|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**КУРСОВАЯ РАБОТА**

по дисциплине «Разработка программных интерфейсов»

**Тема курсовой работы «Мобильная информационная система расписания авиаперевозок»**

**Студент группы** ИКБО-11-19 Лачаев Кирилл Максимович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель курсовой работы** к.т.н. Чернов Е.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена к защите «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к защите «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Утверждаю

Заведующий кафедрой МОСИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Головин С.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2022 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

«Разработка программных интерфейсов»

Студент Лачаев Кирилл Максимович Группа ИКБО-11-19

**Тема работы:** «Мобильная информационная система расписания авиаперевозок»

**Исходные данные:** Разработанное приложение для операционной системы Android служит для предоставление информации пользователям о рейсах авиапереозчика: цена, название, описание. Аудиторией данного приложения являются все пользователи желающие воспользоваться услугами авиперевозчика. Приложение разрабатывается на языке Kotlin в интегрированной среде разработки Android Studio.

**Функционал разрабатываемой системы:** Добавление и редактирование рейсов в базе данных и отображение всех записей базы данных.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:** Анализ существующих аналогов с указание их достоинств и недостатков, описание информационных процессов предметной области, IDEF0 и DFD диаграммы, краткое описание требований к разрабатываемому прототипу, описание и обоснование выбора информационного и программного обеспечения, архитектура программной системы, подробное описание разрабатываемой системы, тестирование и верификация программного продукта.

**Срок представления к защите курсовой работы:** до «\_\_» мая 2022 г.

**Задание на курсовую работу выдал** \_\_\_\_\_\_\_\_\_\_\_\_ к.т.н. Чернов Е.А.

«\_\_» февраля 2022 г.

**Задание на курсовую работу получил**  \_\_\_\_\_\_\_\_\_\_\_\_ (Лачаев К.М.)

ВВЕДЕНИЕ

На сегодняшний день все больше людей предпочитают мобильный и быстрый доступ к информации. С каждым годом процент пользователей мобильных версий сайтов растет, а, следовательно, компьютерные версии становятся все менее популярны. Согласно информации сайта Wi-life.ru, количество различных мобильных устройств в 2015 г достигло уровня 7.7 миллиардов, что превысило всю человеческую популяцию в 7.1 миллиардов. Статистика сайта Cossa.ru говорит о том, что мобильным телефоном в России владеет 98 процентов населения. Следовательно, информацию стоит адаптировать именно под смартфоны, чем под персональные компьютеры. Существующие сервисы, предоставляющие информацию о рейсах, данное приложение будет работать схожим образом, предоставляя пользователям возможность получить краткую информацию к рейсам.

Данная программная система создается с целью обеспечения пользователей краткой информацией о рейсах, а администраторов - управлением каждой записью. Таким образом, можно выделить следующие задачи: обеспечить наполнение базы данных актуальными интересными рейсами, обеспечить регистрацию по ролям, и полный инструментарий админам для наполнения базы данных.

Объектом исследования является разрабатываемое приложение «Мобильная информационная система расписания авиаперевозок» и уже существующие аналоги. В качестве предмета исследования выступают методы по наилучшему пути предоставления информации пользователю.

Для теоретической части курсовой работы использовался метод анализа для лучшего понимания изучаемой сферы и процесса разработки приложения. Для практической части использовался метод наблюдений и экспериментов за оптимальностью выбора предложений в разных условиях. Информационная база исследования включает статистику маркетплейсов и потребности современных пользователей в мобильных приложениях.

Основная часть подразделяется на 3 раздела: исследовательский, проектный и технологический. В свою очередь каждый из разделов подразделяется на подразделы: исследовательский на анализ существующих аналогов и описание информационных процессов предметной области, проектный на краткое описание требований, описание и обоснование выбора ПО, и архитектуру ПС, технологический на подробное описание разрабатываемой системы и тестирование продукта.

