Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

им. Н.П. ОГАРЁВА»

(ФГБОУ ВО «МГУ им. Н.П. Огарёва»)

Факультет довузовской подготовки и среднего профессионального образования

Выпускающая предметная цикловая комиссия (кафедра) общепрофессиональных и  
специальных (информационно-коммуникационных) дисциплин

ОТЧЁТ

по учебной практике

Учебная практика по ПМ.01

студента III курса

Специальность 09.02.03 Программирование в компьютерных системах

Место прохождения практики ФГБОУ ВО «МГУ им. Н. П. Огарёва», корпус №1, аудитория 243

Срок прохождения практики с 01 декабря 2021 г. – 21 декабря 2021 г.

Автор отчёта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. Р. Чураков

подпись, дата

Обозначение отчёта: ОП–02069964– У–09.02.03–38–21

Руководитель практики,

преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н. А. Базеева

подпись, дата

Отчёт защищён \_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

Саранск 2021

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| 1 Правила безопасности на рабочем месте……………………………… | 4 |
| 2 Общая характеристика базы практики………………………………… | 5 |
| 3 День 1. Выбор методов, инструментов и технологий для реализации поставленных задач………………………………………………………. | \_\_\_6 |
| 4 День 2. Регистрация на сайте GitHub………………………………….. | 7 |
| 5 День 3. Создание Windows Forms приложений «Взаимодействие управляемого и неуправляемого кода»………………………………….. | \_\_\_ 8 |
| 6 День 4. Создание Windows Forms приложений «Асинхронное программирование»……………………………………………………….. | \_\_\_9 |
| 7 День 5. Создание Windows Forms приложений «Повышение удобства использования приложений»…………………………………... | \_\_\_10 |
| 8 День 6. Создание Windows Forms приложений «Подключение к базе данных»…………………………………………………………………….. | \_\_\_11 |
| 9 День 7. Верстка экрана «Заставка» и добавление логики……………. | 12 |
| 10 День 8. Верстка Начального экрана и добавление логики…………. | 13 |
| 11 День 9. Верстка экрана «Авторизация» и добавление логики……… | 14 |
| 12 День 10. Верстка экрана «Регистрация» и добавление логики……... | 15 |
| 13 День 11. Верстка экрана «Главная» и добавление логики…………... | 16 |
| 14 День 12. Верстка экрана «Заказы» и добавление логики……………. | 17 |
| 15 День 13. Верстка экрана «Избранное» и добавление логики……….. | 18 |
| 16 День 14. Верстка экрана «Сообщения» и добавление логики………. | 19 |
| 17 День 15. Верстка экрана «Профиль» и добавление логики…………. | 20 |
| 18 День 16. Тестирование приложения………………………………….. | 21 |
| ЗАКЛЮЧЕНИЕ……………………………………………………………. | 22 |
| СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ…………………………. | 24 |
| ПРИЛОЖЕНИЕ А (обязательное) Регистрация на сайте GitHub……… | 25 |
| ПРИЛОЖЕНИЕ Б (обязательное) Взаимодействие управляемого и неуправляемого кода……………………………………………………… | \_\_\_26 |
| ПРИЛОЖЕНИЕ В (обязательное) Асинхронное программирование…. | 28 |
| ПРИЛОЖЕНИЕ Г (обязательное) Повышение удобства использования приложений………………………………………………. | \_\_\_29 |
| ПРИЛОЖЕНИЕ Д (обязательное) Подключение к базе данных………. | 30 |
| ПРИЛОЖЕНИЕ Е (обязательное)Верстка Android приложения……… | 32 |
| ПРИЛОЖЕНИЕ Ж (обязательное)Тестирование приложения ……….. | 43 |
| ПРИЛОЖЕНИЕ И (обязательное)Листинг…….……………………….. | 47 |

**1 Правила безопасности на рабочем месте**

Безопасные условия труда на рабочем месте – это состояние условий труда, при которых воздействие на работающего опасных и вредных производственных факторов исключено, или воздействие вредных производственных факторов не превышает предельно допустимых значений.

В день начала учебной практики, 01 декабря 2021 года, была проведена установочная конференция, которая включала в себя инструктаж по технике безопасности.

Инструкция по технике безопасности при работе на компьютере включает в себя:

* необходимо аккуратно обращаться с проводами;
* запрещается работать с неисправным компьютером;
* нельзя заниматься очисткой компьютера, когда он находится под напряжением;
* недопустимо самостоятельно проводить ремонт оборудования при отсутствии специальных навыков;
* нельзя располагать рядом с компьютером жидкости, а также работать с мокрыми руками;
* нельзя в процессе работы ПК прикасаться к другим металлическим конструкциям.

Студенты могут быть подвергнуты следующими опасным и вредным воздействиям:

* поражение электрическим током;
* механические повреждения;
* электромагнитное излучения;
* опасность пожара;
* повышенный уровень шума и вибрации.

**2 Общая характеристика базы практики**

База практики – ФГБОУ ВО «МГУ им. Н. П. Огарева», корпус №1, аудитория 243 оснащена системным и программным обеспечением, используемым для организации производственного процесса. Она позволяет студентам проектировать, внедрять и эксплуатировать системное и прикладное программное обеспечения, тестировать и отлаживать программный продукт.

**3 День 1. Выбор методов, инструментов и технологий для реализации поставленных задач**

Для реализации поставленных задач были использованы такие программы и языки программирования:

1. Microsoft Visual Studio;
2. Android Studio;
3. Язык разметки XML;
4. Язык программирования Kotlin.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms.

Android Studio – интегрированная среда разработки производства Google, с помощью которой разработчикам становятся доступны инструменты для создания приложений на платформе Android OS от компании JetBrains, разработанная на основе платформы IntelliJ IDEA. Решения для Android разрабатываются в Android Studio с использованием Java или Kotlin.

XML – это расширяемый язык разметки, предназначенный для хранения структурированных данных, для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки, иногда называемых словарями.

Kotlin – это статически типизированный язык программирования, с открытым исходным кодом и разрабатываемый компанией JetBrains. Это язык, который сочетает в себе объектно-ориентированное программирование (ООП) и функциональное программирование на неограниченной, самодостаточной и самобытной платформе.

**4 День 2. Регистрация на сайте GitHub**

GitHub – это система управления проектами и версиями кода, а также платформа социальных сетей, созданная для разработчиков. GitHub позволяет вам работать совместно с другими людьми по всему миру, планировать свои проекты и отслеживать свою работу.

Git –  это система управления версиями, которая пришлась по душе практически всем  от разработчиков до дизайнеров. GitHub можно считать социальной сетью для хранения кода.

Переходим на сайт GitHub.com. Для регистрации на сайте необходимо перейти по кнопке Sign up.

Далее необходимо ввести свой адрес электронной почты, создать надёжный пароль и создать имя, которое будет отображаться на самом сайте. На адрес электронной почты придет восьмизначный код, для подтверждения регистрации. После прохождения всех этапов регистрации аккаунт на GitHub будет успешно создан. После этого возвращаемся к github и создаем новый репозиторий.

Выполненная работа представлена в приложении А, рисунки А.1 – А.2.

**5 День 3. Создание Windows Forms приложений «Взаимодействие управляемого и неуправляемого кода»**

Открываем Visual Studio и создаем новый проект Windows Forms. Добавляем на форму элементы OpenFileDialog и MenuStrip. Изменяем свойства для заданных элементов. Следующим шагом будет настройка поведения кнопки «**Показать рисунок».** Когда пользователь нажимает эту кнопку, приложение открывает окно OpenFileDialog. Если пользователь открывает файл изображения, приложение показывает это изображение в PictureBox.

Выполненная работа представлена в приложении Б, рисунки Б.1 – Б.4.

