Министерство науки и высшего образования Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет**

**Имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИВлГУ)**

Факультет ИТР

Кафедра ПИн

КУРСОВАЯ

РАБОТА

По Разработка приложений для мобильных операционных систем

Тема Информационная система издательства

Руководитель

. . Колпаков А.А. .

(оценка) (фамилия, инициалы)

. .

(подпись) (дата)

Члены комиссии Студент ПИн-121

(группа)

. . Карпов Д.В.

(подпись) (Ф.И.О.) (фамилия, инициалы)

. . .

( подпись) (Ф.И.О.) (подпись) (дата)

Муром 2024

В этой курсовой работе была создана информационная система для издательства. В процессе работы проведен анализ предметной области и определены требования к мобильному приложению. Приложение было разработано для операционной системы Android с использованием языка программирования Kotlin в среде Android Studio.

This coursework involves the development of an information system for a publishing house. Throughout the project, the subject area was analyzed, and the requirements for the mobile application were identified. The application was created for the Android operating system using the Kotlin programming language within the Android Studio environment.

Содержание

[Введение 6](#_Toc154279033)

[1 Анализ технического задания 7](#_Toc154279034)

[2 Разработка алгоритмов 8](#_Toc154279035)

[3. Руководство программиста 13](#_Toc154279036)

[4 Руководство пользователя 20](#_Toc154279037)

[Заключение 25](#_Toc154279038)

[Список используемой литературы 26](#_Toc154279039)

[Приложение 1. Текст программы 27](#_Toc154279040)

# Введение

В настоящее время в большинстве областей профессиональной деятельности практически не осталось ни одной компании, не использующей современные автоматизированные технологии. В связи с этим перед руководителями организаций стоит задача внедрения актуальных информационных решений. В частности, речь идет о внедрении автоматизированных систем для сбора, обработки и хранения информации, которые помогут облегчить труд сотрудников и взаимодействие с клиентами. Одним из таких направлений является предоставление услуг по изданию различной печатной продукции.

С помощью информационной системы издательство сможет оптимизировать и улучшить множество рабочих процессов: формирование заказов, их выполнение и сдача печатной продукции. Кроме того, информационная система обеспечит хранение необходимых данных, их поиск, автоматический расчет стоимости производства продукции и выполнения заказов, а также предоставит отчеты, которые станут основой для принятия дальнейших управленческих решений. Важным аспектом успешной работы издательства являются скорость и мобильность. Для повышения этих показателей используются информационные системы в виде мобильного приложения. Для его использования пользователю достаточно иметь мобильное устройство и доступ в интернет.

Таким образом, целью курсовой работы является создание информационной системы в виде мобильного приложения, автоматизирующего работу издательства.

Для выполнения поставленной цели необходимо выполнить задачи:

1) изучить структуру работы издательства, выявить пользователей системы;

2) разработать модели данных, которые опишут структуру приложения и его логику;

3) на основе моделей данных создать мобильное приложение;

4) протестировать созданное мобильное приложение.

# 1 Анализ технического задания

1.1 Описание предметной области

Издательство – это организация, специализирующаяся на выпуске разнообразной печатной продукции. Оно выполняет заказы в соответствии с требованиями клиента, которым может быть как частное лицо, так и юридическое лицо. Печать заказов осуществляется типографией.

Заказ может включать одну или несколько позиций печатной продукции, таких как книги, брошюры, рекламные буклеты, бюллетени для голосования, газеты и другие виды материалов. Стоимость печатной продукции определяется её типом, наценкой издательства, а также видом используемого материала, его цветом и размером. Для производства печатной продукции применяются различные виды материалов, включая: бумагу для глубокой печати, газетную бумагу, глянцевую бумагу, дизайнерскую бумагу, картон, крафт-бумагу, матовую бумагу, мелованную бумагу, обложечную бумагу, офсетную бумагу, самокопирующуюся бумагу, типографскую бумагу и форзацную бумагу.

Над выполнением заказа трудится один или несколько сотрудников издательства, каждый из которых выполняет свою специфическую роль. Ниже приведены некоторые должности сотрудников и их обязанности: бильдредактор и фоторедактор – ответственные за поиск и выбор фотоиллюстраций; верстальщик – занимается подгонкой шрифтов для печати, исправлением текстовых ошибок и форматированием иллюстраций; корректор – специалист, который нормализует грамматику (орфографические и пунктуационные ошибки) и типографику; рерайтер – эксперт, перерабатывающий исходный текст и создающий универсальную статью по правилам поисковых систем с сохранением смысла, но написанную другими словами; художник-иллюстратор – человек, создающий смысловые изображения для оформления книг, журналов и других печатных материалов.