ОСНОВНАЯ ЧАСТЬ

1. Исследовательская часть
   1. Анализ существующих аналогов с указанием их достоинств и недостатков

Проанализировав несколько приложений с схожим функционалом были выявлены следующие аналоги (Таблица 1).

«Таблица 1 — Плюсы и минусы аналогов»

|  |  |  |
| --- | --- | --- |
| Название аналога | Плюсы | Минусы |
| Авиасейлс | Большой выбор | Комиссионный сбор |
| uralairlines | Удобство использования | Только рейсы uralairlines |

Таким образом можно сделать вывод, что у современных аналогов распространенными недостатками являются: небольшое количество рейсов и скудное неполное описание.

* 1. Описание информационных процессов предметной области IDEF0 и DFD диаграммы

Диаграмма, отражающая структуру и функции системы (Рисунок 1) содержит 4 блока: «Зарегистрировать администратора, авторизоваться как администратор», «Добавление, редактирование, удаление рейса», «Отображение рейсов».

Наполнением расписания занимаются администраторы, но прежде, чем стать админом, необходимо пройти регистрацию. После авторизации как админ, он может добавлять, удалять и редактировать рейсы.

Пользователю входить не надо, базовый функционал приложения доступен сразу без регистрации. Он может просмотреть список всех рейсов и выбрать понравившейся.

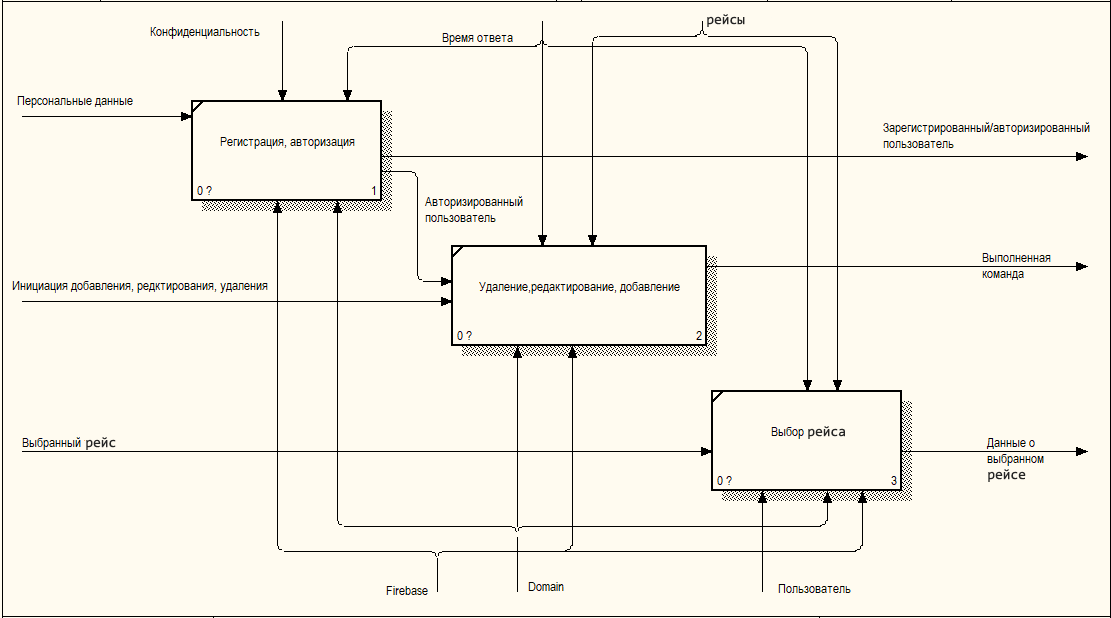


Рисунок 1 ― Диаграмма Idef0

На диаграмме информационных потоков (Рисунок 2) можно увидеть, как система взаимодействует с внешними сущностями и обрабатываем информацию.

