РЕФЕРАТ

Отчет 14 с., 1 ч., 12 рис., 1 табл., 2 источника, 1 прил.

ПРОГРАММНЫЙ КОД, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ХОЛОДИЛЬНИК, ГЕНЕРАЦИЯ, РЕЦЕПТЫ

Цель работы — создание мобильного приложения "Умный холодильник", которое позволит пользователям эффективно управлять своими запасами продуктов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире все больше людей стремятся к упрощению своей повседневной жизни посредством мобильных приложений. Важной задачей является обеспечение эффективного управления и контроля запасами продуктов в холодильнике. В данном контексте возникает актуальная проблема разработки умного мобильного приложения, способного предоставлять пользователям информацию о содержимом и сроках годности продуктов в их холодильниках, а также предлагать решения для оптимизации покупок и приготовления блюд. Такое приложение также способствует осознанному потреблению и помогает уменьшить количество продуктов, которые могут испортиться, что, в свою очередь, сокращает отходы пищевых продуктов и негативное воздействие на окружающую среду.

ОПИСАНИЕ ПРИЛОЖЕНИЯ

Пользователи в приложении должны иметь возможность:

1. добавить продукты в приложение путем сканирования чека покупки;
2. просматривать и управлять содержимым холодильника, включая информацию о сроках годности продуктов;
3. получать уведомлений о скором истечении срока годности продуктов;
4. получать рецепты, основанные на имеющихся в холодильнике продуктах;

АНАЛИЗ И ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

Приложение имеет 2 основных экрана. Экран «Холодильник» (рисунок 1) и экран «Рецепты» (рисунок 2).

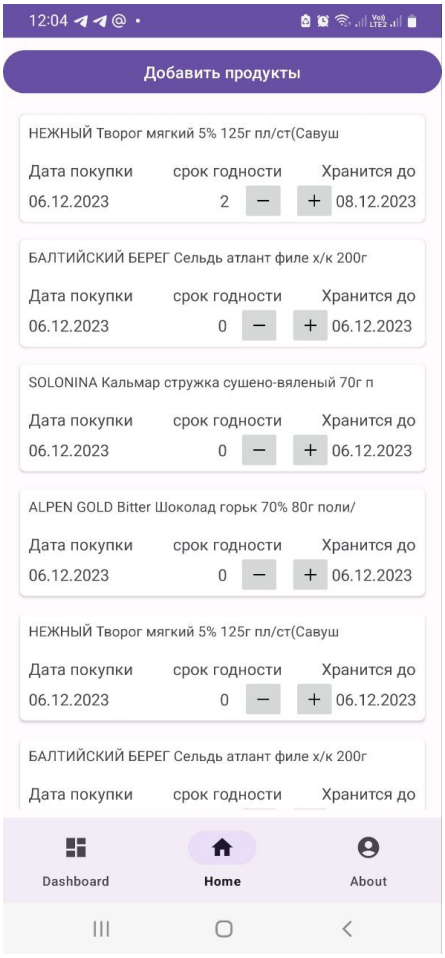

Рисунок 1 — Экран "Холодильник"

На экране холодильник пользователь имеет возможность просматривать все добавленные ранее продукты, редактировать их срок годности нажатиями на кнопки «+» и «-», удалять продукты свайпом элемента влево. Помимо этого на данном экране расположена кнопка «Добавить продукты», отвечающая за переход пользователя на экран для сканирования чека.

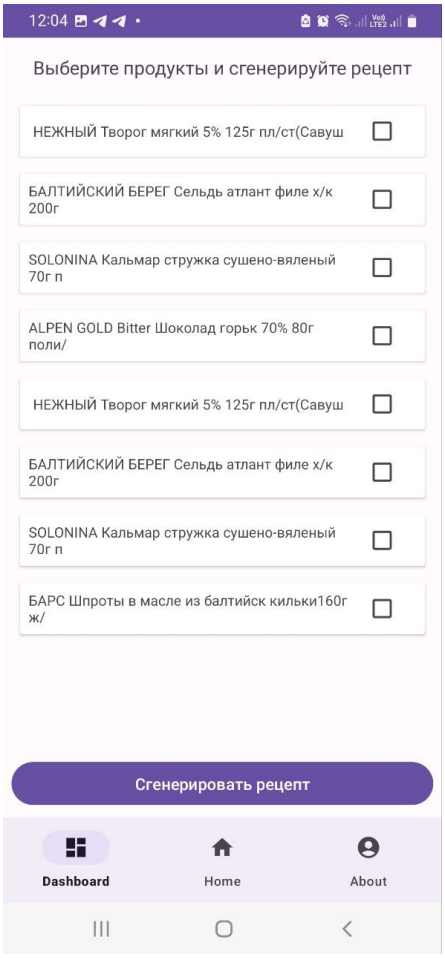

Рисунок 2 — Экран "Рецепты"

