Содержание

[Введение 3](#_Toc201945859)

[1. Анализ требований 5](#_Toc201945860)

[2.Проектирование интерфейса 9](#_Toc201945861)

[2.1 Дизайн - система 9](#_Toc201945862)

[2.2 Пользовательские потоки 9](#_Toc201945863)

[2.3 Структура экранов и компонентов 11](#_Toc201945864)

[3. Технологический стек 12](#_Toc201945865)

[Заключение 13](#_Toc201945866)

[Приложения 14](#_Toc201945872)

[Приложение А 14](#_Toc201945873)

[Приложение Б 20](#_Toc201945876)

# Введение

TastyFood – это современное мобильное приложение для Android, созданное для любителей кулинарии и домашней готовки. Приложение предлагает обширную коллекцию рецептов с пошаговыми инструкциями и помогает пользователям легко находить идеи для вкусных блюд. В основе приложения лежит многослойная архитектура, которая обеспечивает четкое разделение ответственности между компонентами и позволяет системе масштабироваться под растущие потребности пользователей.

Актуальность проекта обусловлена растущим спросом на цифровые решения в кулинарии, которые помогают пользователям быстро находить интересные рецепты, планировать питание и сокращать время на приготовление пищи.

Целью стало приобретение навыков проектирования, разработки и тестирования мобильных приложений, а также изучение современных технологий и инструментов, таких как **Kotlin/Java (для Android).**

**Достижению поставленной цели способствует решение следующих задач:**

1. Провести анализ существующих аналогов и определить ключевые функции приложения.
2. Разработать техническое задание, включая требования к функционалу и дизайну.
3. Спроектировать пользовательский интерфейс (UI/UX) с учетом удобства и эргономики.
4. Реализовать основные функции приложения:

* Просмотр рецептов с категориями и фильтрами.
* Возможность сохранения рецептов в избранное.
* Добавление пользовательских рецептов.

1. Настроить хранение данных (локальная база данных или облачное решение, например, Firebase).
2. Протестировать приложение на различных устройствах и устранить выявленные ошибки.
3. Подготовить документацию и отчет по результатам разработки.

Анализ целевой аудитории приложения Tasty Food:

1. Демографические характеристики:

* Возраст: 18-55 лет (основная активная аудитория 25-45 лет)
* Пол: преимущественно женская аудитория (70%), но с растущим процентом мужчин (30%)
* География: городские жители с доходом средний и выше среднего

1. Поведенческие особенности:
2. Кулинарные навыки:

* Начинающие повара (35%)
* Опытные домашние кулинары (50%)
* Профессионалы (15%)

1. Частота готовки:

* Ежедневно (40%)
* 2-3 раза в неделю (45%)
* По выходным (15%)

1. Мотивация:

* Желание разнообразить рацион
* Поиск здоровых/диетических рецептов
* Экономия времени на приготовление пищи

В отчете представлены этапы работы: анализ требований, проектирование интерфейса, реализация функционала, тестирование и оптимизация приложения.

Результатом стал рабочий прототип приложения, позволяющий пользователям просматривать, сохранять в избранное и добавлять рецепты.

# Анализ требований

В процессе разработки мобильного приложения с рецептами были проанализированы ключевые требования к архитектуре, интерфейсу, хранению данных, сетевому взаимодействию и другим аспектам.

1.Архитектура:

1. Clean Architecture:
2. Приложение построено по принципам Clean Architecture с разделением на слои:

* Data (источники данных, репозитории)
* Domain (бизнес-логика, use cases)
* Presentation (UI, ViewModel)