1.2 Функциональные возможности приложения

Цель данной работы заключается в разработке информационной системы для издательства в формате мобильного приложения, предназначенного для двух категорий пользователей: клиентов и администраторов. Разработанное приложение будет обеспечивать автоматизацию процессов формирования и мониторинга данных, а также генерации отчетности. Это позволит издательствам повысить эффективность своей работы, оптимизировать взаимодействие с клиентами и привлечь новых пользователей.

На основании произведённого анализа предметной области и аналогов были сформированы требования к разрабатываемому приложению:

- Реализовать регистрацию и авторизацию пользователей;

- Предоставлять функционал приложения в зависимости от роли пользователя;

- Предоставить работу с несколькими типами продукции;

- Предоставить клиенту возможность самому создать желаемую продукцию;

- Реализовать валидацию данных;

- Реализовать автоматический расчёт стоимости продукции и заказа;

- Реализовать поиск данных по определенному критерию;

- Реализовать формирование отчётов;

- Реализовать отправку уведомлений и отчётов пользователям в виде e-mail писем.

2 Разработка алгоритмов

Перед тем, как приступить к разработке, необходимо создать модели, описывающие структуру базы данных и их содержимое, а также создать блок-схемы, описывающие работу с данными. Этот этап является самым важным при создании информационной системы.

2.1 Концептуальная модель данных

При проектировании базы данных необходимо построить модель данных, основанную на предметной области. Для отображения множества понятий и связей между ними, определяющих смысловую структуру предметной области или её конкретного объекта, используется концептуальная модель данных.

Концептуальная модель обычно отображается в виде ER-диаграммы. ER-диаграмма – это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы (или ER-модели) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи – глаголов.

Под сущностью в концептуальной модели данных понимают объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении. Каждая сущность имеет свой набор атрибутов. Атрибуты представляют собой свойства, характеризующие сущность.

Между сущностями существует несколько видов связи, а именно:

- Связь один-к-одному. В этой связи объекту одной сущности можно сопоставить только один объект другой сущности;

- Связь один-ко-многим. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице;

- Связь многие-ко-многим. При этом типе связей одна строка из таблицы А может быть связана с множеством строк из таблицы В. В свою очередь одна строка из таблицы В может быть связана с множеством строк из таблицы А.

После анализа предметной области были выделены следующие сущности: заказ, типография, сотрудник, продукция, тип печатной продукции, пользователь, материал. Cвязи между сущностями и их атрибуты отображены в приложении 3.

2.2 Логическая модель данных

**Логическая модель данных** — это модель данных конкретной предметной области, выраженная независимо от конкретного продукта управления базами данных или технологии хранения. Логические модели данных чаще всего используются в бизнес-процессах, которые стремятся охватить важные для организации вещи и то, как они связаны друг с другом. После проверки и утверждения логическая модель данных может стать основой физической модели данных и сформировать дизайн базы данных.

Логические модели данных основываются на структурах, определенных в предыдущей концептуальной модели данных, поскольку это описывает семантику информационного контекста, которую также должна отражать логическая модель. Тем не менее, поскольку логическая модель данных предполагает реализацию в конкретной вычислительной системе, содержимое логической модели данных корректируется для достижения определенной эффективности.

В приложении 3 представлена логическая модель данных. В сущности «Типография» хранятся данные о типографиях, с которыми сотрудничает издательство. В сущности «Сотрудник» хранятся данные о сотрудниках издательства. В сущности «Заказ» хранятся данные о заказе и уникальный номер типографии, которая выполняет/выполнила заказ. Сущность «Заказ\_Сотрудник» указывает над какими заказами работали/работают сотрудники. Сущность «Пользователь» хранит данные о пользователях. Сущность «Материал» хранит данные о материалах, используемых для создания печатных продукций. Сущность «Печатная\_продукция\_Материал» указывает на количество используемых материалов в продукциях. Сущность «Тип\_печатной\_продукции» хранит данные о наценках типов печатной продукции.