Админ вводит персональные данные или логин и пароль. Во время регистрации формируется запрос на добавление его в базу данных, при входе, данные запрашиваются из базы данных и если они совпадают, то админ получает авторизированный доступ. Он может добавлять, редактировать и удалять данные, после чего формируются характерные запросы и происходит исполнение на стороне базы данных.

Пользователю же ничего проходить не надо, ему отображаются все рейсы из базы данных и по выбору предоставляется полная информация о рейсе.

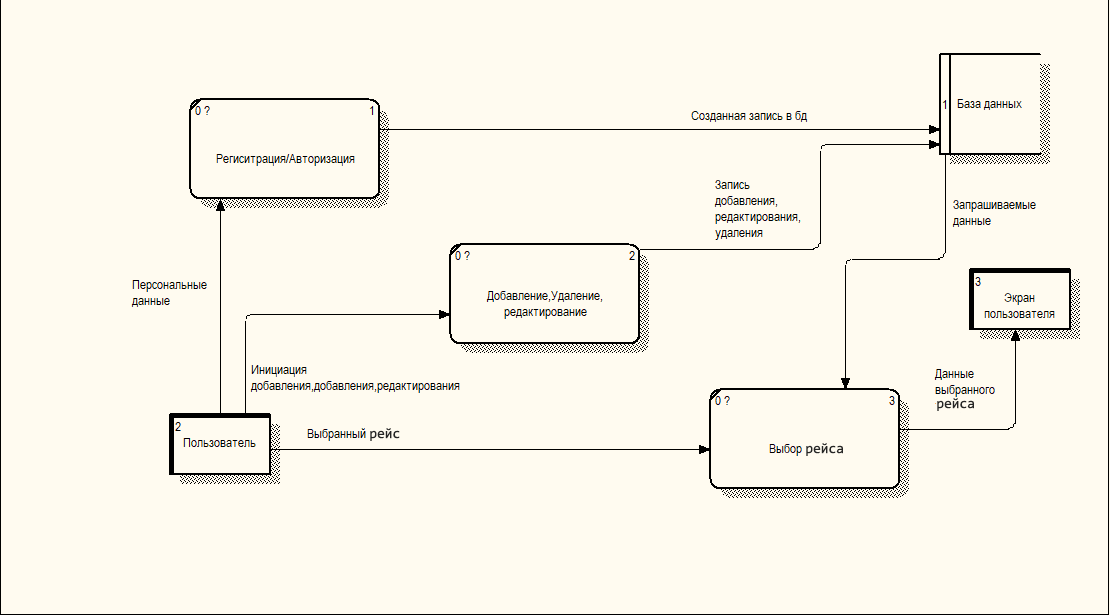


Рисунок 2 ― Диаграмма dfd

1. Проектная часть
   1. Краткое описание требований к разрабатываемому прототипу
      1. Функциональные требования
2. Приложение должно отображать список всех рейсов.
3. Приложение должно предоставлять детальную информацию о выбранном рейсе.
4. Приложение должно давать возможность стать админом при прохождении регистрации, для этого во время регистрации нужно ввести код админа.
5. Админ должен получать авторизированный доступ админа после входа. Также должен иметь возможность выхода из режима.
6. Админ должен иметь возможность добавления, удаления и редактирования рейсов.
7. Админ должен загружать изображение рейса по картинке.
   * 1. Нефункциональные требования
8. Приложение должно функционировать на устройстве с операционной системой Android 8.1 или выше.
9. Для корректной работы приложения требуется наличие Интернет-соединения.
10. Приложение не должно тормозить.
11. Приложение должно быть на русском языке.
12. Приложение не должно запускаться больше 10 секунд.
13. Пользовательский интерфейс должен быть интуитивно понятным.
14. Пользовательский интерфейс не должен содержать токсичных цветом, а именно цветовая палитра приложения должна быть мягкой и сочетаться.
    1. Описание и обоснование выбора информационного и программного обеспечения

Для разработки приложения будет использоваться интегрированная среда разработки для работы с платформой Android от компании Google, кроме того, имеет встроенный эмулятор Android со всеми необходимыми версиями. Была выбрана не только по тому, что является официальным ПО для разработки Android приложений, но и сама среда очень гибкая, имеет большой набор функций и процесс разработки настраивается под разработчика.