**6 День 4. Создание Windows Forms приложений «Асинхронное программирование»**

Создаём новый проект Windows Forms. Назовем его WinBackgroundWorker. Добавляем на форму элементы управления: два элемента Label, TextBox, ProgressBar и две кнопки Button.

Из Toolbox перетаскиваем элемент BackgroundWorker в форму. В окне Properties устанавливаем свойства WorkerSupportsCancellation и WorkerReportsProgress в True (для поддержки асинхронной отмены и возможности сообщения основному потоку информации о продвижении фонового процесса соответственно). Дважды нажмем на BackgroundWorker, чтобы открыть обработчик события backgroundWorker1\_DoWork по умолчанию. Добавляем к этому обработчику события код. Для элемента backgroundWorkerl в окне Properties щелкаем кнопку Events, затем дважды нажимаем на ProgressChanged, чтобы открыть окно кода обработчика события backgroundWorker1\_ProgressChanged. Для кнопки Cancel открываем обработчик события Click. Добавляем. Строим и выполняем приложение, и проверяем его функциональность. Проверяем работоспособность приложения.

Выполненная работа представлена в приложении В, рисунок В.1 – В.3.

**7 День 5. Создание Windows Forms приложений «Повышение удобства использования приложений»**

Открываем Windows-приложение WinAsynchMethod. Открываем форму в режиме конструктора. Выбираем пункт меню View – ToolBox. Добавляем ЭУ HelpProvider на форму. Выделяем поле txbA для отображения ее свойств. Для свойства HelpString on helpProvider1 задаем значение For input integer A. Строим и запускаем приложение. Перемещаемся по форме, используя клавишу Tab, до тех пор, пока поле txbA не окажется в фокусе. Нажимаем на клавишу F1 для отображения контекстной справки для поля txbA.

Выполненная работа представлена в приложении Г, рисунок Г.1 – Г.3.

**8 День 6. Создание Windows Forms приложений «Подключение к базе данных»**

Создаем в Visual Studio новое приложение Windows Forms. Называем его WinBD. В окне Server Explorer нажимаем правой кнопкой узел Data Connections и выбераем команду Add Connection – при первом добавления подключения откроется диалоговое окно Choose Data Sourse. Если вместо диалогового окна Choose Data Source появится диалоговое окно Add Connection, нажимаем кнопку Change, в случае необходимости в изменении источника данных. В диалоговом окне Choose Data Source выбираем источник данных – Microsoft Access Database File (Файл базы данных Microsoft Access), а также провайдер данных для подключения. В окне Add Connection в поле Database file name щелкаем кнопку Brouse. Находим файл Конфетная фабрика.mdb и выбираем его, нажав кнопку Open.

Нажимаем кнопку Test Connection (Проверка соединения), чтобы проверить соединение, – должно появиться сообщение об успешной проверке. Нажмем кнопку ОК – в окне Server Explorer появится новое соединение.

Выполненная работа представлена в приложении Д, рисунок Д.1 – Д.5.

**9 День 7. Верстка экрана «Заставка» и добавление логики**

Создаем новое Activity под названием «SplashActivity» и переходим в XML файл, который создался автоматически.

В XML файл добавим TextView с кодом:

android:layout\_marginTop="100dp"

android:text="Добро пожаловать!"

android:textColor="@color/black"

android:textSize="30sp"

Добавляем ImageView для отображения картинки.

Перейдем в файл Kotlin для добавления логики перехода на другой экран. Подключаем библиотеку Binding для связывания с XML файлом и получение идентификаторов разметки. Воспользуемся функций перехода с задержкой на 3 секунды для загрузки данных приложения.

Выполненная работа представлена в приложении Е, рисунок Е.1.

**10 День 8. Верстка Начального экрана и добавление логики**

Создаем новое Activity под названием «MainActivity» и переходим в XML файл, который создался автоматически.

В XML файл добавим три TextView и одну MaterialButton.

Для первой TextView задаем:

android:layout\_marginStart="20dp"

android:layout\_marginTop="70dp"

android:textColor="@color/black"

android:textSize="24sp"

android:textStyle="bold"

Для второй TextView задаем:

android:layout\_marginTop="20dp"

android:paddingStart="10dp"

android:paddingEnd="10dp"

Для MaterialBotton задаем:

android:id="@+id/loginButton"

android:layout\_marginBottom="80dp"

android:backgroundTint="@color/blue"

android:paddingTop="15dp"

android:paddingBottom="15dp"

android:text="Заказать сейчас"

Переходим в Kotlin файл и подключаем библиотеку Binding для связывания с XML файлом и получение идентификаторов разметки.

Импортируем Intent для перехода на другую активность. По нажатию на loginButton мы перейдем на LoginActivity.

Выполненная работа представлена в приложении Е, рисунок Е.2.

**11 День 9. Верстка экрана «Авторизация» и добавление логики**

Создаем новое Activity под названием «LoginActivity» и переходим в XML файл, который создался автоматически.

В XML файл добавим 6 TextView, 2 TextInputEditText и 2 MaterialButton.

Располагаем элементы согласно макету.

Создадим класс Validator для проверки почты и пароля. Реализуем функции проверки.

Для почты:

а) количество символов должно быть больше 8;

б) почта должна содержать «@»;

в) почта не должна быть пустой.

Для пароля:

а) количество символов должно быть больше 8;

б) пароль не должен быть пустым.

Переходим в Kotlin файл и принимаем функции из Validator.kt. Для отображения ошибок используем команду error. Если ошибки отсутствуют, переходим на Главный экран при помощи Intent.

Выполненная работа представлена в приложении Е, рисунок Е.3.

**12 День 10. Верстка экрана «Регистрация» и добавление логики**

Создаем новое Activity под названием «RegisterActivity» и переходим в XML файл, который создался автоматически.

В XML файл добавим 7 TextView, 4 TextInputEditText и MaterialButton.

Располагаем элементы согласно макету.

Переходим в класс Validator. К имеющим функциям добавляем проверку имени пользователя и номера телефона.

Для имени пользователя:

а) количество символов должно быть больше 4;

б) имя пользователя не должно быть пустым.

Для номера телефона:

а) количество символов должно быть больше 10;

б) номер телефона не должен быть пустым.

Переходим в Kotlin файл и принимаем функции из Validator.kt. Также используем функции для проверки почты и пароля. Для отображения ошибок используем команду error. Если ошибки отсутствуют, переходим на Главный экран при помощи Intent.

Выполненная работа представлена в приложении Е, рисунок Е.4.

**13 День 11. Верстка экрана «Главная» и добавление логики**

Создаем новый Fragment под названием «HomeFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавим 9 TextView, 3 MaterialButton, 3 ImageView, TextInputEditText и RecyclerView.

Располагаем элементы согласно макету.

Создаем новый XML файл и верстаем карточку для RecyclerView.

Создаем Adapter для RecyclerView под названием «HotelsAdapter». Добавляем стандартные функции для отображения элементов списка.

В Kotlin файле создадим список информации для заполнения RecyclerView. Присвоим recyclerHotels.adapter ранее созданный нами HotelNumberAdapter. Добавляем список для заполнения карточки RecyclerView.

Создаем класс DataPickerDialog. Он потребуется для выбора дня посещения отеля в диалоговом окне. В основном классе создаем функцию вызова диалогового окна по нажатию на кнопку и передачу данных.

Создаем класс DataPickerHightDialog. Он потребуется для выбора количества дней посещения отеля в диалоговом окне. В основном классе создаем функцию вызова диалогового окна по нажатию на кнопку и передачу данных.

Выполненная работа представлена в приложении Е, рисунок Е.5 – Е.6.

**14 День 12. Верстка экрана «Заказы» и добавление логики**

Создаем новый Fragment под названием «OrderFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавим Toolbar, ImageView, TextView и MaterialButton.

