Министерство образования и молодежной политики Свердловской области



ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Отчёт по УП

Задание 5. Приложение «Онлайн-библиотека»

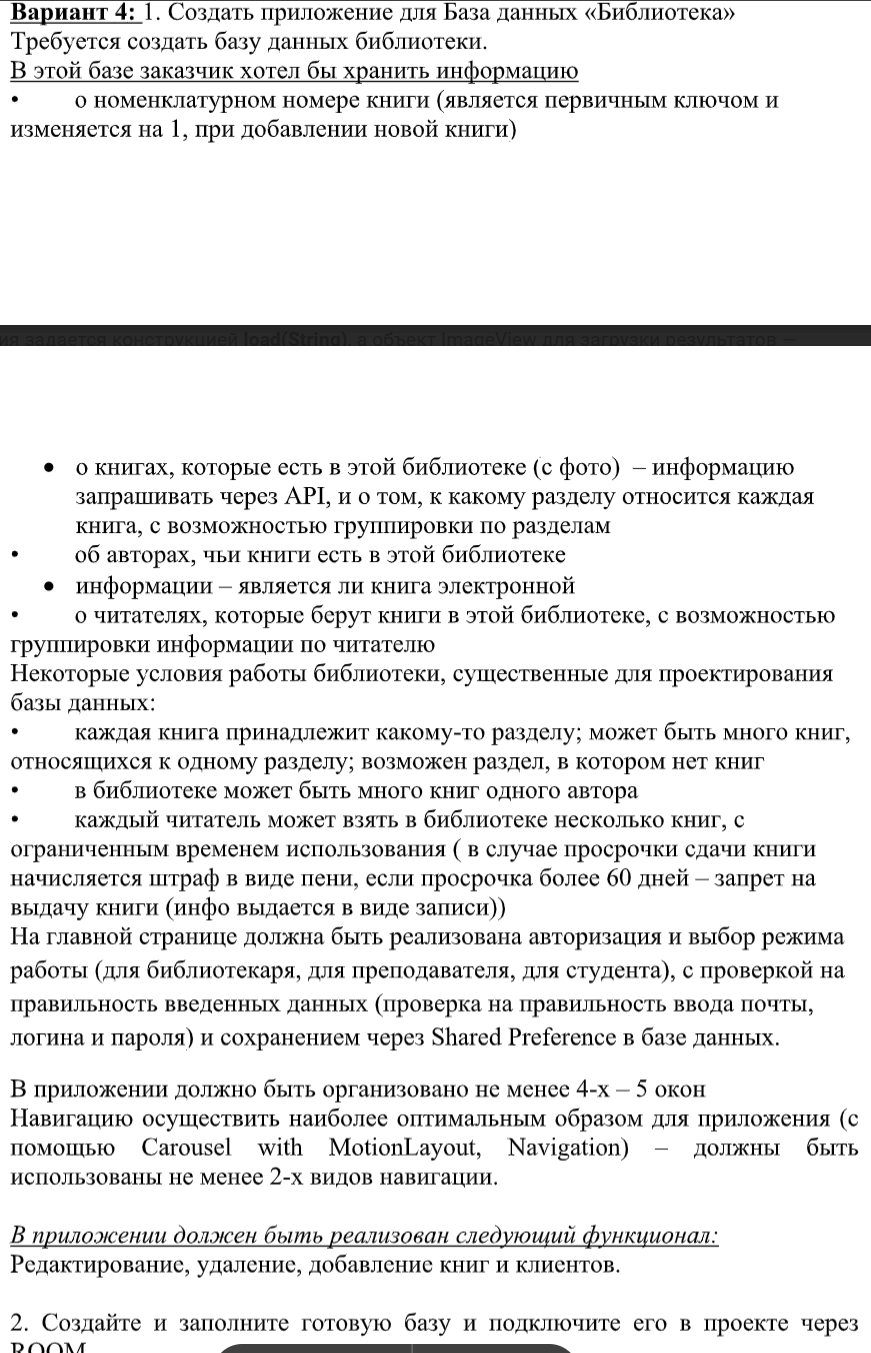
Выполнил: Сюбаев Андрей

Группа: ПР-32

Преподаватель: Мирошниченко Г.В

2025

**Описание задачи:**

****

**Входные данные:**

fisrtName-имя пользователя

lastName-фамилия пользователя

email- почта пользователя

login- логин пользователя

password - пароль пользователя

role- роль пользователя

title-название книги

author- автор книги

section- раздел книги

days- дни на сколько книгу можно взять

isElectronic- электронная книга или нет

**Выходные данные:**

title-название книги

author- автор книги

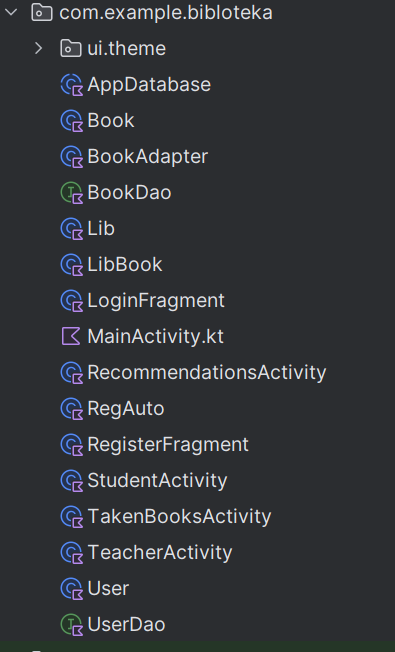
section- раздел книги

coverResId- обложка книги

days- дни на сколько книгу можно взять

isElectronic- электронная книга или нет

**Структура проекта:**

****

**Листинг программы:**

**AppDatabase:**

@Database(entities = [User::class, Book::class], version = 1)  
abstract class AppDatabase : RoomDatabase() {  
 abstract fun bookDao(): BookDao  
 abstract fun userDao(): UserDao  
 companion object {  
 @Volatile  
 private var INSTANCE: AppDatabase? = null  
 }  
}

**Book:**

@Entity(tableName = "books")  
data class Book(  
 @PrimaryKey(autoGenerate = true) val id: Int = 0,  
 val title: String,  
 val author: String,  
 @DrawableRes val coverResId: Int,  
 val section: String,  
 val days: Int,  
 val isElectronic: Boolean,  
 var isSelected: Boolean = false,  
 var isTaken: Boolean = false,  
 var recommendedTo: String? = null  
)

**BookAdapter:**

class BookAdapter : RecyclerView.Adapter<BookAdapter.BookViewHolder>() {  
  
 private val books = *mutableListOf*<Book>()  
  
 // Метод для обновления списка книг  
 fun submitList(newBooks: List<Book>) {  
 books.clear()  
 books.addAll(newBooks)  
 notifyDataSetChanged()  
 }  
 fun getSelectedBook(): Book? {  
 return books.*find* **{ it**.isSelected **}** }  
  
 // Создание нового ViewHolder  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookViewHolder {  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_book*, parent, false)  
 return BookViewHolder(view)  
 }  
  