Сущность «Печатная\_продукция» хранит данные о продукции, уникальный номер типа печатной продукции и уникальный номер заказчика(пользователя) этой продукции. Сущность «Фото\_печатной\_продукции» хранит пути до макетов печатных продукций. Сущность «Печатная\_продукция\_Заказ» указывает на количество печатных продукций в заказах.

2.3 Физическая модель данных

На основе логической модели данных строится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных.

Для информационной системы издательства была выбрана СУБД PostgreSQL. Данная СУБД предоставляет возможность хранить различные типы данных. В нашем случае будут использоваться следующие типы: тип данных bigint для хранения id, integer для хранения целочисленных данных, тип данных double для хранения нецелочисленных данных, тип данных numeric для хранения стоимости, тип данных date для хранения дат, тип данных varchar для хранения строковых данных. Физическая модель данных представлена в приложении 3.

2.4 Описание основных алгоритмов

Разрабатываемый продукт представляет собой многопользовательское приложение с несколькими видами пользователей. Из-за этого особое внимание было выделено процессам регистрации и авторизации, так как при их некорректной работе пользователь не сможет работать с основным функционалом приложения. Стоит отменить, что работа с базой данных происходит на сервере, а мобильное приложение отправляет и принимает данные в виде JSON. Блок-схемы, описывающие алгоритмы регистрации и авторизации, представлены в приложении 4.

Стоит отметить, что алгоритмы схожи. Первое различие заключается в том, что при авторизации происходит проверка на наличие данных о пользователе в памяти устройства. Второе различие: после получения ответа от сервера происходит проверка кода ответа сервера на равенство 200, а не на 400, как при регистрации. Третье различие: в после успешной регистрации открывается экран с авторизацией, а после успешной авторизации – главный экран приложения.

Основная задача информационной системы – работа с данными. Пользователи будут добавлять, просматривать, изменять и удалять данные. Поэтому в приложении 4 представлены блок-схемы, описывающие алгоритмы работы с данными.

Стоит отметить, что добавление и изменение имеет одинаковую логику. Разница лишь в том, что при добавлении запись не имеет уникального номера, а при изменении имеет. При успешном удалении данных происходит отображение сообщения об успехе и переход на экран со списком данных. Такой же переход происходит при добавлении и изменении данных. Алгоритм получения данных можно описать одним предложением: при успешном получении данных от сервера данные выведутся на экран, при неудаче – экран будет пустым.

# 3. Руководство программиста

Приложение создавалось на языке Kotlin версии 1.9.0 и на основе макета Navigation Drawer Views Activity. Помимо библиотек, встроенных при создании проекта были добавлены следующие библиотеки:

Для работы с http и корутинами:

- implementation ("com.squareup.okhttp3:okhttp:4.9.0");

- implementation ("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2");

- implementation ("com.google.code.gson:gson:2.8.5").

Для работы с пагинацией:

implementation ("androidx.paging:paging-runtime-ktx:3.2.1");

implementation ("androidx.swiperefreshlayout:swiperefreshlayout:1.1.0");

implementation ("com.google.dagger:hilt-android:2.39.1").

3.1 Модели

Классы моделей содержат в себе поля, которые являются полями в одноименных таблицах базы данных. Они служат для того, чтобы правильно передавать и получать данные с сервера. В таблице 1 представлены все модели программы.

Таблица 1 – модели приложения

|  |  |
| --- | --- |
| Модель | Пояснение |
| data class Employee(  val id: Long,  var surname: String,  var name: String,  var patronymic:String,  var phone:String,  var email:String,  var post: String,  var pathToImage: String,  var birthday: LocalDate )  data class Material(  val id: Long,  var type: String,  var color: String,  var size: String,  var cost: BigDecimal ) | Модель «Сотрудник». Содержит в себе идентификатор, фамилию, имя, отчество (при наличии), номер телефона, электронную почту, должность, путь до фотографии на сервере и дату рождения.  Модель «Материал». Содержит в себе идентификатор, тип, цвет в формате R;G;B, размер и стоимость. |
| data class PrintingHouse(  val id: Long,  var name: String,  var phone: String,  var email: String,  var state: String,  var city: String,  var street: String,  var house: String ) | Модель «Типография». Содержит в себе идентификатор, название, номер телефона, электронную почту, субъект РФ, город, улицу и номер дома. |
| data class TypeProduct(  val id: Long,  var type: String,  var margin: Double ) | Модель «Тип продукции». Содержит в себе идентификатор, название и наценку в %. |
| data class User(  val id: Long,  val name:String,  val phone: String,  val email: String,  val role: String  ) | Модель «Пользователь». Содержит в себе идентификатор, наименование, номер телефона, электронную почту и роль. |

Помимо моделей используются DTO-модели. Они также используются для приёма и отправки данных на сервер, за исключением того, что они либо содержат дополнительные данные, отсутствующие в простых моделях, либо в них отсутствуют некоторые поля.

