)  
 return binding.*root* }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 adapter = MovieAdapterAdm(movies) **{** movie **->** deleteMovieFromDatabase(movie)  
 **}** binding.recyclerViewMovies.*layoutManager* = LinearLayoutManager(requireContext())  
 binding.recyclerViewMovies.*adapter* = adapter  
  
 loadMoviesFromDatabase()  
 }  
  
 private fun loadMoviesFromDatabase() {  
 *lifecycleScope*.*launch* **{** val database = DatabaseClient.getInstance(requireContext())  
 val movieList = withContext(Dispatchers.IO) **{** database.movieDao().getAllMovies()  
 **}** movies.clear()  
 movies.addAll(movieList)  
 adapter.notifyDataSetChanged()  
 **}** }  
  
 private fun deleteMovieFromDatabase(movie: Movie) {  
 *lifecycleScope*.*launch* **{** val database = DatabaseClient.getInstance(requireContext())  
 withContext(Dispatchers.IO) **{** database.ticketDao().deleteTicketsByMovieId(movie.id) // Удаляем билеты  
 database.movieDao().deleteMovie(movie) // Удаляем фильм  
 **}** withContext(Dispatchers.Main) **{** movies.remove(movie)  
 adapter.notifyDataSetChanged()  
 Toast.makeText(requireContext(), "The movie and its tickets were removed from rental", Toast.*LENGTH\_SHORT*).show()  
 **}  
 }** }  
  
  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
}

Данный класс отображает фильмы в прокате и позволяет администратору удалить фильм из проката, а так же все билеты, выпущенные на этот фильм из базы данных.

Класс SearchingMovieFragment

package com.example.megogo  
  
import android.content.Intent  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.AdapterView  
import android.widget.ArrayAdapter  
import androidx.fragment.app.Fragment  
import androidx.lifecycle.*lifecycleScope*import androidx.recyclerview.widget.LinearLayoutManager  
import com.example.megogo.databinding.FragmentSearchingMovieBinding  
import com.google.android.material.snackbar.Snackbar  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class SearchingMovieFragment : Fragment() {  
  
 private var \_binding: FragmentSearchingMovieBinding? = null  
 private val binding get() = \_binding!!  
  
 private lateinit var movieAdapter: MovieAdapter  
 private lateinit var genres: List<String>  
 private var allMovies: List<MovieItem> = *listOf*()  
 private var filteredMovies: List<MovieItem> = *listOf*()  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View {  
 \_binding = FragmentSearchingMovieBinding.inflate(inflater, container, false)  
 return binding.*root* }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 // Обработчик клика по фильму, передача данных фильма  
 movieAdapter = MovieAdapter(*listOf*()) **{** movie **->** val intent = Intent(requireContext(), MovieDetailsActivity::class.*java*).*apply* **{** putExtra("movieId", movie.id) // Передаем только movieId для запроса билетов  
 **}** startActivity(intent)  
 **}** binding.recyclerViewMovies.*apply* **{** *layoutManager* = LinearLayoutManager(requireContext())  
 *adapter* = movieAdapter  
 **}** // Получение фильмов из базы данных  
 loadMovies()  
  
 // Настройка Spinner для фильтрации  
 genres = *listOf*(  
 "All genres", "Action", "Horror", "Drama", "Comedy",  
 "Thriller", "Mystery", "Romance", "Crime", "Sci-Fi",  
 "Animation", "Adventure"  
 )  
 val spinnerAdapter = ArrayAdapter(requireContext(), android.R.layout.*simple\_spinner\_item*, genres)  
 spinnerAdapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*)  
 binding.spinnerGenres.*adapter* = spinnerAdapter  
  
 binding.spinnerGenres.*onItemSelectedListener* = object : AdapterView.OnItemSelectedListener {  
 override fun onItemSelected(parent: AdapterView<\*>?, view: View?, position: Int, id: Long) {  
 filterMoviesByGenre(genres[position])  
 }  
  
 override fun onNothingSelected(parent: AdapterView<\*>?) {}  
 }  
  
 // Обработка нажатия кнопки поиска  
 binding.searchBut.setOnClickListener **{** val searchQuery = binding.searchEdit.*text*.toString()  
 if (searchQuery.*isNotEmpty*()) {  
 searchMoviesByTitle(searchQuery)  
 }  
 else  
 Snackbar.make(view, "Fill the empty blank", Snackbar.*LENGTH\_SHORT*).show()  
 **}** // Обработка нажатия кнопки очистки  
 binding.clearBut.setOnClickListener **{** binding.searchEdit.*text*.clear()  
 loadMovies() // Загружаем все фильмы из базы данных  
 **}** }  
  
 // Подгрузка фильмов в RecyclerView из БД  
 private fun loadMovies() {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** val db = DatabaseClient.getInstance(requireContext())  
 val moviesFromDb = db.movieDao().getAllMovies().*map* **{** MovieItem(**it**.id, **it**.title, **it**.genre, **it**.poster, **it**.year, **it**.description)  
 **}** withContext(Dispatchers.Main) **{** allMovies = moviesFromDb  
 filteredMovies = allMovies // Изначально показываем все фильмы  
 movieAdapter.updateMovies(filteredMovies)  
 **}  
 }** }  
  
 // Фильтрация фильмов по жанру  
 private fun filterMoviesByGenre(genre: String) {  
 filteredMovies = if (genre == "All genres") {  
 allMovies  
 } else {  
 allMovies.*filter* **{ it**.genre.*contains*(genre, ignoreCase = true) **}** }  
 movieAdapter.updateMovies(filteredMovies)  
 }  
  