1. MVVM/MVI/MVP:
2. Выбран MVVM (Model-View-ViewModel) как наиболее подходящий паттерн для Android-разработки.
3. ViewModel управляет состоянием экранов и взаимодействует с UseCases.
4. LiveData/StateFlow используются для реактивного обновления UI.
5. UI/UX:
6. Material Design 3:
7. Интерфейс соответствует гайдлайнам Material Design 3 (динамические цвета, типографика, компоненты).
8. Учтены принципы доступности (контрастность, размеры элементов).
9. Jetpack Compose:
10. UI реализован на Jetpack Compose для декларативного и гибкого дизайна.
11. Использованы модификаторы, LazyColumn/LazyRow для оптимизации списков.
12. Темная/светлая тема:
13. Поддержка темной и светлой темы через MaterialTheme и Dynamic Colors.
14. Сохранение выбранной темы в DataStore/SharedPreferences.
15. Анимации:
16. Добавлены анимации переходов между экранами (AnimatedVisibility, Crossfade).
17. Микровзаимодействия (например, лайки, загрузка)
18. Хранение данных:
19. Room Database:
20. Локальное хранилище реализовано на Room (SQLite-обертка от Google).
21. DAO для операций с рецептами и избранным.
22. Кэширование:
23. Сетевые ответы кэшируются в Room для работы в offline-режиме.
24. Paging 3 для постепенной загрузки и кэширования данных.
25. Миграции:
26. Поддержка миграций схемы БД через Room Migration.
27. Сеть:
28. Retrofit + OkHttp:
29. Сетевые запросы выполняются через Retrofit 2.
30. OkHttp для логирования и кэширования.
31. Обработка ошибок:
32. Try/Catch + Sealed Class/Result для обработки ошибок.
33. Отображение состояния загрузки/ошибки в UI.
34. Offline-режим:
35. При отсутствии интернета данные загружаются из Room.
36. NetworkBoundResource для гибридного подхода (сеть + кэш).
37. Асинхронность:
38. Корутины + Flow:
39. Все асинхронные операции выполняются через Coroutines (Dispatchers.IO/Default).
40. Flow/StateFlow для передачи данных между слоями.
41. Обработка состояний:
42. Sealed Class (Loading, Success, Error) для управления UI-состояниями.
43. Dependency Injection:
44. Hilt:
45. Внедрение зависимостей через Hilt (упрощает управление ViewModel, репозиториями).
46. Разделение на модули:

* AppModule (общие зависимости)
* NetworkModule (Retrofit)
* DatabaseModule (Room)

1. Scopes (@Singleton, @ActivityScoped) для контроля времени жизни объектов.
2. Тестирование:
3. Unit-тесты (JUnit 5 + MockK):
4. Тестирование UseCases, ViewModel, Repository.

Мокирование зависимостей через MockK.

1. UI-тесты (Compose Testing):
2. Тесты на взаимодействие с интерфейсом (ComposeTestRule).
3. Интеграционные тесты:
4. Проверка совместной работы компонентов (например, Room + Retrofit).
5. Code Coverage:
6. Отслеживание покрытия кода через JaCoCo.
7. CI/CD:
8. GitHub Actions:
9. Автоматизация сборки и тестирования при пулл-реквестах.
10. Fastlane:
11. Автоматизация публикации в Google Play.
12. Firebase App Distribution:
13. Тестовые сборки для раннего тестирования.
14. Release Management:
15. Версионирование через SemVer.
16. Челндж-логи в GitHub Releases.

# 2.Проектирование интерфейса

## Дизайн – система

Дизайн-система **Tasty Food** создана на основе принципов **Material Design 3**с кастомной аппетитной цветовой палитрой, выдержаной в спокойных, теплых оттенках, что способствует уютному и аппетитному восприятию приложения. Такой выбор цветов помогает создать атмосферу домашней кухни и приглашает пользователя к приготовлению пищи (рисунок1). 

Рисунок 1 - Цветовая гамма

## Пользовательские потоки

Пользовательские потоки в приложении TastyFood спроектированы с учетом принципов UX-дизайна для обеспечения интуитивного и эффективного взаимодействия. Основной пользовательский сценарий бронирования услуги включает следующие шаги (рисунок 2):

1. Запуск приложения - пользователь открывает приложение и видит splash screen с логотипом
2. Главный экран - пользователь видит поиск, категории, список из сетевых рецептов и кнопку добавления своего рецепта
3. Поиск - пользователь вводит название рецепта и он отображается в списке
4. Выбор категории - пользователь выбирает конкретную категорию и видит в списке рецепты только из этой категории
5. Кнопка загрузки сетевых рецептов - пользователь нажимает на кнопку и получает список сетевых рецептов
6. Список рецептов - пользователь выбирает нужный рецепт, может открыть его и добавить в избранное
7. Кнопка добавления своего рецепта - пользователь нажимает на кнопку и открывается окно добавления рецепта.