Таблица 2 -DTO-модели

|  |  |
| --- | --- |
| Модель | Пояснение |
| data class BookingSimpleAcceptDTO(  val id: Long,  val cost: BigDecimal ) | DTO-модель, хранящая в себе неполные данные о заказе: идентификатор и стоимость. |
| data class BookingAcceptDTO(  val id: Long,  val status: String,  val startExecution: LocalDate,  val endExecution: LocalDate?,  val cost: BigDecimal,  val printingHouse: PrintingHouse?,  val products: List<ProductSimpleAcceptDTO> ?,  val employees: List<EmployeeDTO>? ) | DTO-модель, хранящая в себе полную информацию о заказе и полученная с сервера: идентификатор, статус, даты приёма и выполнения, стоимость, типографию, список продукции и список сотрудников. |
| data class BookingSendDTO(  var id: Long,  var status: String,  var startExecution: LocalDate,  var endExecution: LocalDate?,  var cost: BigDecimal,  var printingHouse: PrintingHouse?,  var productsWithMargin: List<ProductWithEditionDTO>?,  var idsOfEmployees: List<Long>? ) | DTO-модель, хранящая в себе полную информацию о заказе. Экземпляр данной модели отправляется на сервер. Данные в модели: идентификатор, статус, даты приёма и выполнения, стоимость, типография, список продукций с их количеством и список идентификаторов сотрудников. |
| data class CountProductsInBookingDTO(  val booking: BookingSimpleAcceptDTO,  val edition: Int ) | DTO-модель, хранящая в себе данные о количестве продукции в заказе. Данные: заказ и количество продукции в заказе. |
| data class EmployeeDTO(  val id: Long,  var surname: String,  var name: String ,  var patronymic:String,  var phone:String,  var email:String,  var post: String,  var pathToImage: String,  var photo: String,  var birthday: LocalDate, ) | DTO-модель, получаемая с сервера. Имеет те же поля, что и обычная модель + фотография в виде строки base64. |
| data class ProductAcceptDTO(  val id: Long,  val name: String,  val username: String,  val userEmail: String,  val cost: BigDecimal,  val typeProduct: TypeProduct?,  val productMaterialDTOS: List<ProductMaterialDTO>?,  val countProductsInBookingDTOS: List<CountProductsInBookingDTO>?,  val photos: List<String>? ) | DTO-модель, представляющая собой объект принимаемой с сервера продукции. Данные: идентификатор, название, имя заказчика продукции, его электронная почта, стоимость, тип продукции, список материалов и их количества, список заказов и количества продукций в них, список фотографий. |
| data class ProductMaterialDTO(  val material: Material,  val countMaterials: Int ) | DTO-модель, хранящая в себе данные о материале и его количестве в продукции. Данные: материал и его количество. |
| data class ProductSendDTO(  var id: Long,  var userId: Long,  var name: String,  var cost: BigDecimal,  var typeProduct: TypeProduct?,  var productMaterialDTOS: List<ProductMaterialDTO>?, ) | DTO-модель, представляющий собой объект отправляемой на сервер продукции. Данные: идентификатор, идентификатор заказчика, название, стоимость, тип продукции, список материалов с их количеством. |
| data class ProductSimpleAcceptDTO(  val id: Long,  val name: String,  val username: String,  val edition: Int,  val photo: String,  val cost: BigDecimal ) | DTO-модель, хранящая неполные данные о продукции. Данные: идентификатор, название, наименование пользователя, количество продукции в заказе, фото и стоимость. |
| data class ProductWithEditionDTO(  var id: Long,  var name: String?,  var cost: BigDecimal?,  var edition: Int ) | DTO-модель, хранящая неполные данные о продукции. Данные: идентификатор, название, стоимость и количество продукции в заказе. |
| data class UserAcceptDTO(  val id: Long,  val name: String,  val email: String,  val phone: String,  val products: List<ProductSimpleAcceptDTO>?,  val bookings: List<BookingSimpleAcceptDTO>? ) | DTO-модель с полными данными о пользователе-заказчике. Данные: идентификатор, наименование, электронная почта, номер телефона, список продукции и список заказов. |