 // Привязка данных к ViewHolder  
 override fun onBindViewHolder(holder: BookViewHolder, position: Int) {  
 val book = books[position]  
 holder.bind(book)  
 holder.itemView.setOnClickListener **{** book.isSelected = !book.isSelected   
 notifyItemChanged(position)   
 **}** }  
 fun removeBook(book: Book) {  
 val index = books.indexOf(book)  
 if (index != -1) {  
 books.removeAt(index)  
 notifyItemRemoved(index)  
 }  
 }  
 // Возвращает количество элементов в списке  
 override fun getItemCount(): Int = books.size  
  
 // ViewHolder для книги  
 class BookViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
 private val coverImageView: ImageView = itemView.findViewById(R.id.*bookCoverImageView*)  
 private val titleTextView: TextView = itemView.findViewById(R.id.*bookTitleTextView*)  
 private val sectionTextView: TextView = itemView.findViewById(R.id.*bookSectionTextView*)  
 private val daysTextView: TextView = itemView.findViewById(R.id.*bookDaysTextView*)  
 private val authorTextView: TextView = itemView.findViewById(R.id.*bookAuthorTextView*)  
  
 // Метод для привязки данных книги к элементам интерфейса  
 fun bind(book: Book) {  
 titleTextView.*text* = book.title  
 sectionTextView.*text* = "Раздел: ${book.section}"  
 authorTextView.*text* = "Автор: ${book.author}" // Отображение автора  
 daysTextView.*text* = "Дней: ${book.days}"  
  
 // Загрузка обложки книги с помощью Glide  
 coverImageView.setImageResource(book.coverResId)  
 }  
 }  
}

**BookDao:**

@Dao  
interface BookDao {  
 @Insert  
 suspend fun insert(book: Book)  
  
 @Query("SELECT \* FROM books")  
 suspend fun getAllBooks(): List<Book>  
 @Delete  
 suspend fun delete(book: Book)  
 @Update  
 suspend fun update(book: Book)  
 @Query("SELECT \* FROM books WHERE recommendedTo IS NOT NULL")  
 suspend fun getRecommendedBooks(): List<Book>  
}

**Lib:**

class Lib : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var titleEditText: EditText  
 private lateinit var searchButton: Button  
 private lateinit var coverImageView: ImageView  
 private lateinit var titleTextView: TextView  
 private lateinit var sectionEditText: EditText  
 private lateinit var daysEditText: EditText  
 private lateinit var electronicCheckBox: CheckBox  
 private lateinit var saveButton: Button  
 private lateinit var nazad: Button  
 var coverResId = R.drawable.*icon* private val bookCollection = *listOf*(  
 Book(title = "Гарри Поттер", author = "Дж. К. Роулинг", coverResId = R.drawable.*gg*, section = "", days = 0, isElectronic = false),  
 Book(title = "Война и мир", author = "Лев Толстой", coverResId = R.drawable.*ww*, section = "", days = 0, isElectronic = false),  
 Book(title = "Преступление и наказание", author = "Фёдор Достоевский", coverResId = R.drawable.*pp*, section = "", days = 0, isElectronic = false),  
 Book(title = "1984", author = "Джордж Оруэлл", coverResId = R.drawable.*oo*, section = "", days = 0, isElectronic = false)  
 )  
  
  
 private var bookCoverUrl: String? = null // URL обложки книги  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_lib*)  
  
 // Инициализация базы данных  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
  
 // Инициализация элементов интерфейса  
 titleEditText = findViewById(R.id.*titleEditText*)  
 searchButton = findViewById(R.id.*searchButton*)  
 coverImageView = findViewById(R.id.*coverImageView*)  
 titleTextView = findViewById(R.id.*titleTextView*)  
 sectionEditText = findViewById(R.id.*sectionEditText*)  
 daysEditText = findViewById(R.id.*daysEditText*)  
 electronicCheckBox = findViewById(R.id.*electronicCheckBox*)  
 saveButton = findViewById(R.id.*saveButton*)  
 nazad = findViewById(R.id.*nazadButton*)  
 // Обработчик кнопки поиска  
 searchButton.setOnClickListener **{** val title = titleEditText.*text*.toString().*trim*()  
 if (title.*isNotEmpty*()) {  
 searchBook(title)  
 } else {  
 Toast.makeText(this, "Введите название книги", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** // Обработчик кнопки сохранения  
 saveButton.setOnClickListener **{** saveBook()  
 **}** nazad.setOnClickListener **{** val intent = Intent(this, LibBook::class.*java*)  
 startActivity(intent)   
 **}** }  
 private fun searchBook(title: String) {  
 val book = bookCollection.*find* **{ it**.title.*equals*(title, ignoreCase = true) **}** if (book != null) {  
 // Обновляем интерфейс  
 titleTextView.*text* = book.title  
 findViewById<TextView>(R.id.*authorTextView*).*text* = book.author  
 coverImageView.setImageResource(book.coverResId) // Используем идентификатор ресурса  
 } else {  
 Toast.makeText(this, "Книга не найдена", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
  
 private fun saveBook() {  
 val title = titleTextView.*text*.toString().*trim*()  
 val author = findViewById<TextView>(R.id.*authorTextView*).*text*.toString().*trim*()  
 val section = sectionEditText.*text*.toString().*trim*()  
 val days = daysEditText.*text*.toString().*trim*()  
 val isElectronic = electronicCheckBox.*isChecked* if (title.*isEmpty*() || author.*isEmpty*() || section.*isEmpty*() || days.*isEmpty*()) {  
 Toast.makeText(this, "Заполните все поля", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
  
 *lifecycleScope*.*launch* **{** val book = Book(  
 title = title,  
 author = author,  
 coverResId = coverResId,  
 section = section,  
 days = days.*toInt*(),  
 isElectronic = isElectronic  
 )  
 db.bookDao().insert(book)  
 Toast.makeText(this@Lib, "Книга сохранена", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
}

**LibBook:**

class LibBook : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var booksRecyclerView: RecyclerView  
 private lateinit var adapter: BookAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_lib\_book*)  
  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
 booksRecyclerView = findViewById(R.id.*booksRecyclerView*)  
 booksRecyclerView.*layoutManager* = LinearLayoutManager(this)  
  
 adapter = BookAdapter()  
 booksRecyclerView.*adapter* = adapter  
  
 loadBooks()  
  
 val button: Button = findViewById(R.id.*addBookButton*)  
 button.setOnClickListener **{** // Создаем намерение для перехода на SecondActivity  
 val intent = Intent(this, Lib::class.*java*)  
 startActivity(intent) // Запускаем новую активность  
 **}** val deleteButton: Button = findViewById(R.id.*deleteBookButton*)  
 deleteButton.setOnClickListener **{** val selectedBook = adapter.getSelectedBook()  
 if (selectedBook != null) {  
 *lifecycleScope*.*launch* **{** db.bookDao().delete(selectedBook)  
 adapter.removeBook(selectedBook)  
 Toast.makeText(this@LibBook, "Книга удалена", Toast.*LENGTH\_SHORT*).show()  
 **}** } else {  
 Toast.makeText(this@LibBook, "Выберите книгу для удаления", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** }  
 private fun loadBooks() {  
 *lifecycleScope*.*launch* **{** val books = db.bookDao().getAllBooks()  
 adapter.submitList(books)  
 **}** }  
  