Для Toolbar задаем:

android:id="@+id/orderToolbar"

app:title="Бронь"

Для TextView задаем:

android:layout\_marginStart="80dp"

android:layout\_marginTop="30dp"

android:layout\_marginEnd="80dp"

android:text="@string/helper\_text\_order"

android:textAlignment="center"

Для MaterialButton задаем:

android:id="@+id/orderButton"

android:layout\_marginStart="100dp"

android:layout\_marginTop="30dp"

android:layout\_marginEnd="100dp"

android:backgroundTint="@color/blue"

android:paddingTop="15dp"

android:paddingBottom="15dp"

android:text="Заказать сейчас"

android:textSize="12sp"

app:cornerRadius="10dp"

Переходим в Kotlin файл и подключаем библиотеку Binding для связывания с XML файлом и получение идентификаторов разметки. Добавляем слушатель нажатий для перехода на HomeFragment.

Выполненная работа представлена в приложении Е, рисунок Е.7.

**15 День 13. Верстка экрана «Избранное» и добавление логики**

Создаем новый Fragment под названием «LikedFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавляем Toolbar и RecyclerView.

Для Toolbar задаем:

android:id="@+id/likedToolbar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:title="Избранное"

Для RecyclerView задаем:

android:id="@+id/hotelNumberRecyclerView"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" app:layoutManager="android.recyclerview.widget.LinearLayoutManager"

Создаем Adapter для RecyclerView под названием «HotelNumberAdapter». Добавляем стандартные функции для отображения элементов списка.

В Kotlin файле создадим список информации для заполнения RecyclerView. Присвоим hotelNumberRecyclerView.adapter ранее созданный нами HotelNumberAdapter.

Выполненная работа представлена в приложении Е, рисунок Е.8.

**16 День 14. Верстка экрана «Сообщения» и добавление логики**

Создаем экран «MessageFragment»

В XML добавим ViewPager2 и TabLayout. Располагаем их согласно макетую

Создаем Adapter для ViewPager2 под названием «ViewPagerAdapter». Добавляем стандартные функции для отображения IncomingFragment и NotificationFragment.

В Kotlin файле добавляем функцию для пролистывания экранов. Присвоим viewPagerMessage.adapter ранее созданный нами ViewPagerAdapter.

Создаем новый Fragment под названием «IncomingFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавляем ImageView и TextView.

Создаем новый Fragment под названием «IncomingFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавляем 2 ImageView и 2 TextView.

Располагаем элементы согласно макету.

Выполненная работа представлена в приложении Е, рисунок Е.95 – Е.10.

**17 День 15. Верстка экрана «Профиль» и добавление логики**

Создаем новый Fragment под названием «ProfileFragment» и переходим в XML файл, который создался автоматически.

В XML файл добавляем Toolbar, 4 ImageView и 7 TextView.

Располагаем элементы согласно макету.

Переходим в Kotlin файл и подключаем библиотеку Binding для связывания с XML файлом и получение идентификаторов разметки. Добавляем функцию получения почты, имени пользователя и пароля из RegisterActivity и присваиваем их TextView согласно макету.

Выполненная работа представлена в приложении Е, рисунок Е.11.

**18 День 16. Тестирования приложения**

Тестирование приложения – это обязательный этап технической разработки приложения. Он играет важнейшую роль в процессе создания. Целью тестирования является общая проверка реального функционирования приложения.

В тестирование входит проверка работы, проверка разрешений и плотностей пикселей (DPI). Проверка всех кнопок. Тестирование приложения происходило в таких телефонах, как Pixel XL, Pixel 3 и Pixel. В ходе тестирования ошибки не найдены. Приложение корректно отображается и функционирует во всех телефонах.

Выполненная работа представлена в приложении Ж, рисунок Ж.1 – Ж.4.

**ЗАКЛЮЧЕНИЕ**

В результате прохождения учебной практики по ПМ.01 были приобретены и усовершенствованы навыки программирования на языке программирования C#, а также верстка Android приложения по макету.

В результате прохождения учебной практики были освоены общие и профессиональные компетенции:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.

ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения задания.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

В результате освоения компетенций базовой подготовки специальности

09.02.03 Программирование в компьютерных системах техник-программист должен обладать профессиональными компетенциями, соответствующими основным видам профессиональной деятельности:

ПК 1.1. Выполнять разработку спецификаций отдельных компонент

ПК 1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей

ПК 1.5. Осуществлять оптимизацию программного кода модуля.

ПК 1.6. Разрабатывать компоненты проектной и технической документации с использованием графических языков спецификаций.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 СМК СТО 020–2021. Практика студентов среднего профессионального образования. Общие требования, правила оформления отчётности. Стандарт организации : издание официальное : утвержден и введен в действие приказом ФГБОУ ВО «МГУ им. Н.П. Огарёва» от 18 августа 2021 г. № 596 : введен взамен СТО СМК 020-2018 «Практика студентов среднего профессионального образования. Организация, общие требования, правила оформления отчётности» : дата введения 2021-08-18 / подготовлен отделом менеджмента качества образовательной деятельности ФГБОУ ВО «МГУ им. Н. П. Огарёва». – Саранск, 2021. – 30 с. – Текст : электронный.

2 СМК СТО 006–2020. Общие требования к построению, изложению и оформлению документов учебной деятельности. Стандарт организации : издание официальное : утвержден и введен в действие приказом врио ректора ФГБОУ ВО «МГУ им. Н.П. Огарёва» от 08 сентября 2020 г. № 490 : введен взамен СТО 006–2014 «Общие требования к построению, изложению и оформлению документов учебной деятельности» : дата введения 2020-09-08 / подготовлен отделом менеджмента качества образовательной деятельности. – Саранск, 2020. – 69 с. – Текст : электронный.

3 Гриффитс Д. Head First. Программирование для Android : Д. Гриффитс, Д. Гриффитс. – СПб. : Питер, 2018. – 912 с. – ISBN 978-5-4461-0708-7. – URL:

https://www.rulit.me/data/programs/resources/pdf/Head-First-Kotlin-\_RuLit\_Me\_ 610054.pdf. – Режим доступа: свободный. – Текст : электронный.

4 Гриффитс Д. Head First. Kotlin : руководство для начинающих программистов / Д. Гриффитс, Д. Гриффитс; [пер. с англ. Я. Е. Матвеев]. – СПб. Питер, 2020. – 446с. – ISBN 978-5-4461-1335-4. – URL: https://www.rulit.me/data/programs/resources/pdf/Head-First-Kotlin-\_RuLit\_Me\_ 610054.pdf. – Режим доступа: свободный. – Текст : электронный.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Регистрация на сайте GitHub**

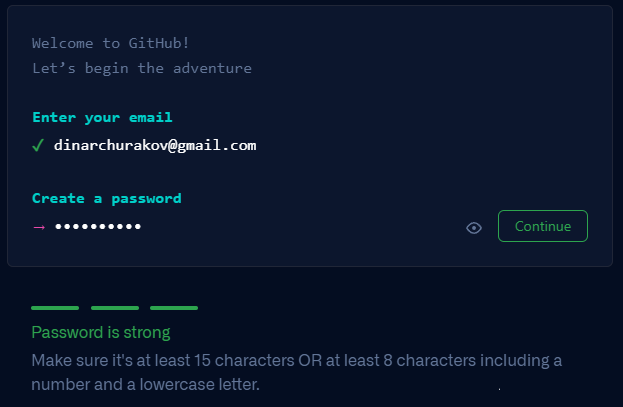


Рисунок А.1 – Регистрация на GitHub

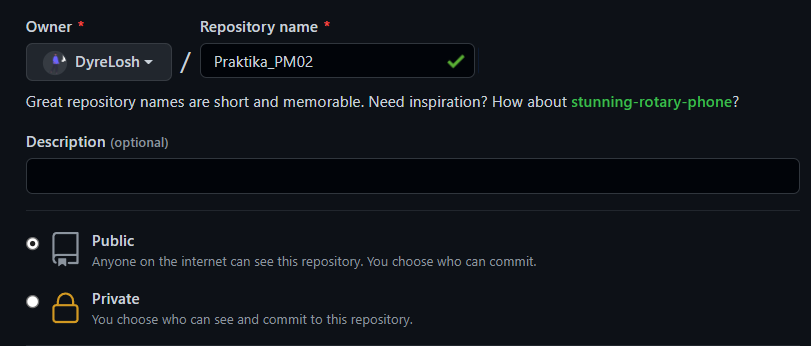
****

Рисунок А.2 – Создание репозитория

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Взаимодействие управляемого и неуправляемого кода**