3.2 Репозитории

Следующим набором классов являются репозитории. Их задача – это отправка и приём данных с сервера. Для каждой модели, хранящейся в базе данных, был создан свой репозиторий: BookingRepository, EmployeeRepository, MaterialRepository,PrintingHouseRepository,ProductRepository,TypeProductRepository, UserRepository. Структуры репозиториев схожи, поэтому они были обобщены и сведены в одну таблицу 3.

Таблица 3 - структура репозитория

|  |  |
| --- | --- |
| Поля и методы | Пояснение |
| private val client = OkHttpClient.Builder()  .addInterceptor(JwtInterceptor()).build() | Объект класса OkHttpClient(). Выполняет запрос на сервер и возвращает ответ. |
| private val gson = GsonBuilder().  registerTypeAdapter(LocalDate::class.java, LocalDateAdapter()).create() | Объект класса Gson. Сериализует объект класса в Json или десериализует объект или список объектов из Json. registerTypeAdapter используется только для тех моделей, в которых есть поле с типом данных. |
| suspend fun add([объект модели]): MessageResponse? или  suspend fun add([объект модели]): Int или  suspend fun add([объект модели], [фотографии])MessageResponse? | Метод добавления данных в базу данных. Принимает объект модели или объект модели и фотографии(для сотрудников и продукций). Возвращает либо объект MessageResponse (возможен null), либо код состояния HTTP. |
| suspend fun get(page: Int?, [параметр поиска]: String): [Список моделей]? или  suspend fun get(page: Int, [параметр поиска]: Long?, [параметр поиска]: String): [Список моделей]? | Метод получения списка моделей. Принимает номер страницы для пагинации и параметры поиска, которые могут быть пусты. В случае успеха возвращает список моделей, иначе null. |
| suspend fun get([идентификатор]: Long):[Модель]? | Метод получения модели. Принимает идентификатор модели, возвращает модель в случае успеха, иначе null. Данный метод имеют 3 репозитория: BookingRepository, ProductRepository, UserRepository. |
| fun getPaged[Название модели]([параметр поиска]: String) = Pager или  fun getPaged[Название модели]([параметр поиска]: Long? [параметр поиска]: String) = Pager или | Метод получения данных,прошедших пагинацию, в виде livedata. Принимает параметры поиска, которые могут быть пустыми или null. |
| suspend fun update([объект модели]): MessageResponse? или  suspend fun update([объект модели]): Int или  suspend fun update([объект модели], [фотографии])MessageResponse? | Метод изменения данных. Принимает объект модели или объект модели и фотографии (для сотрудников и продукций). Возвращает либо объект MessageResponse (возможен null), либо код состояния HTTP. |
| suspend fun delete([идентификатор]:Long): MessageResponse? | Метод удаления данных. Принимает идентификатор записи, возвращает объект Message Response в случае успеха, иначе null. |

3.3 DataSources

Для каждой основной сущности базы данных был создан DataSource – класс, представляющий источник данных для загрузки списка с использованием пагинации. Объявление класса: class [Название модели]DataSource([параметр поиска]: String): PagingSource<Int, [Название модели]>. Каждый DataSource класс имеет два метода. Данные методы описаны в таблице 4.

Таблица 4 методы класса DataSource

|  |  |
| --- | --- |
| Метод | Пояснение |
| override fun getRefreshKey(state: PagingState<Int, [Название модели]>): Int? | Метод получения ключа обновления. Принимает состояние текущей загрузки данных, возвращает ключ. |
| override suspend fun load(params: LoadParams<Int>): LoadResult<Int, [Название модели]> | Метод загрузки данных в асинхронном режиме с использованием пагинации. Возвращает объект LoadResult. |

3.4 Adapters

Классы адаптеры нужны для отображения данных в компонент RecyclerView или в Spinner. Адаптеры делятся на три типа:

1) Адаптеры, наследующиеся от ArrayAdapter, используются для работы с компонентом Spinner. Объявление класса: class [Название модели] SpinnerAdapter (context: Context, [переменная списка]:List<[Модель]>): ArrayAdapter<[Модель]> (context, R.layout.simple\_spinner\_dropdown\_item, [переменная списка]).