 // Поиск фильмов по названию  
 private fun searchMoviesByTitle(title: String) {  
 filteredMovies = allMovies.*filter* **{ it**.title.*contains*(title, ignoreCase = true) **}** movieAdapter.updateMovies(filteredMovies)  
 }  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
}

Этот класс позволяет пользователю просмотреть все фильмы, имеющиеся в прокате, а также отфильтровать их показ по жанрам, также класс позволяет пользователю найти определённый фильм, на который хочет пойти пользователь и очистить результаты поиска, показав все фильмы, имеющиеся в прокате. Так же, по нажатию на конкретный фильм, осуществляется переход на экран покупки билетов с последующей передачей данных о фильме и имеющихся на него билетов.

## 1.4 Используемые библиотеки

*implementation*("com.google.android.material:material:1.9.0")  
*implementation*("androidx.room:room-runtime:2.5.0")  
*annotationProcessor*("androidx.room:room-compiler:2.5.0")  
*kapt*("androidx.room:room-compiler:2.5.0")  
*implementation*("androidx.room:room-ktx:2.5.0")  
*implementation*("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.2")  
*implementation*("androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1")  
*implementation*("androidx.navigation:navigation-fragment-ktx:2.5.3")  
*implementation*("androidx.navigation:navigation-ui-ktx:2.5.3")  
*implementation*("androidx.recyclerview:recyclerview:1.2.1")  
*implementation*("androidx.constraintlayout:constraintlayout:2.1.3")  
*implementation*("com.squareup.picasso:picasso:2.8")  
*implementation*("com.android.volley:volley:1.2.1")  
*implementation*("androidx.appcompat:appcompat:1.7.0")  
*implementation*("com.squareup.retrofit2:converter-gson:2.9.0")  
*testImplementation*("junit:junit:4.13.2")  
*androidTestImplementation*("androidx.test.ext:junit:1.1.3")  
*androidTestImplementation*("androidx.test.espresso:espresso-core:3.4.0")  
*androidTestImplementation*("androidx.test:runner:1.4.0")  
*androidTestImplementation*("androidx.test:rules:1.4.0")

## 1.5 Используемые инструменты

Язык программирования – Kotlin

Язык гиперболизированной разметки текста – xml

Среда разработки – Android Studio

## 1.6 Описание пользовательского интерфейса

При входе в приложение, в зависимости от того, был ли авторизован пользователь ранее, открывается: экран авторизации, меню клиента, меню регистрации. Так же от роли пользователя, зависит то, на какое меню перебросит пользователя. На экране авторизации расположены 2 поля: логин или электронная почта, пароль, также на экране расположена кнопка входа. После заполнения данных и по нажатию на кнопку пользователя перебрасывает в меню. Снизу расположен текст, при нажатии на который пользователя перебрасывает на экран регистрации.

На экране регистрации расположены несколько полей: Почта, Логин, Пароль, Подтверждение пароля и spinner с элементами: User и Admin, после заполнения всех полей и выбора роли нажимается кнопка регистрации, после чего данные о пользователе заносятся в БД.

При переходе в меню пользователя есть разветвление на клиента и администратора приложения.

Активность администратора: внизу расположено меню с выбором фрагментов для контейнера на экране: добавление и удаление.

На фрагменте добавления расположено поля поиска фильма и кнопка, при заполнении полей и нажатие на кнопку, программа через API-запрос на Omdb ищет фильм по названию или по названию и году, в зависимости от того, пустое ли поле года. Информация о фильме отображается в ScrollView, снизу фрагмента находится кнопка добавления фильма в прокат, которая осуществляет занесение фильма и 50 билетов на него в прокат.

На фрагменте удаления в recyclerView отображаются все фильмы, уже имеющиеся в прокате с кнопкой удаления, при нажатии на кнопку, фильм удаляется из БД вместе со всеми билетами, выпущенными к нему, в том числе и уже купленные клиентами.

Активность клиента: на экране расположено меню для навигации с пунктами: поиск и мои билеты. В контейнере отображается фрагмент, выбранный в меню.

На фрагменте поиска отображается список фильмов, имеющихся в прокате в recyclerView. Сверху имеется spinner, который обеспечивает фильтрацию фильмов по жанру. Ещё выше расположено поле поиска, на котором есть EditText и 2 кнопки, при заполнении поля и нажатии на кнопку поиск, в recyclerView отображаются все фильмы в прокате, которые содержат то, что было написано пользователем в поле. При нажатии на кнопку очистить, результаты поиска очищаются и показываются все фильмы, имеющиеся в прокате. При нажатии на какой-либо фильм происходит переход на экран выбора и покупки билетов.

На экране покупки билетов отображается информация о фильме, который был выбран и информация о всех его билетах, доступных в продаже, расположенных в recyclerView. Напротив каждого билета расположена кнопка купить, при нажатии на которую билет числится как билет с имеющимся владельцем и обновляется в БД, как недоступный для продажи.

На фрагменте Мои билеты расположен recyclerView, в котором расположены билеты, купленные пользователем с информацией о: ряде, месте, названии фильма и годе выпуска фильма. Напротив каждого билета имеется кнопка вернуть билет, при нажатии на которую осуществляется обновление информации о билете в БД и сброс владельца билета и билет возвращается в продажу.

1.7 Приложение (pr screen экранов)