Приложение пишется на официальном языке Android разработки Kotlin. Он был выбран по причине того, что покрывает все возможности языка в более компактном виде, легче читается. Удобен в использовании и имеет дополнительный функционал, которых нет в Java. Кроме того, он более безопасный и имеет проверку на null значения.

В качестве СУБД была выбрана Firebase.

Firebase это платформа для разработки мобильных приложений от компании Google, в которой есть самые современные функции для разработки, перекомпоновки и улучшения приложений.

Firebase имеет следующие достоинства: работает на платформе Google: Firebase работает на базе Google, одного из самых известных и надежных брендов в мире технологий, для работы не требуется сервер: Firebase поставляется с архитектурой, в которой нет серверов и в которой оплата производится на основе запросов, в которой нет необходимости управлять инфраструктурой серверов и даже беспокоиться о ней, безопасность: Firebase обеспечивает оптимальную безопасность и доступность данных за счет осуществления регулярного резервного копирования.

Для удобства работы с изображениями была использована библиотека Picasso с открытым исходным кодом для загрузки изображений по ссылке.

* 1. Архитектура программной системы

Архитектура разрабатываемой программной системы двух уровневая, когда уровень представлений лежит на клиента, а база данных на сервере. (Рисунок 3). Приложение использую интернет для отправки и получения данных. Данные от приложения отправляются в облачную базу данных, а от туда обратно пользователю.

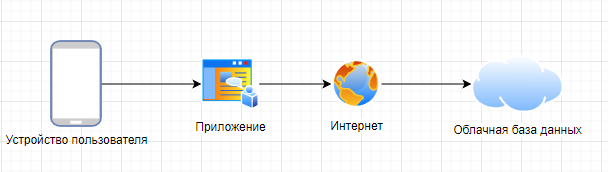


Рисунок 3 ― Архитектура программной системы

* 1. Описание структуры данных

В качестве базы данных была выбрана Realtime Database от компании Google, которая управляется платформой firebase. Это сервис со множеством полезных функций, который не требует отдельного сервера. Так же компания Google является лидирующей в сфере IT, так что в ее надежности сомневаться не приходится.

Realtime Database является облачной базой данных и хранит данные на основе JSON. В моем случае я реализовал каскадную модель для двух таблиц Admin и Product, ShopProduct(Рисунок 4).



Рисунок 4 ― Структура базы данных

1. Технологическая часть
   1. Подробное описание разрабатываемой системы

Классом, который отвечает за отображение списка рейсов и получение данных из бд(Листинг 1).

Листинг 1— Исходный код MainActivity.kt

|  |
| --- |
| class MainActivity : AppCompatActivity(),ListAdapter.ClickListener {  lateinit var binding: ActivityMainBinding  var ListAdapter:ListAdapter?=null  private var database: FirebaseDatabase = FirebaseDatabase.getInstance()  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  binding = ActivityMainBinding.inflate(*layoutInflater*)  setContentView(binding.*root*)   binding.bottomMenu.setOnNavigationItemSelectedListener **{** when (**it**.*itemId*) {  R.id.*admin* -> {  val admin = Intent(this, RegAvtorization::class.*java*)  startActivity(admin)  }  R.id.*ticket*->{  val admin = Intent(this, YourTickets::class.*java*)  startActivity(admin)  }  }   true  **}** binding.recUser.*layoutManager*= GridLayoutManager(this,2)  ListAdapter = ListAdapter(this)  binding.recUser.*adapter*=ListAdapter  ListAdapter?.loadListToAdapter(getData())  }  fun getData():ArrayList<FlightModel>{     val List=ArrayList<FlightModel>()  database.getReference("Flight").get().addOnSuccessListener **{** for (t in **it**.*children*){  for(item in t.*children*){  var flight=item.getValue(FlightModel::class.*java*)  if(flight!=null){  List.add(flight)   }  }    }  ListAdapter?.loadListToAdapter(List)  **}** return List  }  override fun onClick(flight: FlightModel) {  startActivity(Intent(this, BuyTicket::class.*java*).*apply* **{** putExtra("flightName",flight.name)  putExtra("flightTime",flight.time)   **}**)  }  override fun onStart() {  super.onStart()  ListAdapter?.loadListToAdapter(getData())  } } |