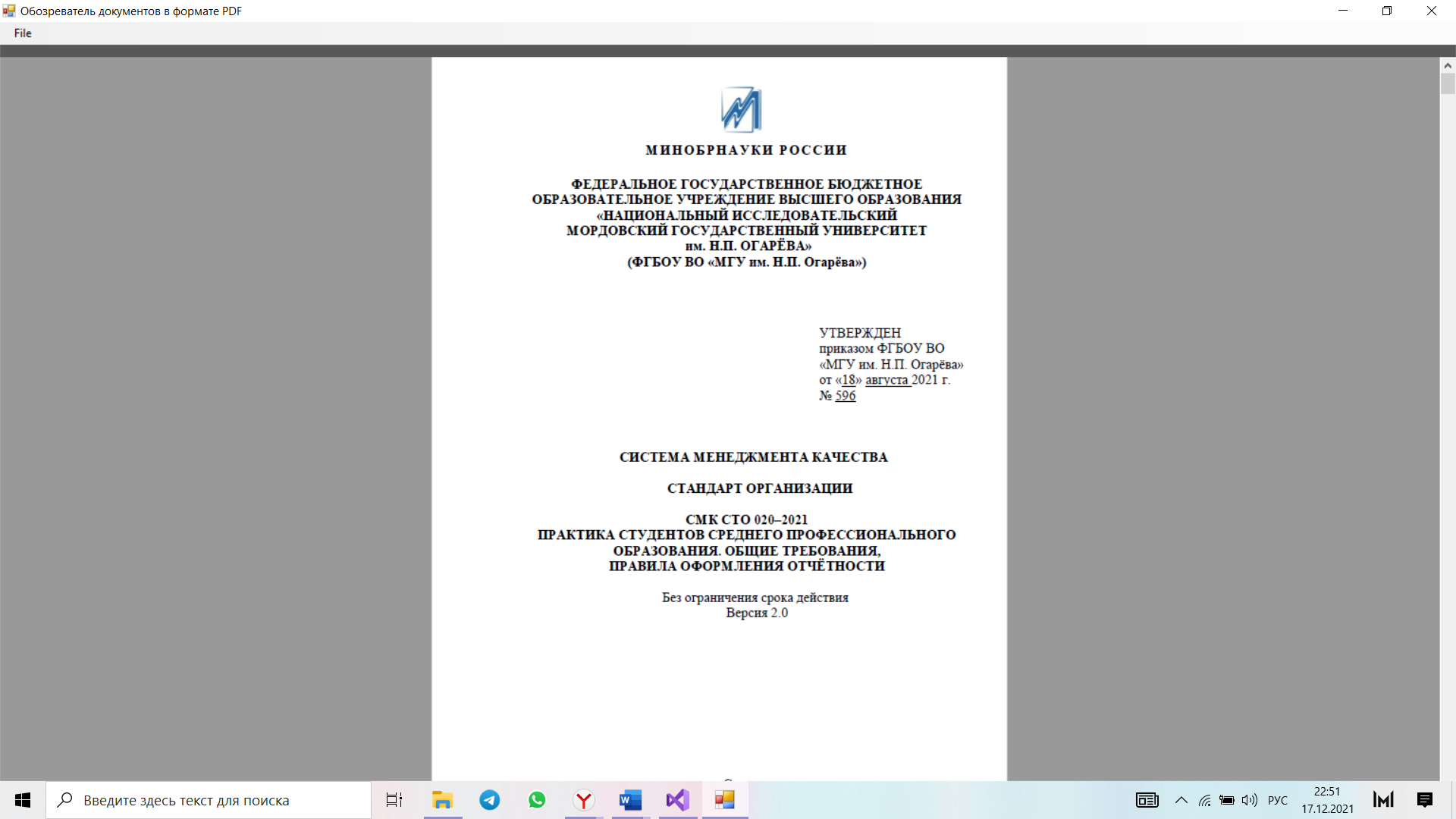


Рисунок Б.1 – Обозреватель документов в формате PDF

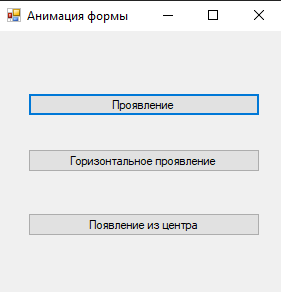


Рисунок Б.2 – Анимация формы

**Окончание приложения Б**

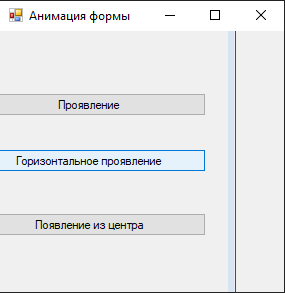


Рисунок Б.3 – Анимация формы

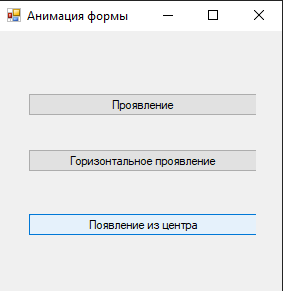


Рисунок Б.4 – Анимация формы

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Асинхронное программирование**