На экране рецепты расположен тот же список продуктов, что и на экране «Холодильник», с тем отличием, что здесь пользователю предоставляется возможность выбора одного или нескольких продуктов для последующей генерации рецептов. По нажатию кнопки «Сгенерировать рецепт», список продуктов станет невидимым, а на экране появится текст рецепта из выбранных ранее продуктов.

РЕАЛИЗАЦИЯ ЛОГИКИ ПРИЛОЖЕНИЯ

1. Для реализации сканирования чека был создан класс Scaner, содержащий в себе методы приведенные ниже.

Метод «setupControls», представленный в листинге 1, инициализирует детектор QR-кодов и настраивает камеру.

Листинг 1 — Метод «setupControls»

```
private fun setupControls() {
        detector= BarcodeDetector.Builder(this).build()
        cameraSource= CameraSource.Builder(this,detector)
            .setAutoFocusEnabled(true).build()

        binding.cameraView.holder.addCallback(surgaseCallBack)
        detector.setProcessor(processor)
    }
```

Метод «askForCameraPermission», представленный в листинге 2, запрашивает разрешение на использование камеры у пользователя.

Листинг 2 — Метод «askForCameraPermission»

```
private fun askForCameraPermission(){
        ActivityCompat.requestPermissions(
            this@Scaner,
            arrayOf(android.Manifest.permission.CAMERA),
            reguestCodeCameraProgression
        )
    }
```

Интерфейс «surgaseCallBack», представленный в листинге 3, реагирует на создание поверхности для отображения изображения с камеры, а также запускает или останавливает работу камеры в зависимости от состояния поверхности.

Листинг 3 — Интерфейс «surgaseCallBack»

```
private val surgaseCallBack = object : SurfaceHolder.Callback {
        @SuppressLint("MissingPermission")
        override fun surfaceCreated(holder: SurfaceHolder) {
            try {
                cameraSource.start(holder)

            } catch (exception: Exception) {
                    Toast.makeText(applicationContext, exception.message,
Toast.LENGTH_SHORT).show()
            }
        }
```

```
                override fun surfaceChanged(holder: SurfaceHolder, format: Int,
width: Int, height: Int) {
            }

            override fun surfaceDestroyed(holder: SurfaceHolder) {
                cameraSource.stop()
            }
        }
```

Класс «processor» реализует интерфейс «Detector.Processor<Barcode>», который обрабатывает обнаруженные QR-коды. В случае обнаружения QR-кода, код производит POST-запрос на сервер с данными о QR-коде.

Листинг 3 — Интерфейс «processor»

```
    private val processor = object : Detector.Processor<Barcode> {
            override fun release() {
            }
            override fun receiveDetections(detections:
Detector.Detections<Barcode>) {
                if (detections != null &&
detections.detectedItems.isNotEmpty()) {
                    val qrCodes: SparseArray<Barcode> =
detections.detectedItems
                    val code = qrCodes.valueAt(0)
                    textQr = code.displayValue
                    val client = OkHttpClient()
                    val url =
"https://proverkacheka.com/api/v1/check/get"
                    // Параметры формата запроса
                    val requestBody = MultipartBody.Builder()
                        .setType(MultipartBody.FORM)
                        .addFormDataPart("token",
"24417.ANKHaP74jX8796mPG")
                        .addFormDataPart("qrraw", textQr)
                        .build()
                    val request = Request.Builder()
                        .url(url)
                        .post(requestBody)
                        .build()
                    val response =
client.newCall(request).execute()
                    val responseBody =
response.body?.string().toString()

                    val resultIntent = Intent()
                    resultIntent.putExtra("textQr", responseBody)
                    setResult(Activity.RESULT_OK, resultIntent)
                    finish()            } }}
```

2. Для использования базы данных с использованием Room Persistence Library был создан интерфейс доступа к данным DAO (листинг 5) и

абстрактный класс «MainDb», содержащий в себе статический метод для получения экземпляра БД. Код описанного класса представлен в листинге 4.