Классом, отвечающим за то, как функционирует ресайклер и распределяются данные, является рейсайклер адаптер (Листинг 2). Он получает в себя список актуальных данных, создает холдер каждого элемента, распихивает данные в нужные поля, назначается слушателя и выводит пользователю.

Листинг 2— Фрагмент исходного кода ListAdapter.kt

|  |
| --- |
| class ListAdapter(val clickListener: ClickListener) : RecyclerView.Adapter<ListAdapter.ViewHolder>() {   private var ListInAdapter = ArrayList<FlightModel>()   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ListAdapter.ViewHolder {  val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*element*, parent, false)   return ViewHolder(view)  }   override fun onBindViewHolder(holder: ListAdapter.ViewHolder, position: Int) {  holder.bind(ListInAdapter[position], clickListener)  }   override fun getItemCount(): Int {  return ListInAdapter.size  }   class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  val binding = ElementBinding.bind(itemView)  fun bind(flight: FlightModel, clickListener: ClickListener) {  binding.elName.*text* = flight.name  binding.elPrice.*text* = flight.price  binding.elTime.*text* = flight.time  Picasso.get().load(flight.link).fit().into(binding.elImage)   itemView.setOnClickListener **{** clickListener.onClick(flight)  **}** }  }   fun loadListToAdapter(productList: ArrayList<FlightModel>) {  ListInAdapter = productList  notifyDataSetChanged()  }   interface ClickListener {  fun onClick(flight: FlightModel) {   }  }   fun deleteItem(i: Int): String? {  var id = ListInAdapter.get(i).name   ListInAdapter.removeAt(i)   notifyDataSetChanged()   return id  } } |

Класс, который отвечает добавление рейсов(Листинг 3).

Листинг 3— Исходного кода add.kt

|  |
| --- |
| class Add : AppCompatActivity() {  private lateinit var binding: ActivityAddBinding  private var database: FirebaseDatabase = FirebaseDatabase.getInstance()  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  binding= ActivityAddBinding.inflate(*layoutInflater*)  setContentView(binding.*root*)   binding.addMovie.setOnClickListener **{** val flight=FlightModel(binding.addName.*text*.toString(),binding.addDesc.*text*.toString(),binding.addPrice.*text*.toString(),binding.addTime.*text*.toString(),binding.addLink.*text*.toString())   database.getReference("Flight").addListenerForSingleValueEvent(object :  ValueEventListener {  override fun onDataChange(snapshot: DataSnapshot) {  if(snapshot.child(flight.name.*toString*()).child(flight.time.*toString*()).exists()){  Toast.makeText(this@Add,"рейс на данное время уже существует", Toast.*LENGTH\_SHORT*).show()  }  else{   database.getReference("Flight").child(flight.name.*toString*()).child(flight.time.*toString*()).setValue(flight)  finish()  }  }   override fun onCancelled(error: DatabaseError) {  *TODO*("Not yet implemented")  }   })  **}** binding.addImage.setOnClickListener **{** try {  Picasso.get().load(binding.addLink.*text*.toString()).fit().into(binding.addImage)  }catch (ex:Exception){  Toast.makeText(this,"Нет ссылки на картинку", Toast.*LENGTH\_SHORT*).show()  }   **}** } } |

Класс который отвечает за покупку билетов (Листинг 4).