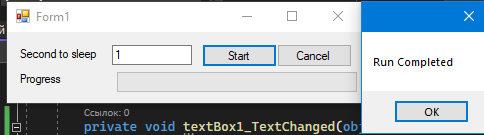


Рисунок В.1 – Работа с компонентом BackgroundWorker

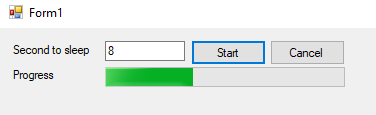


Рисунок В.2 – Использование делегатов

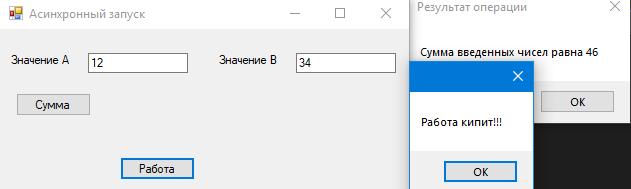


Рисунок В.3 – Асинхронный запуск произвольного метода

**ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Повышение удобства использования приложений**

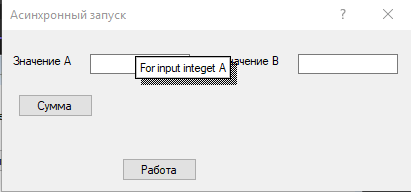


Рисунок Г.1 – Создание контекстной справки

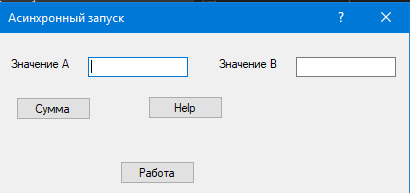


Рисунок Г.2 – Использование справочного файла

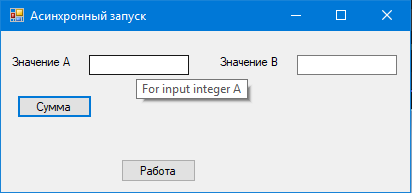


Рисунок Г.3 – Добавление всплывающих подсказок

**ПРИЛОЖЕНИЕ Д**

**(обязательное)**

**Подключение к базе данных**

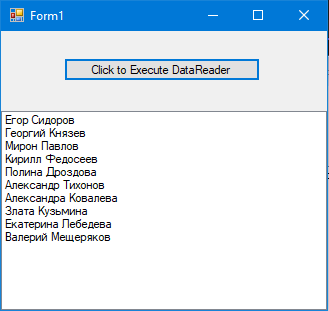


Рисунок Д.1 – Организация доступа к данным

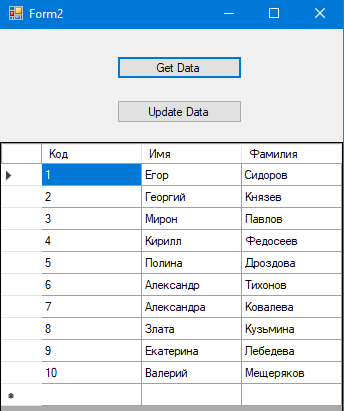


Рисунок Д.2 – Извлечение и обновление данных

**Окончание приложения Д**

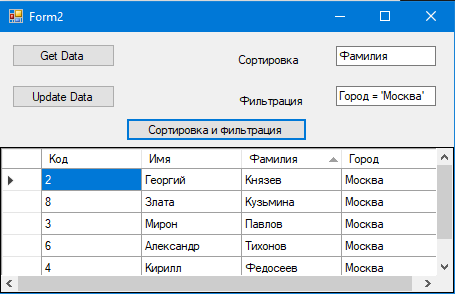


Рисунок Д.3 – Использование объектов DataView

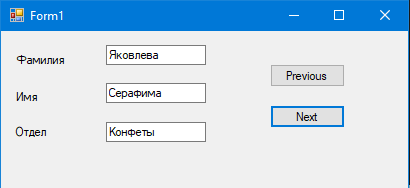


Рисунок Д.4 – Связывание данных с элементами управления

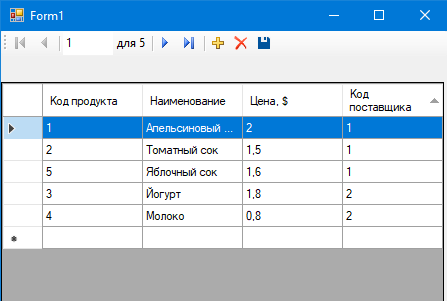


Рисунок Д.5 – Создание связанной с данными формы источников данных

**ПРИЛОЖЕНИЕ Е**

**(обязательное)**

**Вёрстка Android приложения**

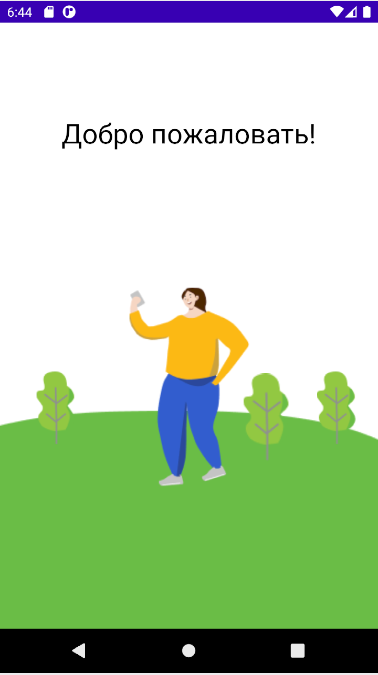
****

Рисунок Е.1 – Экран «Заставка»

**Продолжение приложение Е**

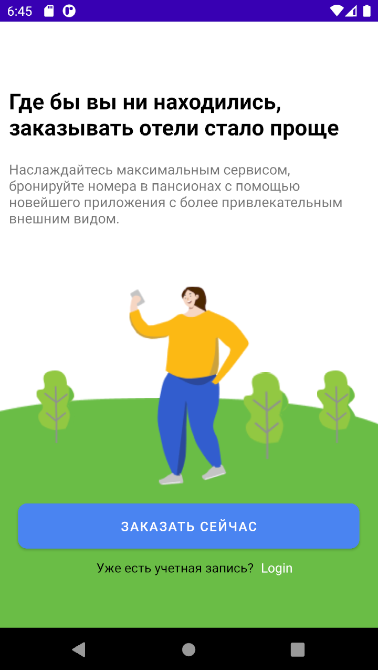


Рисунок Е.2 – Экран «Начальный»

**Продолжение приложение Е**

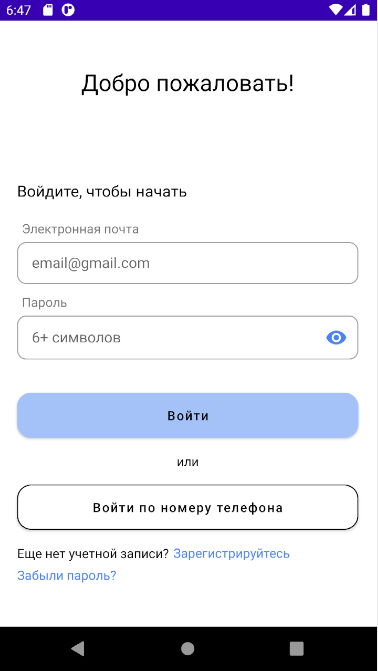
\

Рисунок Е.3 – Экран «Авторизация»

**Продолжение приложение Е**

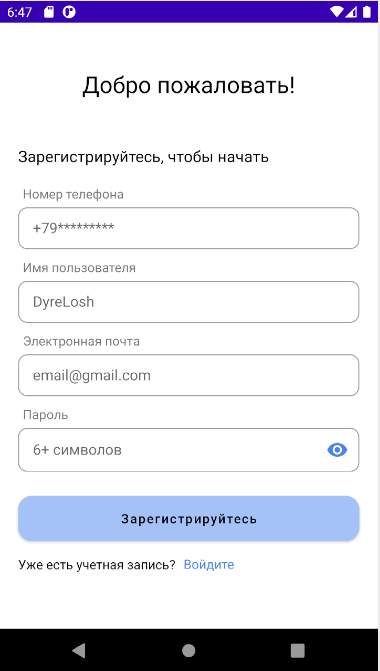


Рисунок Е.4 – Экран «Регистрация»

**Продолжение приложение Е**

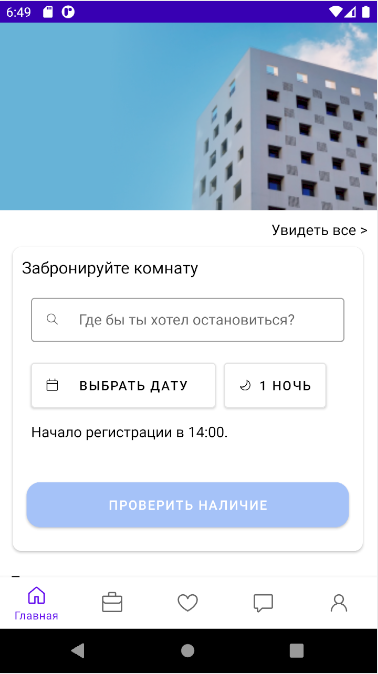


Рисунок Е.5 – Экран «Главная»

**Продолжение приложение Е**

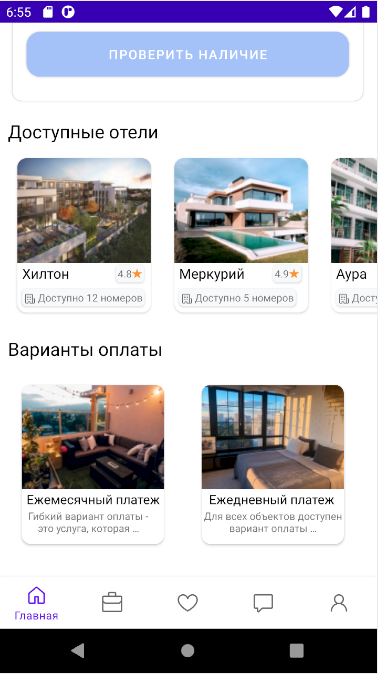


Рисунок Е.6 – Экран «Главная»