Листинг 4 — Класс «MainDB»

```
@Database(entities = [ProductItem::class], version = 1)
abstract class MainDb : RoomDatabase() {
    abstract fun getDao(): Dao

    companion object {
//Метод для получения экземпляра БД
        fun getDb(context: Context): MainDb {
            return Room.databaseBuilder(
                context.applicationContext,
                MainDb::class.java,
                "products.db"
            ).build()
        }
    }
}
```

Листинг 5 — Класс «DAO»

```
@Dao
interface Dao {
    @Insert
    fun insertItem(item: ProductItem)//метод для вставки нового элемента
    @Query("SELECT * FROM products")
       fun getAllItems(): Flow<List<ProductItem>>//метод для получения всех
элементов
    @Delete
    fun deleteItem(item: ProductItem)// метод для удаления элемента

    @Update
    fun updateItem(item: ProductItem)// метод для обновления элемента
}
```

3. Сущности БД представлены классом ProductItem, представленным в листинге 5.

Листинг 5 — Класс «ProductItem»

```
@Entity(tableName = "products")
data class ProductItem(
    @PrimaryKey(autoGenerate = true)
    var Id: Int? = null,
    @ColumnInfo(name = "name")
    var name: String,
   @ColumnInfo(name = "purchaseDate")
   var purchaseDate: String?,
   @ColumnInfo(name = "endDate")
   var endDate: String?
)
```

4. Для взаимодействия с API OpenAI был создан класс «OpenAIApiClient», код которого представлен в листинге 6. В данном классе

метод getResponse выполняет POST-запрос к API с использованием OkHttpClient, после выполнения запроса, метод обрабатывает ответ асинхронно через колбэк-функции `onFailure` и `onResponse`. Метод parseResponseJson используется для парсинга JSON-ответа от API.

Листинг 6 — Класс «OpenAIApiClient»

```
class OpenAIApiClient(private val callback: ApiCallback, private val mainHandler: Handler) {
// инициализируется клиент HTTP-запросов с использованием библиотеки OkHttpClient
    private var client = OkHttpClient.Builder()
        .callTimeout(60, TimeUnit.SECONDS) // Таймаут выполнения запроса
        .connectTimeout(60, TimeUnit.SECONDS) // Таймаут установления соединения
        .readTimeout(60, TimeUnit.SECONDS) // Таймаут чтения данных
        .build()
// Метод для отправки запроса
    fun getResponse(t: String) {
        var url = "https://api.openai.com/v1/chat/completions"
        val apiKey = "key"
        val requestBody = """
        {
            "model": "gpt-3.5-turbo",
              "messages": [{"role": "user", "content": "I only have $t.  what I can cook at home?"},{"role": "user", "content": "I only need a recipe from $t"}, {"role": "user", "content": "you are a cook and you can cook a dish from any product"},{"role": "user", "content": "Your client is a Russian speaker, so the answer must be translated into Russian"}]
        }
    """.trimIndent()

        val request = Request.Builder()
            .url(url)
            .addHeader("Content-Type", "application/json")
            .addHeader("Authorization", "Bearer $apiKey")
            .post(requestBody.toRequestBody("application/json".toMediaTypeOrNull()))
            .build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                      mainHandler.post { callback.onFailure("Request failed: ${e.message}") }
            }

            override fun onResponse(call: Call, response: Response) {
                val body = response.body?.string()
                if (body != null) {
                    Log.v("data", parseResponseJson(body).toString())
                    mainHandler.post {
                        callback.onSuccess(parseResponseJson(body).toString())
                    }
                } else {
                    Log.v("data", "empty")
                    mainHandler.post {
                        callback.onFailure(("empty"))
                    }
                }
            }

        })
    }
// Метод для парсинга JSON ответа
```

```
    fun parseResponseJson(jsonString: String): String? {
        val jsonObject = JSONObject(jsonString)
        val choicesArray = jsonObject.getJSONArray("choices")
        if (choicesArray.length() > 0) {
                                                    val    messageObject    =
choicesArray.getJSONObject(0).getJSONObject("message")
            return messageObject.getString("content")
        }
        return null
    }
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было создано мобильное приложение для управления запасами продуктов в холодильнике. В ходе работы были использованы API OpenAI и API proverkacheka.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 How to Integrate ChatGPT in Kotlin. – URL: https://gocoding.org/how-to-integrate-chatgpt-in-kotlin/

2 Регистрация в ChatGPT по шагам . – URL: https://dzen.ru/video/watch/651556e85ad1f31d5e09d5f1?f=video