Листинг 4— Исходного кода BuyTicket.kt

|  |
| --- |
| class BuyTicket : AppCompatActivity() {  lateinit var binding: ActivityBuyTicketBinding  private var flight:FlightModel?=null  private var database: DatabaseReference = FirebaseDatabase.getInstance().getReference()  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  binding= ActivityBuyTicketBinding.inflate(*layoutInflater*)  setContentView(binding.*root*)   val name= *intent*.getStringExtra("flightName").*toString*()  val time= *intent*.getStringExtra("flightTime").*toString*()   getFlight(name,time)   binding.buyBuy.setOnClickListener **{** val id = userModel.currentuser?.phone.*toString*()  val ticket=ticketModel(id,flight?.name,flight?.price,flight?.time,flight?.link)  database.child("Ticket").addListenerForSingleValueEvent(object : ValueEventListener{  override fun onDataChange(snapshot: DataSnapshot) {  if(snapshot.child(id).child(flight?.name.*toString*()).exists()){  Toast.makeText(this@BuyTicket,"У вас уже куплен билет",Toast.*LENGTH\_LONG*).show()  }  else{  database.child("Ticket").child(id).child(flight?.name.*toString*()).setValue(ticket)  Toast.makeText(this@BuyTicket,"Вы купили билет",Toast.*LENGTH\_LONG*).show()  }  }   override fun onCancelled(error: DatabaseError) {  *TODO*("Not yet implemented")  }   })  **}** }  fun getFlight(name:String,time:String){   database.child("Flight").child(name).child(time).get().addOnSuccessListener **{** flight=**it**.getValue(FlightModel::class.*java*)   Picasso.get().load(flight?.link).fit().into(binding.descImage)   binding.descName.setText(flight?.name)  binding.descDesc.setText(flight?.description)  binding.descTime.setText(flight?.time)  binding.descPrice.setText(flight?.price)  **}** }  } |

* 1. Тестирование и верификация программного продукта

Тестирование приложения проводилось методом «Черного ящика» при доступе к базе данных, это один из самых популярных способов протестировать приложение и проверить его на наличие ошибок, багов и недоработок как для обычного пользователя, так и для программиста, имеющего весь код приложения.

При запуске приложения должны погружаться все рейсы из базы данных (Рисунок 6).

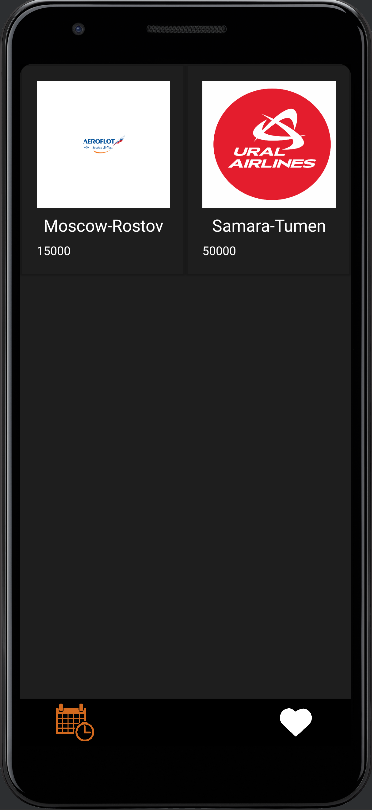


Рисунок 6 ― Стартовый экран

При нажатии на выбранный рейс в обычном режиме должна прогрузиться не редактируемая информация о выбранном рейсе (Рисунок 7).



Рисунок 7 ― Детальный просмотр

При переходе в режим админа, должен встречать экран регистрации или входа (Рисунок 8).