**Продолжение приложение Е**

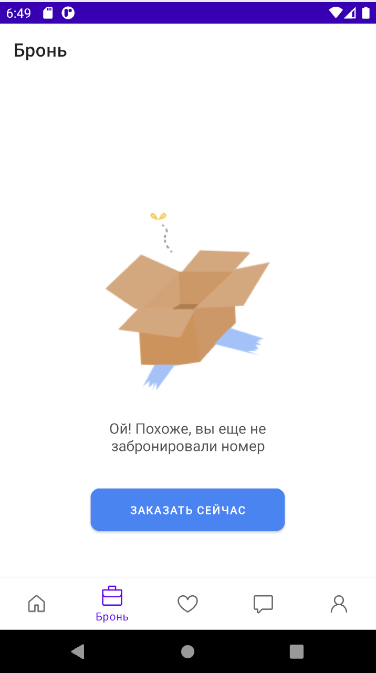


Рисунок Е.7 – Экран «Заказы»

**Продолжение приложение Е**

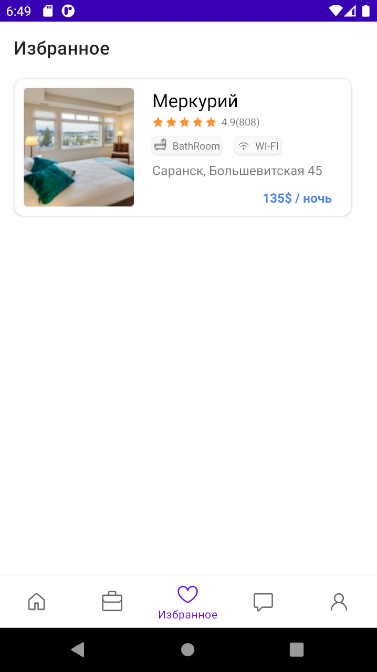


Рисунок Е.8 – Экран «Избранное»

**Продолжение приложение Е**



Рисунок Е.9 – Экран «Сообщения»

**Продолжение приложение Е**

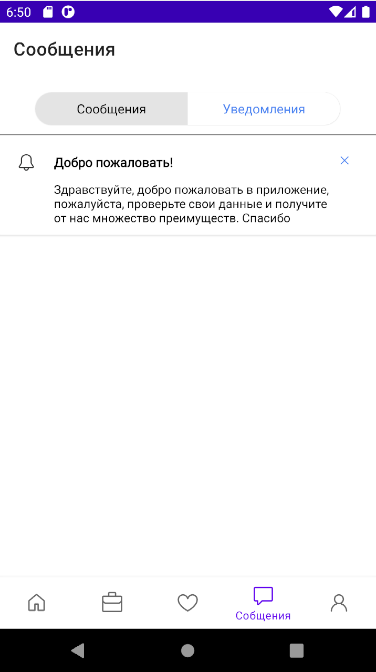


Рисунок Е.10 – Экран «Уведомления»

**Окончание приложение Е**

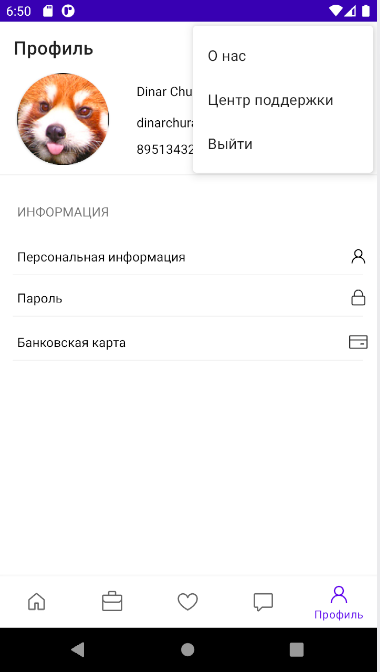


Рисунок Е.11 – Экран «Профиль»

**ПРИЛОЖЕНИЕ Ж**

**(обязательное)**

**Тестирование приложения**

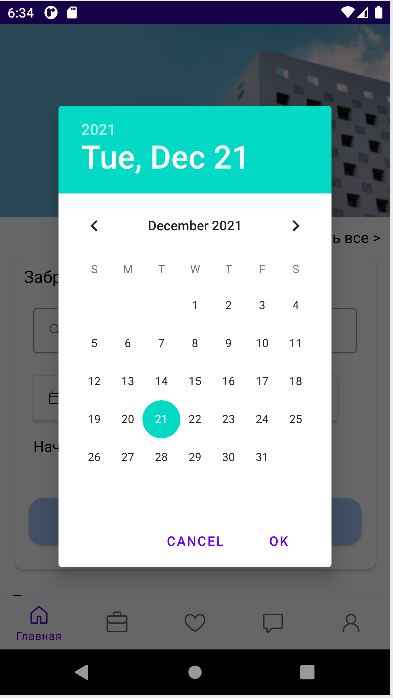


Рисунок Ж.1 – Тестирование диалогового окна

**Продолжение приложения Ж**

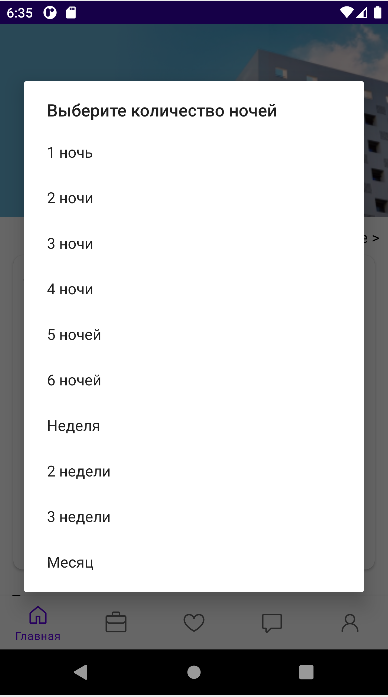


Рисунок Ж.2 – Тестирование диалогового окна

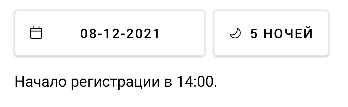


Рисунок Ж.3 – Тестирование передачи из диалогового окна

**Продолжение приложения Ж**

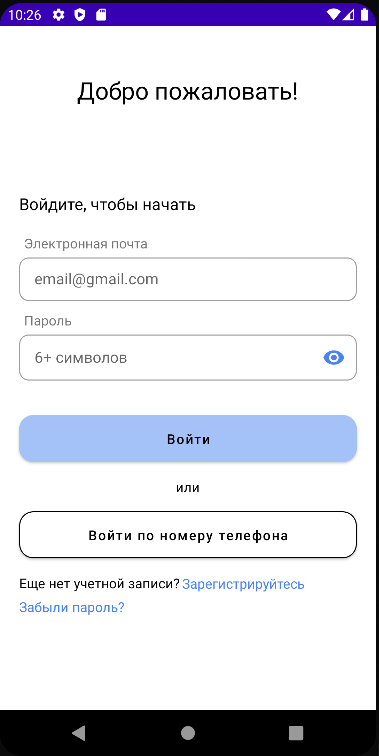


Рисунок Ж.3 – Тестирование на Pixel 3

**Окончание приложения Ж**

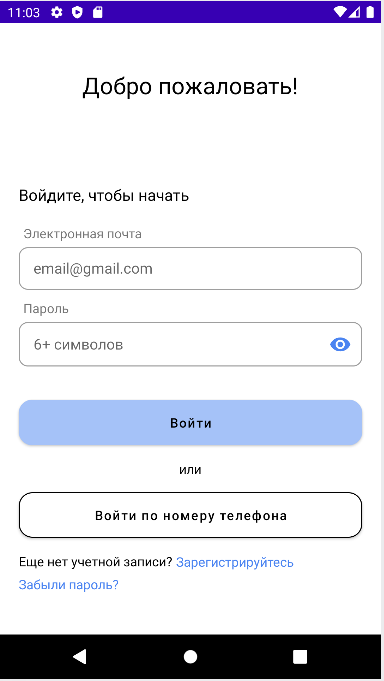


Рисунок Ж.4 – Тестирование на Pixel

**ПРИЛОЖЕНИЕ И**

**(обязательное)**

**Листинг**

class SplashActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_splash)

val handler = Handler(Looper.getMainLooper())

handler.postDelayed({

val intent = Intent(this, MainActivity::class.java)

startActivity(intent)

finish()