  
}

**LoginFragment:**

class LoginFragment : Fragment() {  
 // *TODO: Rename and change types of parameters* private var param1: String? = null  
 private var param2: String? = null  
 /\* lateinit var logbutton: Button\*/  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *arguments*?.*let* **{** param1 = **it**.getString(*ARG\_PARAM1*)  
 param2 = **it**.getString(*ARG\_PARAM2*)  
  
 **}** }  
  
 private lateinit var db: AppDatabase  
  
 @SuppressLint("MissingInflatedId")  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 val view = inflater.inflate(R.layout.*activity\_login\_fragment*, container, false)  
 db = Room.databaseBuilder(requireContext(), AppDatabase::class.*java*, "user-db").build()  
  
 val roleSpinner: Spinner = view.findViewById(R.id.*roleSpinner*)  
 val roles = *arrayOf*("Библиотекарь", "Преподаватель", "Студент")  
 val adapter = ArrayAdapter(requireContext(), android.R.layout.*simple\_spinner\_item*, roles)  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*)  
 roleSpinner.*adapter* = adapter  
  
 view.findViewById<Button>(R.id.*loginButton*).setOnClickListener **{** val email = view.findViewById<EditText>(R.id.*emailEditText*).*text*.toString()  
 val login = view.findViewById<EditText>(R.id.*loginEditText*).*text*.toString()  
 val password = view.findViewById<EditText>(R.id.*passwordEditText*).*text*.toString()  
 val role = roleSpinner.*selectedItem*.toString()  
 val logbutton: Button = view.findViewById(R.id.*loginButton*)  
  
 // Проверка введенных данных  
 if (email.*isNotEmpty*() && login.*isNotEmpty*() && password.*isNotEmpty*()) {  
 *lifecycleScope*.*launch* **{** // Проверка на наличие пользователей в базе данных  
 val userCount = db.userDao().getUserCount()  
 if (userCount == 0) {  
 Toast.makeText(requireContext(), "База данных пуста. Пожалуйста, зарегистрируйтесь.", Toast.*LENGTH\_SHORT*).show()  
 return@launch  
 }  
  
 val user = db.userDao().getUser(email, login, password, role)  
 if (user != null) {  
 when (role) {  
 "Библиотекарь" -> {  
 logbutton.setOnClickListener **{** // Создаем намерение для перехода на NewActivity  
 val intent = Intent(*activity*, LibBook::class.*java*)  
 startActivity(intent) // Запускаем новую активность  
 **}** }  
 "Преподаватель" -> {  
 logbutton.setOnClickListener **{** // Создаем намерение для перехода на NewActivity  
 val intent = Intent(*activity*, TeacherActivity::class.*java*)  
 startActivity(intent) // Запускаем новую активность  
 **}** }  
 "Студент" -> {  
 logbutton.setOnClickListener **{** // Создаем намерение для перехода на NewActivity  
 val intent = Intent(*activity*, StudentActivity::class.*java*)  
 startActivity(intent) // Запускаем новую активность  
 **}** }  
 else -> {  
 Toast.makeText(requireContext(), "Неизвестная роль", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
 } else {  
 Toast.makeText(requireContext(), "Неверные данные", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** } else {  
 Toast.makeText(requireContext(), "Пожалуйста, заполните все поля", Toast.*LENGTH\_SHORT*).show()  
 }  
  
  
  
 **}** // Обработка нажатия на кнопку "Зарегистрироваться"  
 view.findViewById<Button>(R.id.*registerButton*).setOnClickListener **{** val registerFragment = RegisterFragment()  
 *parentFragmentManager*.beginTransaction()  
 .replace(R.id.*fragmentContainer*, registerFragment)  
 .addToBackStack(null)  
 .commit()  
 **}** return view  
 }

**RecommendationsActivity:**

class RecommendationsActivity : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var recommendationsRecyclerView: RecyclerView  
 private lateinit var adapter: BookAdapter  
  
 @SuppressLint("MissingInflatedId")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_recommendations*)  
  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
 recommendationsRecyclerView = findViewById(R.id.*recommendationsRecyclerView*)  
  
 recommendationsRecyclerView.*layoutManager* = LinearLayoutManager(this)  
 adapter = BookAdapter()  
 recommendationsRecyclerView.*adapter* = adapter  
  
 loadRecommendations()  
 }  
  
 private fun loadRecommendations() {  
 *lifecycleScope*.*launch* **{** val recommendedBooks = db.bookDao().getAllBooks()   
 adapter.submitList(recommendedBooks)  
 **}** }  
}

**RegAuto:**

class RegAuto : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_reg\_auto*)  
 if (savedInstanceState == null) {  
 // Set LoginFragment as the default fragment  
 *supportFragmentManager*.beginTransaction()  
 .replace(R.id.*fragmentContainer*, LoginFragment())  
 .commit()  
 }  
 }  
}

**RegisterActivity:**

class RegisterFragment : Fragment() {  
 // *TODO: Rename and change types of parameters* private var param1: String? = null  
 private var param2: String? = null  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *arguments*?.*let* **{** param1 = **it**.getString(*ARG\_PARAM1*)  
 param2 = **it**.getString(*ARG\_PARAM2*)  
  
  
  
 **}** }  
  
 private lateinit var db: AppDatabase  
  
 @SuppressLint("MissingInflatedId")  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
  
  
  
  
 val view = inflater.inflate(R.layout.*activity\_register\_fragment*, container, false)  
 db = Room.databaseBuilder(requireContext(), AppDatabase::class.*java*, "user-db").build()  
  
 val roleSpinner: Spinner = view.findViewById(R.id.*roleSpinner*)  
 val roles = *arrayOf*("Библиотекарь", "Преподаватель", "Студент")  
 roleSpinner.*adapter* = ArrayAdapter(requireContext(), android.R.layout.*simple\_spinner\_item*, roles)  
  
 view.findViewById<Button>(R.id.*registerButton*).setOnClickListener **{** val lastName = view.findViewById<EditText>(R.id.*lastNameEditText*).*text*.toString().*trim*()  
 val firstName = view.findViewById<EditText>(R.id.*firstNameEditText*).*text*.toString().*trim*()  
 val email = view.findViewById<EditText>(R.id.*emailEditText*).*text*.toString().*trim*()  
 val login = view.findViewById<EditText>(R.id.*loginEditText*).*text*.toString().*trim*()  
 val password = view.findViewById<EditText>(R.id.*passwordEditText*).*text*.toString().*trim*()  
 val role = roleSpinner.*selectedItem*.toString()  
  