2) Адаптеры, наследующиеся от RecyclerView.Adapter. Объявление класса: class [Название модели]Adapter(private val [переменная списка]: List<[Модель]>): RecyclerView.Adapter<[Название адаптера].ViewHolder>().

3) Адаптеры, наследующиеся от PagingDataAdapter. Объявление класса: class [Название модели]Adapter (private val clickListener: OnItemClickListener): PagingDataAdapter<[Модель],[Название адаптера].Holder>(COMPARATOR).

Методы адаптеров, относящихся к первому типу, отображены в таблице 5.

Таблица 5 – методы адаптеров первого типа

|  |  |
| --- | --- |
| Метод | Пояснение |
| override fun getView(position: Int, convertView: View?, parent:ViewGroup): View | Метод получения отображаемого представления элемента в Spinner. Принимает позицию элемента в spinner, представление элемента, родительский контейнер, к которому привязан элемент. Возвращает представление элемента. |
| override fun getDropDownView(position: Int, convertView: View?, parent: ViewGroup): View | Метод получения представления выпадающего списка. Принимает позицию элемента в spinner, представление элемента, родительский контейнер, к которому привязан элемент. Возвращает представление. |
| fun getSelected[Модель](position: Int):[Модель]? | Метод получения выбранного объекта модели. Принимает позицию объекта, возвращает объект или null. |

У адаптеров второго и третьего типов есть встроенный класс ViewHolder (Holder). Он предоставляет элементы интерфейса, связанные с каждым элементом RecyclerView. Методы адаптеров, относящихся ко второму типу, отображены в таблице 6.

Таблица 6 - методы адаптеров второго типа

|  |  |
| --- | --- |
| Метод | Пояснение |
| override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):[Название ViewHolder].ViewHolder | Метод создания нового ViewHolder. Принимает контейнер, в котором будет размещен новый ViewHoder и тип ViewHolder, возвращает ViewHolder. |
| override fun onBindViewHolder(holder: RecyclerViewMaterialsAddAdapter.ViewHolder, position: Int) | Метод, связывающий данные из списка с элементом ViewHolder. Принимает ViewHolder и индекс элемента в списке. |
| override fun getItemCount(): Int | Метод, возвращающий количество элементов в списке. |
| override fun getItemId(position: Int): Long | Метод, возвращающий уникальный идентификатор элемента в указанной позиции |
| fun get[Модель](position: Int):[Модель] | Метод, возвращающий модель. Принимает позицию модели в RecyclerView. |

Адаптеры, относящиеся к 3 типу, не имеют методов getItemCount() и getItemId(position: Int) адаптеров 2 типа, но имеют реализацию метода onClick(v: View?) интерфейса OnItemClickListener и статический компаратор для сравнения элементов RecyclerView.

3.5 ViewModel и Fragment

Для каждой основной модели (Booking, User, Employee, Material, PrintingHouse, Product, TypeProduct) были созданы одноименные ViewModel и Fragment. Задача ViewModel – это управление списком данных о моделях, прошедших пагинацию, в рамках фрагмента. У каждой ViewModel есть метод updateSearchType(query: String). Его задача – обновить параметр поиска данных.

Fragment – клаcc-прослойка между Activity и визуальными составляющими программы. С одной стороны, это контейнер для любых View-объектов, которые могут быть показаны пользователю. С другой — продолжение Activity, от которого Fragment получает всю информацию об изменениях в жизненном цикле.

Каждый фрагмент в приложении имеет binding, связывающий фрагмент и xml-разметку, связанную с ним ViewModel, а также Adapter для RecyclerView.

Помимо описанных полей каждый фрагмент имеет 3 метода, описанные в таблице 7.

Таблица 7 – методы фрагмента

|  |  |
| --- | --- |
| Метод | Описание |
| override fun onDestroyView() | Метод, вызываемый при уничтожении представления фрагмента. |
| override fun onCreateView(  inflater: LayoutInflater,  container: ViewGroup?,  savedInstanceState: Bundle? ): View | Метод, вызываемый при создании и отображении макета фрагмента. Принимает объект LayoutInflater, который может быть использован для создания представления фрагмента, корневое представление, состояние фрагмента, сохраненное в последнем вызове onSaveInstanceState(). |
| override fun onItemClick(position: Int) | Открывает экран с детальной информацией о выбранном объекте. Принимает номер объекта в RecyclerView. |

3.6 Activity

Activity – это компонент приложения, который является одним из его фундаментальных строительных блоков. Его основное предназначение заключается в том, что оно служит точкой входа для взаимодействия приложения с пользователем, а также отвечает за то, как пользователь перемещается внутри приложения.