}, 3000)

}

}

Листинг И.1 – SplashActivity.kt

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

val binding = ActivityMainBinding.inflate(layoutInflater)

super.onCreate(savedInstanceState)

setContentView(binding.root)

binding.loginButton.setOnClickListener {

val intent = Intent(this, HomeActivity::class.java)

startActivity(intent)

finish()

}

**Продолжение приложения И**

binding.loginTextMain.setOnClickListener{

val intent = Intent(this, LoginActivity::class.java)

startActivity(intent)

finish()

}

}

}

Листинг И.2 – MainActivity.kt

class LoginActivity : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 val binding = ActivityLoginBinding.inflate(layoutInflater)  
 setContentView(binding.root)  
 binding.registerButtonLogin.setOnClickListener {val intent = Intent(this**,** RegisterActivity::class.java)  
 startActivity(intent)  
 }  
 binding.loginButtonLogin.setOnClickListener {  
 verificationLogin(binding)  
 }  
}  
 private fun verificationLogin(binding: ActivityLoginBinding) {  
 val email = binding.emailEditLogin  
 val password = binding.passwordEditLogin  
 val emailInputLayout = binding.emailInputLogin  
 val passwordInputLayout = binding.passwordInputLogin  
 val validate = Validator(this)

**Продолжение приложения И**

binding.emailInputLogin.error = validate.validateEmail(email)  
 passwordInputLayout.error = validate.validatePassword(password)

if (emailInputLayout.error == null &&  
 passwordInputLayout.error == null  
 ) {  
 val intent = Intent(this**,** HomeActivity::class.java)  
 intent.putExtra("Name"**,** email.text.toString())  
 startActivity(intent)  
 finish()  
 }  
 }  
}

Листинг И.3 – LoginActivity.kt

class RegisterActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

val binding = ActivityRegisterBinding.inflate(layoutInflater)

setContentView(binding.root)

binding.loginButtonRegister.setOnClickListener{

val intent = Intent(this, LoginActivity::class.java)

startActivity(intent)

}

binding.registerButtonRegister.setOnClickListener {

verificationRegister(binding)

}

}

**Продолжение приложения И**

private fun verificationRegister(binding: ActivityRegisterBinding) {

val emailInputLayout = binding.emailInputRegister

val passwordInputLayout = binding.passwordInputRegister

val usernameInputLayout = binding.userInputRegister

val phoneInputLayout = binding.phoneInputRegister

val validate = Validator(this)

usernameInputLayout.error =

passwordInputLayout.error = validate.validatePassword(binding.passwordEditRegister)

phoneInputLayout.error = validate.validatePhone(binding.phoneEditRegister)

if (usernameInputLayout.error == null &&

emailInputLayout.error == null &&

passwordInputLayout.error == null &&

phoneInputLayout.error == null

) {

val intent = Intent(this, HomeActivity::class.java)

intent.putExtra("Name", binding.userEditRegister.text.toString())

intent.putExtra("Email", binding.emailEditRegister.text.toString())

intent.putExtra("Phone", binding.phoneEditRegister.text.toString())

startActivity(intent)

finish()

}

}

Листинг И.4 – RegisterActivity.kt

**Продолжение приложения И**

class HomeActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

val binding = ActivityHomeBinding.inflate(layoutInflater)

setContentView(binding.root)

binding.bottomNavigation.setOnItemSelectedListener { menuItem ->

when (menuItem.itemId) {

R.id.homeNavigation -> supportBottomNavigation(HomeFragment())

R.id.orderNavigation -> supportBottomNavigation(OrderFragment())

R.id.likedNavigation -> supportBottomNavigation(LikedFragment())

R.id.messageNavigation -> supportBottomNavigation(MessageFragment())

R.id.profileNavigation -> supportBottomNavigation(ProfileFragment())

}

true

}

}

private fun supportBottomNavigation(fragment: Fragment) {

supportFragmentManager

.beginTransaction()

.replace(R.id.fragmentContainerView, fragment)

.commit()

}

}

Листинг И.5 – HomeActivity.kt

**Продолжение приложения И**

class HomeFragment : Fragment() {

lateinit var binding: FragmentHomeBinding

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

binding = FragmentHomeBinding.inflate(inflater, container, false)

binding.everymonthCard.setOnClickListener {}

binding.everydayCard.setOnClickListener {}

binding.apply {

binding.calendarButton.setOnClickListener {

openDataPickerDialog()

}

binding.selectNightNumber.setOnClickListener {

val supportFragmentManager = requireActivity().supportFragmentManager

val hotelSelectNightDialog = HotelSelectNightDialog(this)

hotelSelectNightDialog.show(supportFragmentManager, "myDialog")

}

val hotels = listOf(

Hotel(

imageView = R.drawable.hotel\_one\_image,

nameCity = "Хилтон",

availableHotel = "Доступно 12 номеров",

**Продолжение приложения И**

ratingHotel = "4.8"

),

Hotel(

imageView = R.drawable.hotel\_two\_image,

nameCity = "Меркурий",

availableHotel = "Доступно 5 номеров",

ratingHotel = "4.9"

),

Hotel(

imageView = R.drawable.hotel\_three\_image,

nameCity = "Аура",

availableHotel = "Доступно 7 номеров",

ratingHotel = "4.9"

),

Hotel(

imageView = R.drawable.hotel\_four\_image,

nameCity = "Ambassador",

availableHotel = "Доступно 12 номеров",

ratingHotel = "4.9"

),

Hotel(

imageView = R.drawable.hotel\_five\_image,

nameCity = "Звездная",

availableHotel = "Доступно 9 номеров",

ratingHotel = "4.9"

)

)

val adapter = HotelsAdapter()

binding.recyclerHotels.adapter = adapter

adapter.submitList(hotels)

**Продолжение приложения И**

return binding.root

}

}

private fun openDataPickerDialog() {

val datePickerFragment = DatePickerDialog()

val supportFragmentManager = requireActivity().supportFragmentManager

supportFragmentManager.setFragmentResultListener(

"REQUEST\_KEY",

viewLifecycleOwner

) { resultKey, bundle ->

if (resultKey == "REQUEST\_KEY") {

val date = bundle.getString("SELECTED\_DATE")

binding.calendarButton.text = date

}

}

datePickerFragment.show(supportFragmentManager, "DatePickerFragment")

}

}

Листинг И.6 – HomeFragment.kt

class LikedFragment : Fragment() {

lateinit var binding: FragmentLikedBinding

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

**Продолжение приложения И**

): View {

binding = FragmentLikedBinding.inflate(inflater, container, false)

val hotelNumber = listOf(

HotelNumber(

imageHotelNumber = R.drawable.hotel\_card\_image,

nameNumber = "Меркурий",

ratingNumber = "4.9(808)",

addresHotel = "Саранск, Большевитская 45",

priceNumber = "135$ / ночь"

)