 // Проверка на пустые поля  
 if (lastName.*isEmpty*() || firstName.*isEmpty*() || email.*isEmpty*() || login.*isEmpty*() || password.*isEmpty*()) {  
 Toast.makeText(requireContext(), "Все поля должны быть заполнены", Toast.*LENGTH\_SHORT*).show()  
 return@setOnClickListener  
 }  
  
 // Проверка длины пароля  
 if (password.length < 8) {  
 Toast.makeText(requireContext(), "Пароль должен быть не менее 8 символов", Toast.*LENGTH\_SHORT*).show()  
 return@setOnClickListener  
 }  
  
 // Проверка формата email  
 if (!android.util.Patterns.*EMAIL\_ADDRESS*.matcher(email).matches()) {  
 Toast.makeText(requireContext(), "Введите корректный email", Toast.*LENGTH\_SHORT*).show()  
 return@setOnClickListener  
 }  
  
 *lifecycleScope*.*launch* **{** // Проверка, есть ли пользователь с таким email или логином  
 val existingUser = db.userDao().getUserByEmailOrLogin(email, login)  
 if (existingUser != null) {  
 Toast.makeText(requireContext(), "Пользователь с таким email или логином уже существует", Toast.*LENGTH\_SHORT*).show()  
 return@launch  
 }  
  
 // Если все проверки пройдены, регистрируем пользователя  
 val user = User(lastName = lastName, firstName = firstName, email = email, login = login, password = password, role = role)  
 db.userDao().insert(user)  
 Toast.makeText(requireContext(), "Регистрация успешна", Toast.*LENGTH\_SHORT*).show()  
  
 // Переход на экран входа  
 *parentFragmentManager*.beginTransaction()  
 .replace(R.id.*fragmentContainer*, LoginFragment())  
 .commit()  
 **}  
 }** // Обработка нажатия на кнопку "Назад"  
 view.findViewById<Button>(R.id.*nazadButton*).setOnClickListener **{** val loginFragment = LoginFragment()  
 *parentFragmentManager*.beginTransaction()  
 .replace(R.id.*fragmentContainer*, loginFragment )  
 .addToBackStack(null)  
 .commit()  
 **}** return view  
  
  
 }

**StudentActivity:**

class StudentActivity : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var booksRecyclerView: RecyclerView  
 private lateinit var adapter: BookAdapter  
 private lateinit var sectionSpinner: Spinner  
  
 @SuppressLint("MissingInflatedId")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_student*)  
  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
 booksRecyclerView = findViewById(R.id.*booksRecyclerView*)  
 sectionSpinner = findViewById(R.id.*sectionSpinner*)  
  
 booksRecyclerView.*layoutManager* = LinearLayoutManager(this)  
 adapter = BookAdapter()  
 booksRecyclerView.*adapter* = adapter  
  
 loadBooks()  
  
 // Настройка Spinner  
 setupSectionSpinner()  
  
 // Обработка нажатия кнопки "Взять книгу"  
 findViewById<Button>(R.id.*takeBookButton*).setOnClickListener **{** val selectedBook = adapter.getSelectedBook() // Получаем выбранную книгу  
 if (selectedBook != null) {  
 takeBook(selectedBook)  
 } else {  
 Toast.makeText(this, "Выберите книгу для взятия", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** findViewById<Button>(R.id.*viewTakenBooksButton*).setOnClickListener **{** navigateToTakenBooks()  
 **}** findViewById<Button>(R.id.*recommendationsButton*).setOnClickListener **{** loadRecommendedBooks()  
 **}** findViewById<Button>(R.id.*takeBookButton*).setOnClickListener **{** val selectedBook = adapter.getSelectedBook()  
 if (selectedBook != null) {  
 *lifecycleScope*.*launch* **{** selectedBook.isTaken = true  
 db.bookDao().update(selectedBook)  
 Toast.makeText(this@StudentActivity, "Книга '${selectedBook.title}' добавлена в взятые", Toast.*LENGTH\_SHORT*).show()  
 **}** } else {  
 Toast.makeText(this@StudentActivity, "Выберите книгу для взятия", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** }  
 private fun loadRecommendedBooks() {  
 *lifecycleScope*.*launch* **{** val recommendedBooks = db.bookDao().getRecommendedBooks()  
 adapter.submitList(recommendedBooks)  
 **}** }  
 private fun loadBooks() {  
 *lifecycleScope*.*launch* **{** val books = db.bookDao().getAllBooks()  
 adapter.submitList(books)  
 **}** }  
 private fun navigateToTakenBooks() {  
 val intent = Intent(this, TakenBooksActivity::class.*java*)  
 startActivity(intent)  
 }  
  
 private fun setupSectionSpinner() {  
 // Здесь можно добавить логику для заполнения Spinner разделами книг  
 }  
  
 private fun takeBook(book: Book) {  
 // Логика для добавления книги в список взятых книг  
 Toast.makeText(this, "Книга '${book.title}' взята", Toast.*LENGTH\_SHORT*).show()  
 }  
}

**TakenBooksActivity:**

class TakenBooksActivity : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var takenBooksRecyclerView: RecyclerView  
 private lateinit var adapter: BookAdapter  
  
 @SuppressLint("MissingInflatedId")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_taken\_books*)  
  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
 takenBooksRecyclerView = findViewById(R.id.*takenBooksRecyclerView*)  
  
 takenBooksRecyclerView.*layoutManager* = LinearLayoutManager(this)  
 adapter = BookAdapter()  
 takenBooksRecyclerView.*adapter* = adapter  
  
 loadTakenBooks()  
 }  
  
 private fun loadTakenBooks() {  
 *lifecycleScope*.*launch* **{** val takenBooks = db.bookDao().getAllBooks()  
 adapter.submitList(takenBooks)  
 **}** }  
}

**TeacherActivity:**

class TeacherActivity : AppCompatActivity() {  
  
 private lateinit var db: AppDatabase  
 private lateinit var booksRecyclerView: RecyclerView  
 private lateinit var adapter: BookAdapter  
 private lateinit var sectionSpinner: Spinner  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_teacher*)  
  
 db = Room.databaseBuilder(*applicationContext*, AppDatabase::class.*java*, "book-db").build()  
 booksRecyclerView = findViewById(R.id.*booksRecyclerView*)  
 sectionSpinner = findViewById(R.id.*sectionSpinner*)  
  
 booksRecyclerView.*layoutManager* = LinearLayoutManager(this)  
 adapter = BookAdapter()  
 booksRecyclerView.*adapter* = adapter  
  
 loadBooks()  
  
 // Настройка Spinner  
 setupSectionSpinner()  
  
 findViewById<Button>(R.id.*recommendButton*).setOnClickListener **{** val selectedBook = adapter.getSelectedBook()  
 if (selectedBook != null) {  
 val dialogView = *layoutInflater*.inflate(R.layout.*dialog\_recommendation*, null)  
 val userEditText = dialogView.findViewById<EditText>(R.id.*userEditText*)  
  