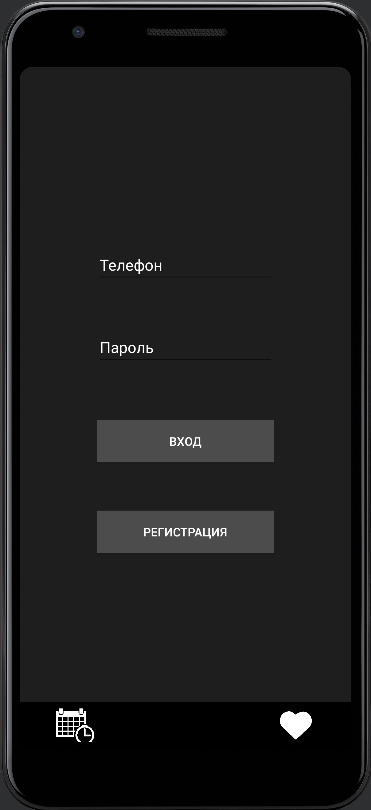


Рисунок 8 ― Регистрация или вход

При нажатии на регистрацию, должны появиться поля, где необходимо ввести нужные данные (Рисунок 9).

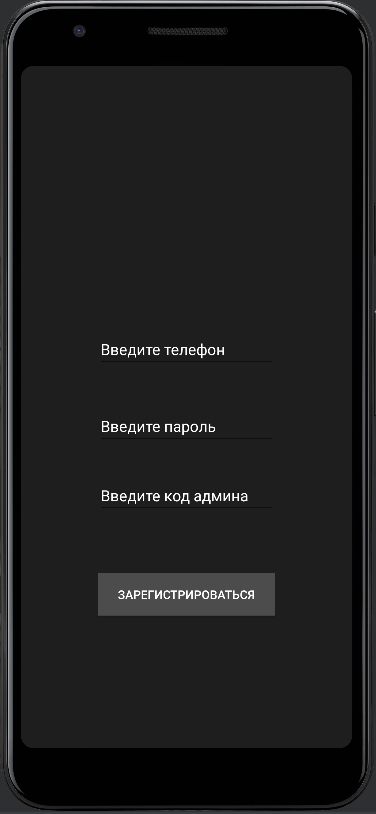


Рисунок 9 ― Регистрация

В режиме админа должен появиться список всех рейсов, кнопка добавление (Рисунок 10).

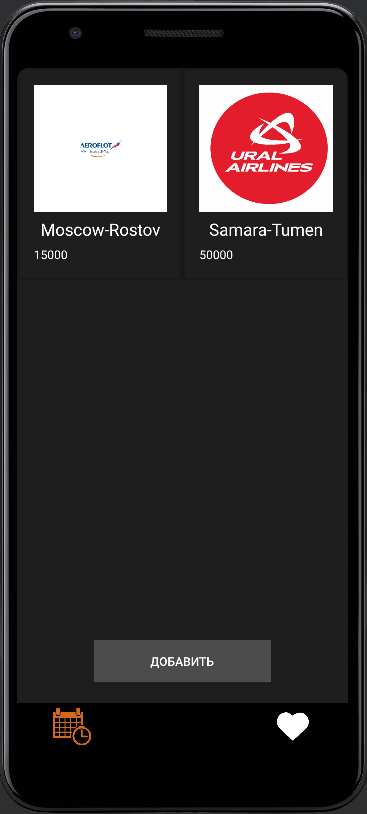


Рисунок 10 ― Режим админа

Во время добавления должны появиться поля для ввода данных о р , картинка для отображения загружаемого изображения и кнопка добавления (Рисунок 11).