В приложении Activity делятся на 2 типа: DetailsActivity для отображения информации и Activity, в которых обрабатывается пользовательский ввод. В каждой Activity есть переменная binding, связывающая Activity и xml-разметку, используется экземпляр класса Messages для отображения сообщений пользователям.

В таблице 8 описаны методы, присутствующие в DetaisActivity и Activity, обрабатывающий пользовательский ввод.

Таблица 8 -общие методы Activity

|  |  |
| --- | --- |
| Метод | Пояснение |
| override fun onCreate(savedInstanceState: Bundle?) | Метод, вызываемый при создании Activity. |
| override fun onSupportNavigateUp(): Boolean | Метод перехода в прошлую активность. Возвращает true, если нужно завершить текущую активность. |
| private fun setStartData() | Метод заполнения компонентов данными о модели. |

Методы в Activity, отвечающие за обработку пользовательского ввода имеют следующие методы: setListeners() – метод добавляющий прослушивателей пользовательского метода, save() (register(), login(), generateReport()) проверяет корректность пользовательского ввода, в случае успеха выполняет добавление, изменение данных (save), авторизацию(login), регистрацию (register) и генерацию отчета (generateReport). Также некоторые Activity имеют вспомогательные методы для обработки выбора фотографий(openGallery(), waitingPhoto()) и ввода даты(setBirthday()).

3.7 Dialogs

Классы, заканчивающиеся на Dialog, отвечают за создание и отображение диалогового окна, в котором подтверждается выполнение действия. В основном, от пользователя требуется подтверждение удаления данных или выполнения заказа. Для создания диалогового окна класс Dialog наследуется от DialogFragment() и в нём переопределяется метод onCreateDialog( savedInstanceState: Bundle?), который возвращает диалоговое окно, в котором обрабатываются нажатия на negativeButton(отмена) и positiveButton(подтвердить).

# Заключение

В ходе выполнения курсовой работы было разработано Android-приложение на языке программирования Kotlin, с помощью среды разработки Android Studio, предназначенное для автоматизации работы издательства. Приложение является многопользовательским с двумя видами пользователей. Приложение предоставляет функционал в зависимости от роли пользователя. Администраторы могут быстро и удобно работать с данными, непосредственно связанными с работой издательства: типографиями, сотрудниками, типами продукций, материалами, а клиенты могут сами создавать желаемую продукцию в удобном конструкторе.

Основные технологии, использованные при разработке приложения, включают в себя язык программирования Kotlin, архитектурный шаблон MVVM (Model-View-ViewModel) для построения чистой архитектуры, использование библиотек OkHttp и Gson для сериализации и десериализации объектов Kotlin в JSON и их отправки/приёма от сервера, а также различные компоненты Android, такие как Activities, Fragments, RecyclerView, Spinner и др.

Процесс разработки включал в себя проектирование пользовательского интерфейса, создание функционала для взаимодействия с сервером, обработку пользовательского ввода, а также реализацию многопоточности для корректной работы с http-запросами с помощью корутин.

В результате проделанной работы было получено полноценное Android приложение, предоставляющее удобный и функциональный интерфейс для взаимодействия с данными издательства.

# Список используемой литературы

1. Введение в разработку приложений для смартфонов на ОС Android / А.Семакова – М.: Национальный открытый Университет «ИНТУИТ», 2016

2. Колисниченко Д.Н. Программирование для Android 5. Самоучитель. — СПб.: БХВПетербург, 2015. — 303 с.

3. Дейтел П., Дейтел Х., Уолд А. Android для разработчиков. 3-е изд. — СПб.: Питер, 2016.

4. Гриффитс Дэвид, Гриффитс Дон Head First. Программирование для Android. 2-е изд. — СПб.: Питер, 2018. — 912 с

5. Head First. Kotlin. — СПб.: Питер, 2020. — 464 с.: ил. — (Серия «Head First O’Reilly»)

# Приложение 1. Текст программы

Ссылка на github репозиторий с исходным кодом приложения: https://github.com/shish1mora/Publishing-house-kotlin