 AlertDialog.Builder(this)  
 .setTitle("Рекомендовать книгу")  
 .setView(dialogView)  
 .setPositiveButton("Отправить") **{** \_, \_ **->** val username = userEditText.*text*.toString()  
 if (username.*isNotEmpty*()) {  
 *lifecycleScope*.*launch* **{** selectedBook.recommendedTo = username  
 db.bookDao().update(selectedBook)  
 Toast.makeText(this@TeacherActivity, "Книга рекомендована $username", Toast.*LENGTH\_SHORT*).show()  
 **}** } else {  
 Toast.makeText(this@TeacherActivity, "Введите имя ученика", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** .setNegativeButton("Отмена", null)  
 .show()  
 } else {  
 Toast.makeText(this, "Выберите книгу для рекомендации", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** findViewById<Button>(R.id.*viewTakenBooksButton*).setOnClickListener **{** navigateToTakenBooks()  
 **}** }  
  
 private fun loadBooks() {  
 *lifecycleScope*.*launch* **{** val books = db.bookDao().getAllBooks()  
 adapter.submitList(books)  
 **}** }  
 private fun navigateToTakenBooks() {  
 val intent = Intent(this, RecommendationsActivity::class.*java*)  
 startActivity(intent)  
 }  
  
 private fun setupSectionSpinner() {  
 // Здесь можно добавить логику для заполнения Spinner разделами книг  
 }  
  
 @SuppressLint("MissingInflatedId")  
 private fun showRecommendationDialog(book: Book) {  
 val dialogView = *layoutInflater*.inflate(R.layout.*dialog\_recommendation*, null)  
 val userEditText = dialogView.findViewById<EditText>(R.id.*userEditText*)  
  
 AlertDialog.Builder(this)  
 .setTitle("Рекомендовать книгу")  
 .setView(dialogView)  
 .setPositiveButton("Отправить") **{** \_, \_ **->** val username = userEditText.*text*.toString()  
 recommendBookToUser(book, username)  
 **}** .setNegativeButton("Отмена", null)  
 .show()  
 }  
  
 private fun recommendBookToUser(book: Book, username: String) {  
 // Логика для отправки рекомендации пользователю  
 Toast.makeText(this, "Книга '${book.title}' рекомендована пользователю $username", Toast.*LENGTH\_SHORT*).show()  
 }  
}