)

val adapter = HotelNumbersAdapter()

binding.hotelNumberRecyclerView.adapter = adapter

adapter.submitList(hotelNumber)

return binding.root

}

}

Листинг И.7 – LikedFragment.kt

class MessageFragment : Fragment() {

lateinit var binding: FragmentMessageBinding

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

binding = FragmentMessageBinding.inflate(inflater, container, false)

**Продолжение приложения И**

val adapter = ViewPagerAdapter(this)

binding.viewPagerMessage.adapter = adapter

TabLayoutMediator(binding.tabLayoutMessage, binding.viewPagerMessage) { tab, position ->

when (position) {

0 -> tab.text = getString(R.string.message\_text\_tablayout)

1 -> tab.text = getString(R.string.notification\_text\_tablayout)

}

}.attach()

return binding.root

}

}

Листинг И.8 – MessageFragment.kt

class ProfileFragment : Fragment() {

lateinit var binding: FragmentProfileBinding

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

binding = FragmentProfileBinding.inflate(inflater, container, false)

binding.usernameProfileText.text = activity?.intent?.extras?.getString("Name")

binding.emailProfile.text =

**Продолжение приложения И**

activity?.intent?.extras?.getString("Email")

binding.phoneProfile.text = activity?.intent?.extras?.getString("Phone")

binding.personalInfoCard.setOnClickListener {}

binding.passwordProfileCard.setOnClickListener {}

return binding.root

}

override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {

inflater.inflate(R.menu.profile\_navigation, menu)

}

override fun onOptionsItemSelected(item: MenuItem): Boolean {

when(item.itemId) {

R.id.exitMenu -> {

activity?.finish()

binding.phoneProfile.text = "dfssf"

}

return true

}

}

Листинг И.9 – ProfileFragment.kt

class Validator(private val context: Context) {

fun validateName(name: EditText): String? =

when {

name.text.toString().isBlank() -> context.getString(R.string.error\_empty)

name.length() <= ValidatorConst.USERNAME\_LENGTH -> context.getString(R.string.error\_username\_more4)

**Продолжение приложения И**

else -> null

}

fun validateEmail(email: EditText): String? =

when {

email.text.toString().isBlank() -> context.getString(R.string.error\_empty)

email.length() <= ValidatorConst.EMAIL\_LENGTH -> context.getString(R.string.error\_email\_more8)

!(email.text.toString().contains(ValidatorConst.EMAIL\_TRUE)) -> context.getString(R.string.error\_email\_true)

else -> null

}

fun validatePassword(password: EditText): String? =

when {

password.text.toString().isBlank() -> context.getString(R.string.error\_empty)

password.length() <= ValidatorConst.PASSWORD\_LENGTH -> context.getString(R.string.error\_email\_more8)

else -> null

}

fun validatePhone(phoneNumber: EditText): String? =

when {

phoneNumber.text.toString().isBlank() -> context.getString(R.string.error\_empty)

phoneNumber.length() < ValidatorConst.PHONE\_LENGTH -> context.getString(R.string.error\_password\_empty)

else -> null

}

Листинг И.10 – Validator.kt

**Продолжение приложения И**

class DatePickerDialog : DialogFragment(), DatePickerDialog.OnDateSetListener {

private val calendar = Calendar.getInstance()

override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

val year = calendar.get(Calendar.YEAR)

val month = calendar.get(Calendar.MONTH)

val day = calendar.get(Calendar.DAY\_OF\_MONTH)

return DatePickerDialog(requireActivity(), this, year, month, day)

}

override fun onDateSet(view: DatePicker?, year: Int, month: Int, dayOfMonth: Int) {

calendar.set(Calendar.YEAR, year)

calendar.set(Calendar.MONTH, month)

calendar.set(Calendar.DAY\_OF\_MONTH, dayOfMonth)

val selectedDate = SimpleDateFormat("dd-MM-yyyy", Locale.ENGLISH).format(calendar.time)

val selectedDateBundle = Bundle()

selectedDateBundle.putString("SELECTED\_DATE", selectedDate)

setFragmentResult("REQUEST\_KEY", selectedDateBundle)

}

}

Листинг И.11 – DataPickerDialog.kt

class HotelSelectNightDialog(context: FragmentHomeBinding) : DialogFragment() {

**Продолжение приложения И**

val act = context

private val nights = arrayOf(

"1 ночь","2 ночи","3 ночи","4 ночи","5 ночей","6 ночей", "Неделя","2 недели","3 недели","Месяц")

override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

return activity?.let {

val builder = AlertDialog.Builder(it)

builder.setTitle("Выберите количество ночей")

.setItems(nights

) { \_, which ->

act.selectNightNumber.text = nights[which]

}

return builder.create()

} ?: throw IllegalStateException("Activity cannot be null")

}

}

Листинг И.12 – HotelSelectNightDialog.kt

data class Hotel(

val imageView: Int,

val nameCity: String,

val availableHotel: String,

val ratingHotel: String

)

Листинг И.13 – Hotel.kt

**Продолжение приложения И**

data class HotelNumber(

val imageHotelNumber: Int,

val nameNumber: String,

val ratingNumber: String,

val addresHotel: String,

val priceNumber: String

)

Листинг И.14 – HotelNumber.kt

class HotelNumbersAdapter : RecyclerView.Adapter<HotelNumbersViewAdapter>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HotelNumbersViewAdapter {

return HotelNumbersViewAdapter(ItemHotelLikedCardBinding.inflate(LayoutInflater.from(parent.context), parent, false))

}

override fun onBindViewHolder(holder: HotelNumbersViewAdapter, position: Int) {

holder.bind(items[position])}

override fun getItemCount() = items.size

fun submitList(hotelNumber: List<HotelNumber>) {

items.clear()

items.addAll(hotelNumber)

notifyDataSetChanged()

}

}

Листинг И.15 – HotelNumber.kt

**Продолжение приложения И**

class HotelNumbersViewAdapter(private val binding: ItemHotelLikedCardBinding) : RecyclerView.ViewHolder(binding.root) {

fun bind(hotelNumber: HotelNumber) = with(binding) {

hotelNumberImage.setImageResource(hotelNumber.imageHotelNumber)

nameNumber.text = hotelNumber.nameNumber

ratingNumber.text = hotelNumber.ratingNumber

addresHotel.text = hotelNumber.addresHotel

priceNumber.text = hotelNumber.priceNumber

root.setOnClickListener{}

}

}

Листинг И.16 – HotelNumberViewAdapter.kt

class HotelsViewAdapter(private val binding: ItemHotelsBinding) : RecyclerView.ViewHolder(binding.root) {

fun bind(hotel: Hotel) = with(binding) {

imageHotel.setImageResource(hotel.imageView)

nameCityText.text = hotel.nameCity

availableHotelsText.text = hotel.availableHotel

ratingHotels.text = hotel.ratingHotel

root.setOnClickListener{}

}

}

Листинг И.17 – HotelsViewAdapter.kt

class ViewPagerAdapter(fragment: MessageFragment) : FragmentStateAdapter(fragment) {

override fun getItemCount() = 2

**Окончание приложения И**

override fun createFragment(position: Int): Fragment {

return if(position == 0) IncomingFragment()

else NotificationFragment()

}

}

Листинг И.18 – ViewPagerAdapter.kt

class HotelsAdapter : RecyclerView.Adapter<HotelsViewAdapter>() {

val items = mutableListOf<Hotel>()

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HotelsViewAdapter {

return HotelsViewAdapter(ItemHotelsBinding.inflate(LayoutInflater.from(parent.context), parent, false))

}

override fun onBindViewHolder(holder: HotelsViewAdapter, position: Int) {

holder.bind(items[position])

}

override fun getItemCount() = items.size

fun submitList(hotel: List<Hotel>) {

items.clear()

items.addAll(hotel)

notifyDataSetChanged()

}

}

Листинг И.19 – HotelAdapter.kt