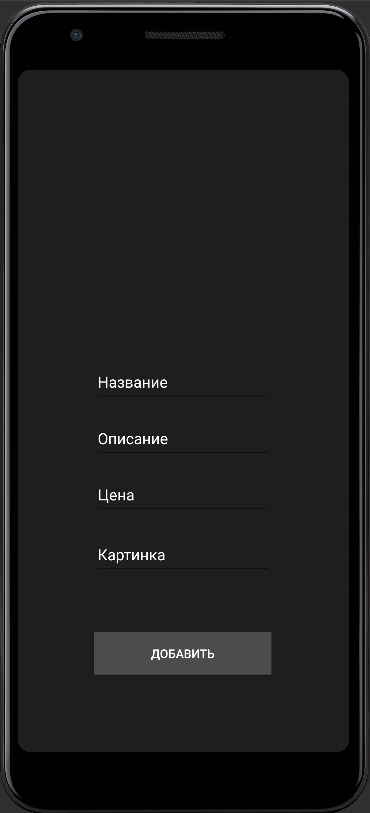


Рисунок 11 ― Добавление

ЗАКЛЮЧЕНИЕ

Результатом выполнения данной работы является разработанная мной программная система расписания авиаперевозок.

Для достижения цели работы был выполнен перечень задач. Была исследована предметная область, определены цель работы и задачи, описаны объекты и предмет исследования, также используемые методы исследования и информационная база исследований. Охарактеризована структура работы по разделам.

Сперва была выполнена исследовательская часть, в которой были проанализированы аналоги с учетом их достоинств и недостатков, чтобы понять, в какую сторону будет разрабатываться приложение. Также была сделана структурная диаграмма и диаграмма информационных потоков, чтобы понять, как и на какие блоки будет разделяться приложение и как взаимодействовать с хранилищами данных и внешними сущностями.

После выполнялась проектная часть, в которой были описаны функциональные требования к приложению, то есть что приложение должно делать и чего от него ожидать и нефункциональные требования, то есть какие свойства приложение должно демонстрировать. Затем описывалось и обосновывалось информационное и программное обеспечение, то есть какими инструментами мы пользовались для создания приложения и по каким причинам. Кроме того, была представлена архитектура программной системы в целом и описана структура базы данных.

В конце выполнялась технологическая часть, где описывалась архитектура и структурное представление приложения. Затем описывается подробное описание разрабатываемой системы, что делает каждый класс. Затем проводилось тестирование программного обеспечения методом черного ящика, чтобы убедиться, что приложение работает как задумано.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ\э

1. Документация платформы Android для разработчиков — URL: https://developer.android.com/docs (Дата обращения: 28.06.2022)
2. Документация языка программирования Kotlin — URL: https://kotlinlang.org/docs (Дата обращения: 28.06.2022)
3. Лекции по Android разработке от Samsung IT Academy — URL: https://myitschool.ru/edu (Дата обращения: 28.06.2022)
4. Статья об архитектурах программного обеспечения — URL: https://javarush.ru/groups/posts/2519-chastjh-2-pogovorim-nemnogo-ob-arkhitekture-po (Дата обращения: 28.06.2022)
5. Документация Material Design от Google — URL: https://material.io/design/introduction (Дата обращения: 28.06.2022)
6. Документация библиотеки Picasso — URL: https://square.github.io/picasso/ (Дата обращения: 28.06.2022)
7. Документация по Microsoft SQL Server — URL: https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver16 (Дата обращения: 07.02.2019)
8. Документация по JTDC Driver — URL: http://jtds.sourceforge.net/doc.html (Дата обращения: 28.06.2022)
9. Разработка структуры базы данных — URL: https://www.internet-technologies.ru/articles/rukovodstvo-po-razrabotke-struktury-i-proektirovaniyu-bazy-dannyh.html (Дата обращения: 28.06.2022)
10. Курс по Kotlin и Android Studio — URL: https://[www.youtube.com/watch?v=-V8bDVhob3g&list=PLmjT2NFTgg1fdHN-](http://www.youtube.com/watch?v=-V8bDVhob3g&list=PLmjT2NFTgg1fdHN-) 9Wn4XYr-IOuadxMm5 (Дата обращения: 28.06.2022)