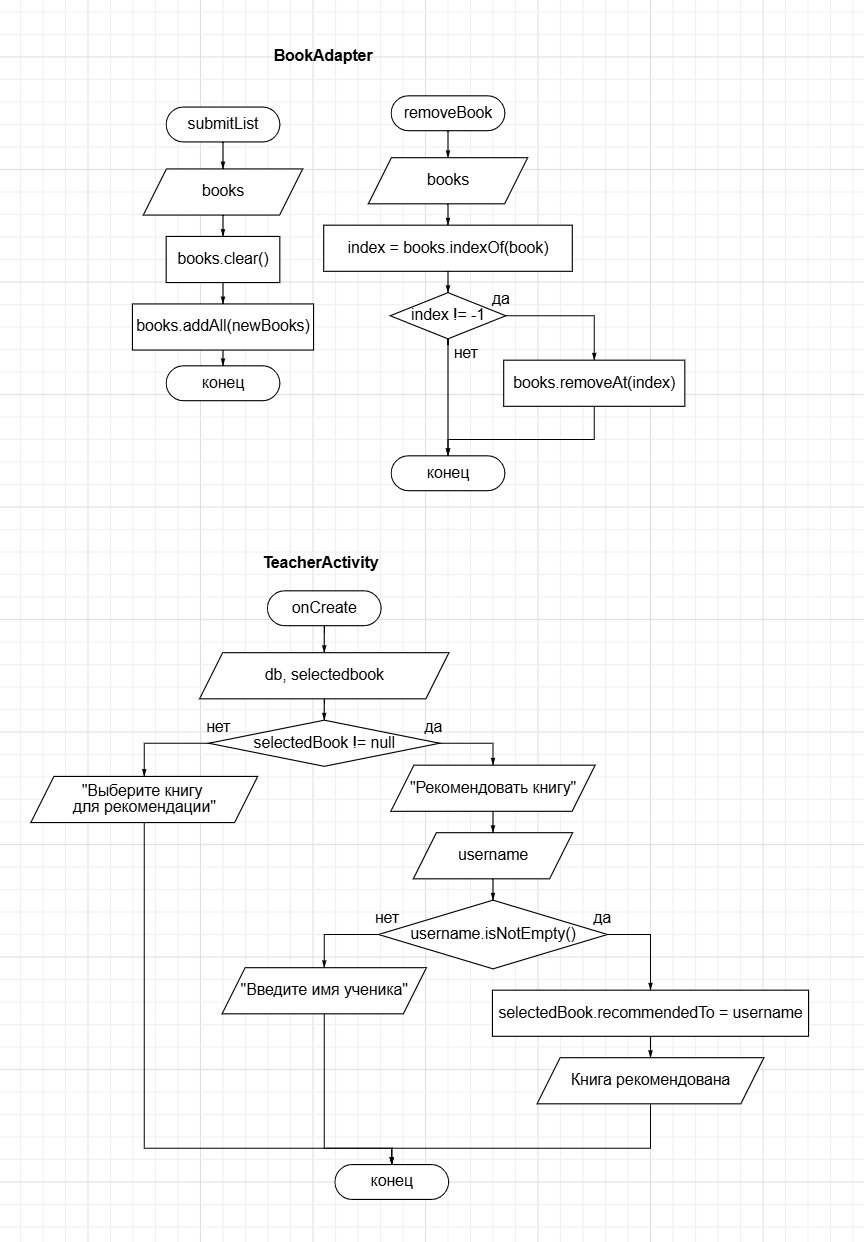
**User:**

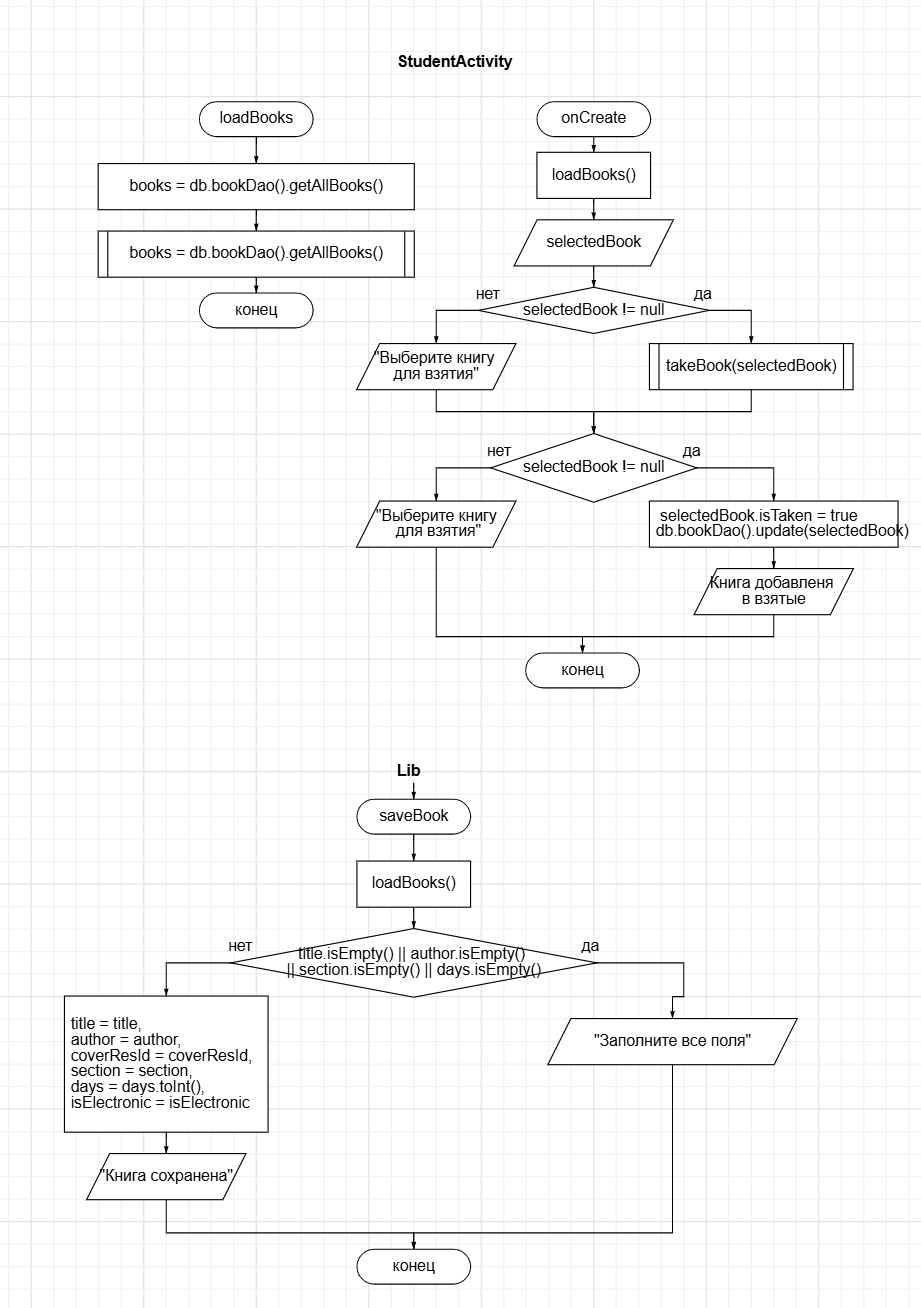
@Entity(tableName = "User")  
data class User(  
 @PrimaryKey(autoGenerate = true) val id: Int = 0,  
 val lastName: String,  
 val firstName: String,  
 val email: String,  
 val login: String,  
 val password: String,  
 val role: String  
)

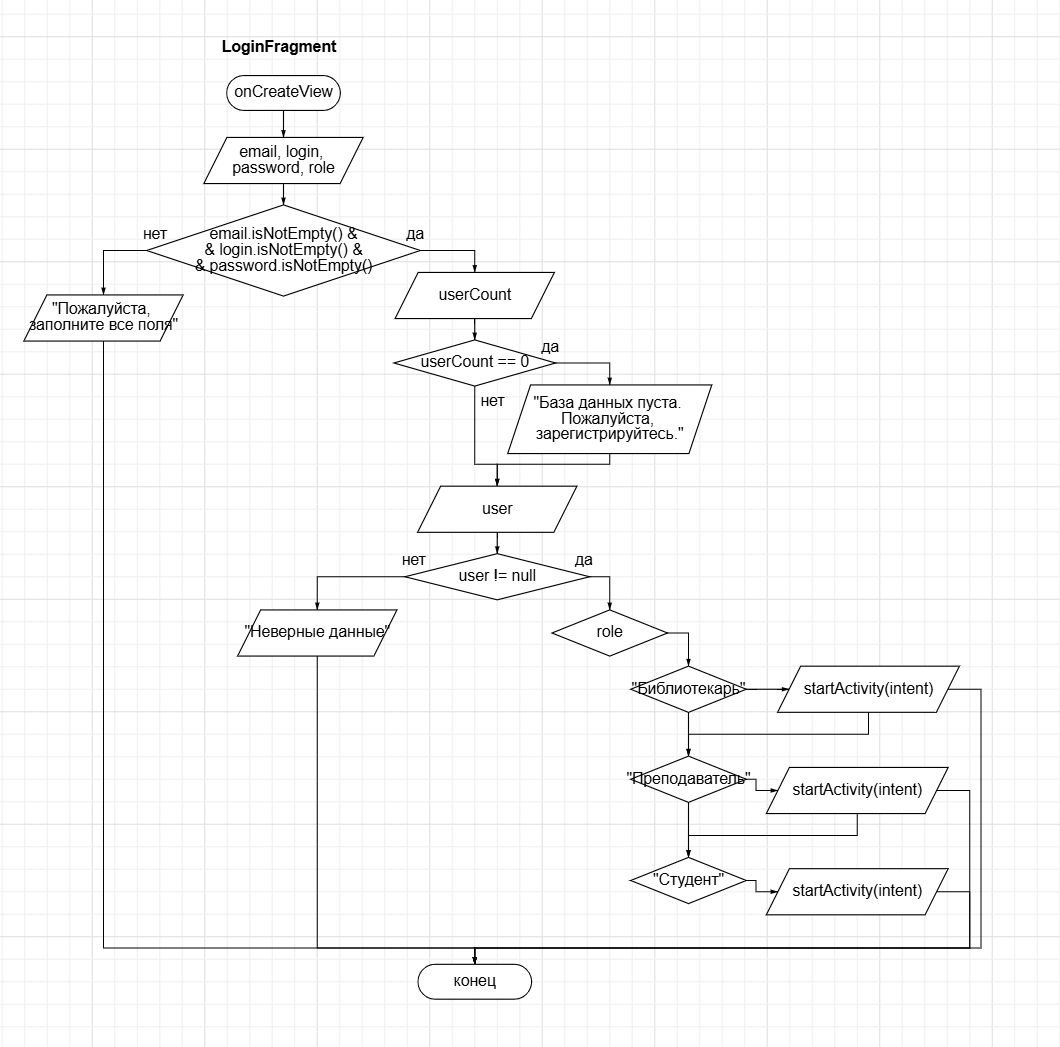
**UserDao:**

@Dao  
interface UserDao {  
 @Insert  
 suspend fun insert(user: User)  
  
 @Query("SELECT \* FROM User WHERE email = :email AND login = :login AND password = :password AND role = :role LIMIT 1")  
 suspend fun getUser(email: String, login: String, password: String, role: String): User?  
  
 @Query("SELECT COUNT(\*) FROM User")  
 suspend fun getUserCount(): Int // Method to count users  
 @Query("SELECT \* FROM User WHERE email = :email OR login = :login LIMIT 1")  
 suspend fun getUserByEmailOrLogin(email: String, login: String): User?  
}

**Блок схемы:**

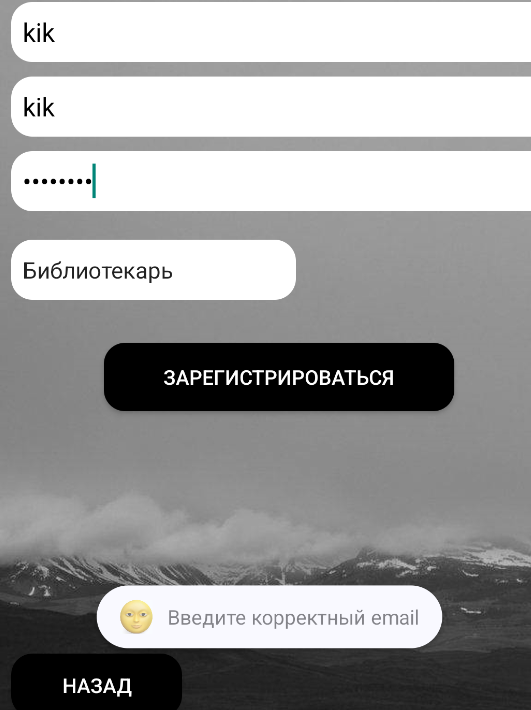
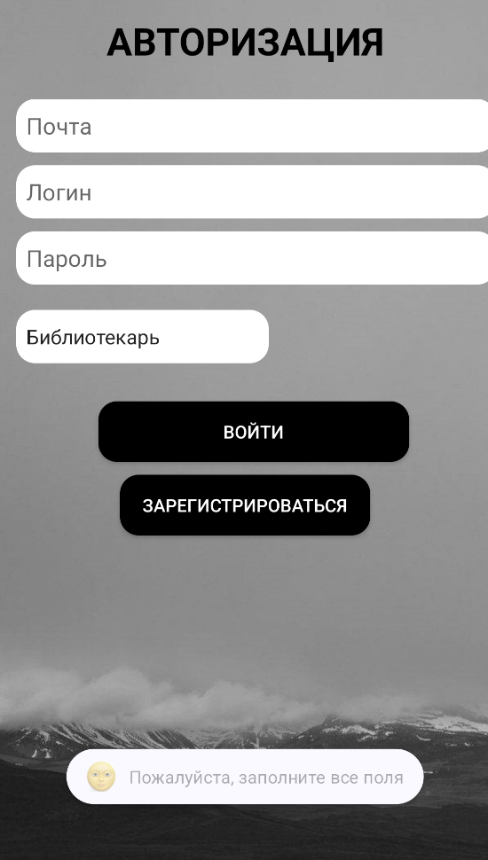
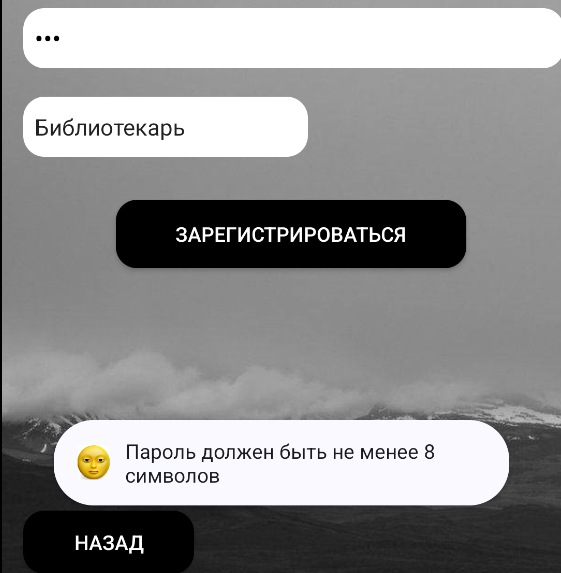
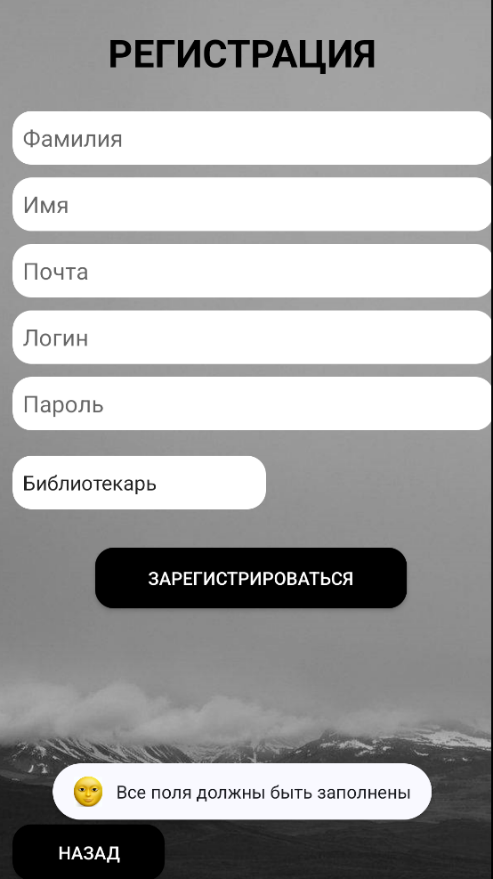
****

****

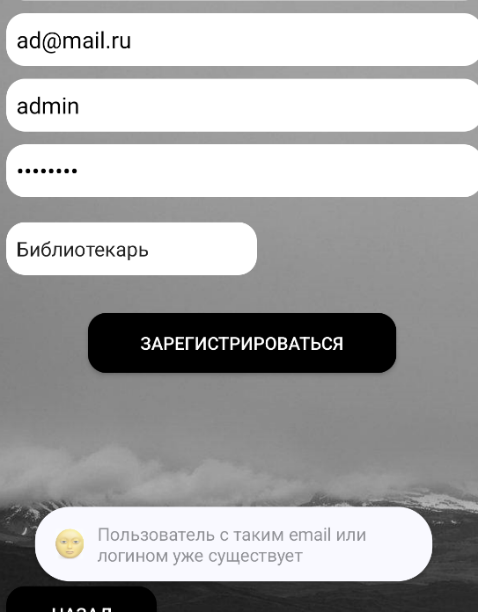
****

**Интерфейс и тестовые ситуации:**

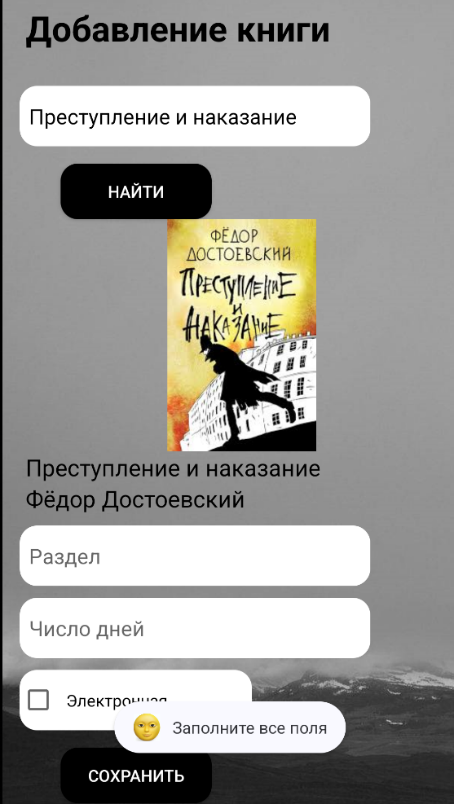
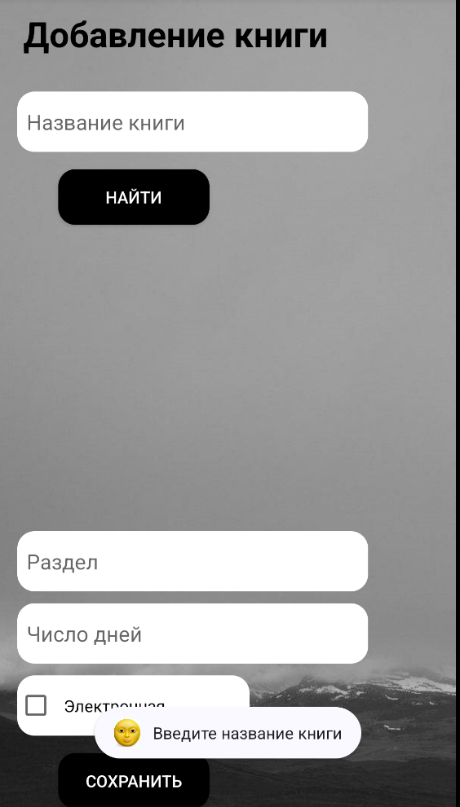
При регистрации и входе в аккаунт все поля должны быть не пустые, а также пароль должен быть не менее 8 символов, а почта в формате “...@mail.ru”

****

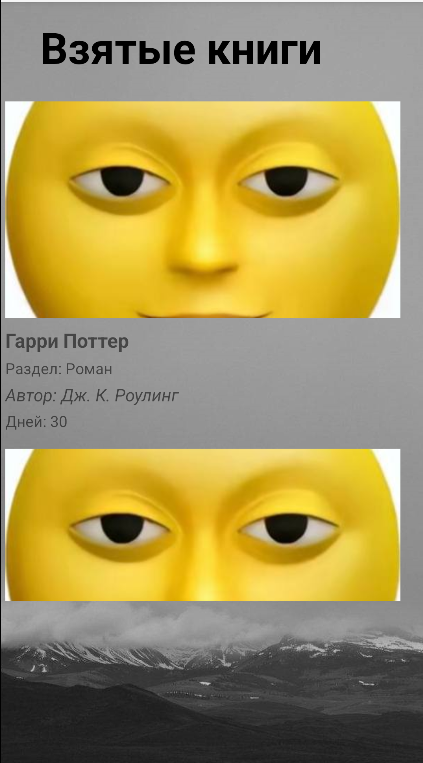
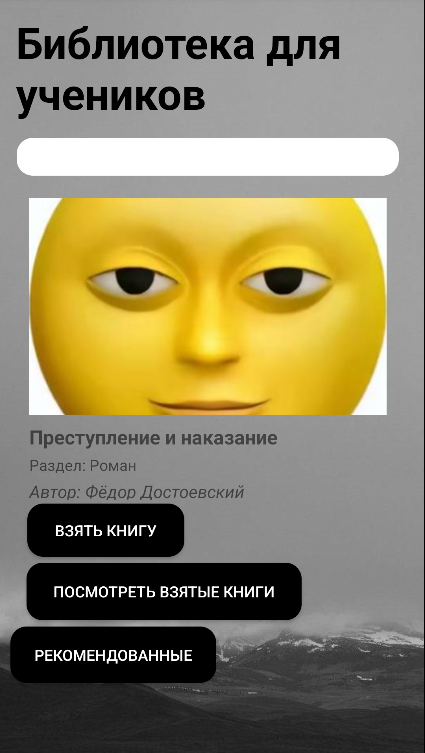
При регистрации есть проверка на то нет ли уже пользователя с такими же данными в базе данных:



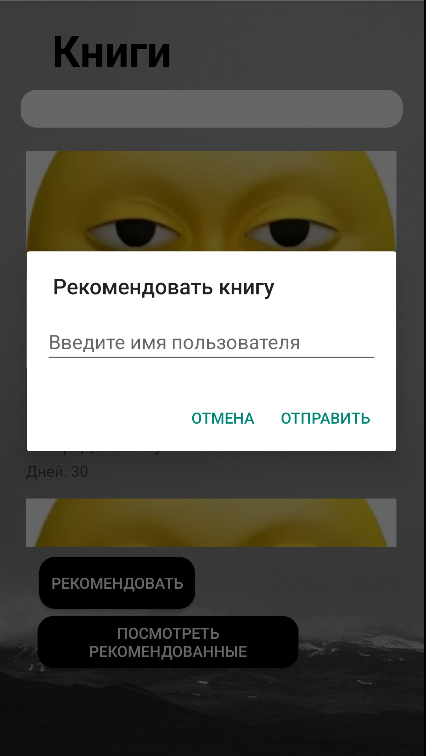
При добавлении книги поля должны быть не пустые:



Интерфейс для учеников:

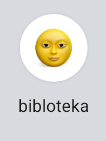


Интерфейс для преподвателей:



Чтобы порекомендовать, выбрать или удалить книгу ее сначала надо выбрать:



**Иконка** 

**Используемые библиотеки:**

|  |  |
| --- | --- |
| android.annotation.SuppressLint | Эта аннотация используется для подавления предупреждений компилятора о потенциальных проблемах в коде |
| android.graphics.Color | Этот класс предоставляет константы и методы для работы с цветами |
| androidx.appcompat.app.AppCompatActivity | AppCompatActivity — это базовый класс для активностей, который обеспечивает поддержку старых версий Android и позволяет использовать компоненты Material Design |
| android.os.Bundle | Класс Bundle используется для передачи данных между активностями и сохранения состояния активности |
| android.widget.GridLayout | GridLayout — это контейнер, который позволяет размещать дочерние элементы в виде сетки |
| android.widget.ImageView | Этот класс представляет элемент пользовательского интерфейса, который отображает изображения. |
| android.widget.LinearLayout | LinearLayout — это контейнер, который размещает дочерние элементы в одном направлении (горизонтально или вертикально) |
| android.widget.TextView | Этот класс представляет собой элемент пользовательского интерфейса для отображения текста |
| com.google.gson.Gson | Gson — это библиотека для работы с JSON в Java и Kotlin. Она позволяет легко сериализовать и десериализовать объекты, преобразуя их в JSON и обратно |
| com.squareup.picasso.Picasso | Picasso — это библиотека для загрузки и кэширования изображений для Android. Она упрощает процесс загрузки изображений из интернета в ImageView |
| okhttp3 | OkHttp — это мощный HTTP-клиент для приложений Android и Java. Он используется для отправки и получения сетевых запросов на основе HTTP |

**Используемые инструменты:**

Android Stusio, Kotlin