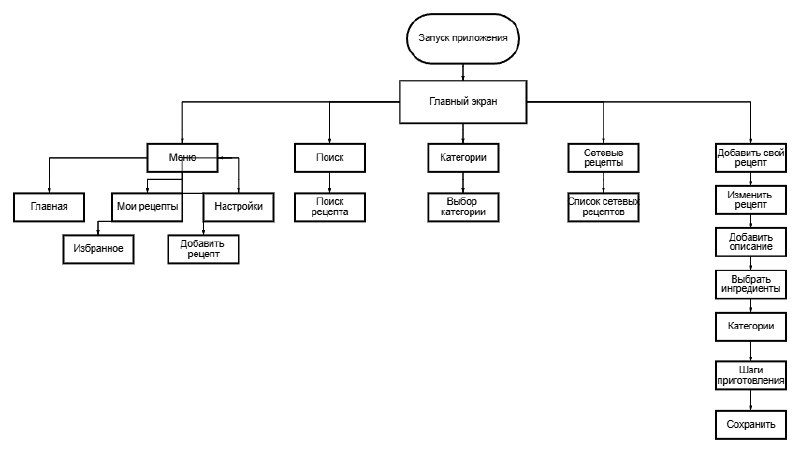


Рисунок 2 – Пользовательские потоки

Каждый шаг в этом потоке спроектирован для минимизации когнитивной нагрузки на пользователя и обеспечения четкого понимания текущего этапа процесса.

## Структура экранов и компонентов

Приложение Tasty Food построено по модульному принципу, где каждый экран и UI-компонент организованы в соответствии с ключевыми функциональными блоками системы (рисунок 3).

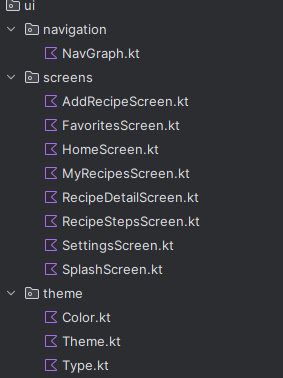


Рисунок 3 - Структура экранов и компонентов

# 3.Технологический стек

Приложение Tasty Food создано с применением актуальных технологий Android-разработки, что гарантирует высокую производительность, возможность масштабирования и простоту сопровождения (рисунок 4).

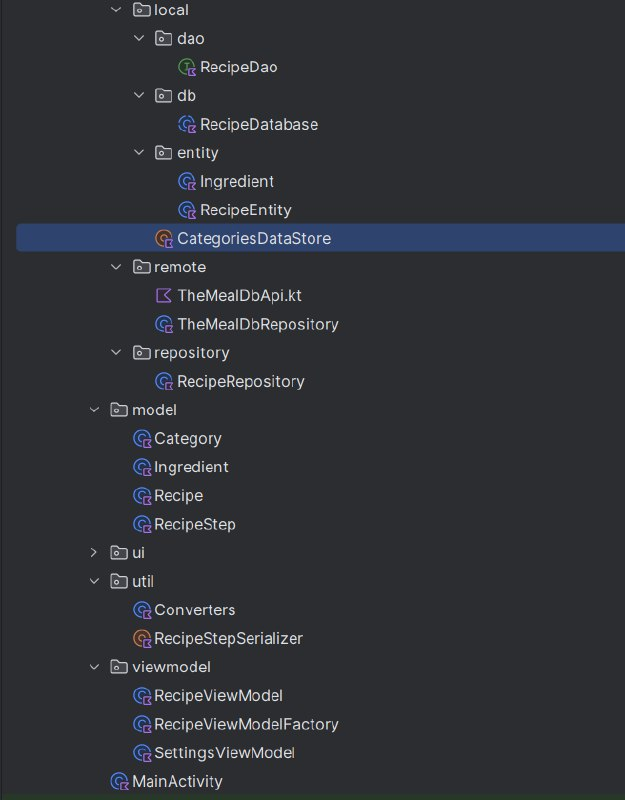


Рисунок 4 - Технологический стек

# Заключение

# Приложение Tasty Food — это современный и удобный помощник для всех, кто любит готовить. Оно сочетает в себе простой и приятный дизайн, полезные функции и понятный интерфейс.

# Главные преимущества:

# Создан уникальный дизайн на основе Material Design 3 с приятными цветами и фирменным стилем, который делает использование приложения комфортным и приятным.

# Приложение построено из отдельных модулей, что облегчает его развитие, поддержку и добавление новых функций.

# Используются современные технологии (Jetpack Compose, Kotlin Coroutines, Room), которые обеспечивают быструю и стабильную работу приложения.

Tasty Food помогает пользователям легко находить идеи для готовки, экономить время и получать удовольствие от процесса. Благодаря удобной архитектуре и продуманному дизайну приложение готово к дальнейшему развитию и добавлению новых возможностей.

# Приложения

# **Приложение А**

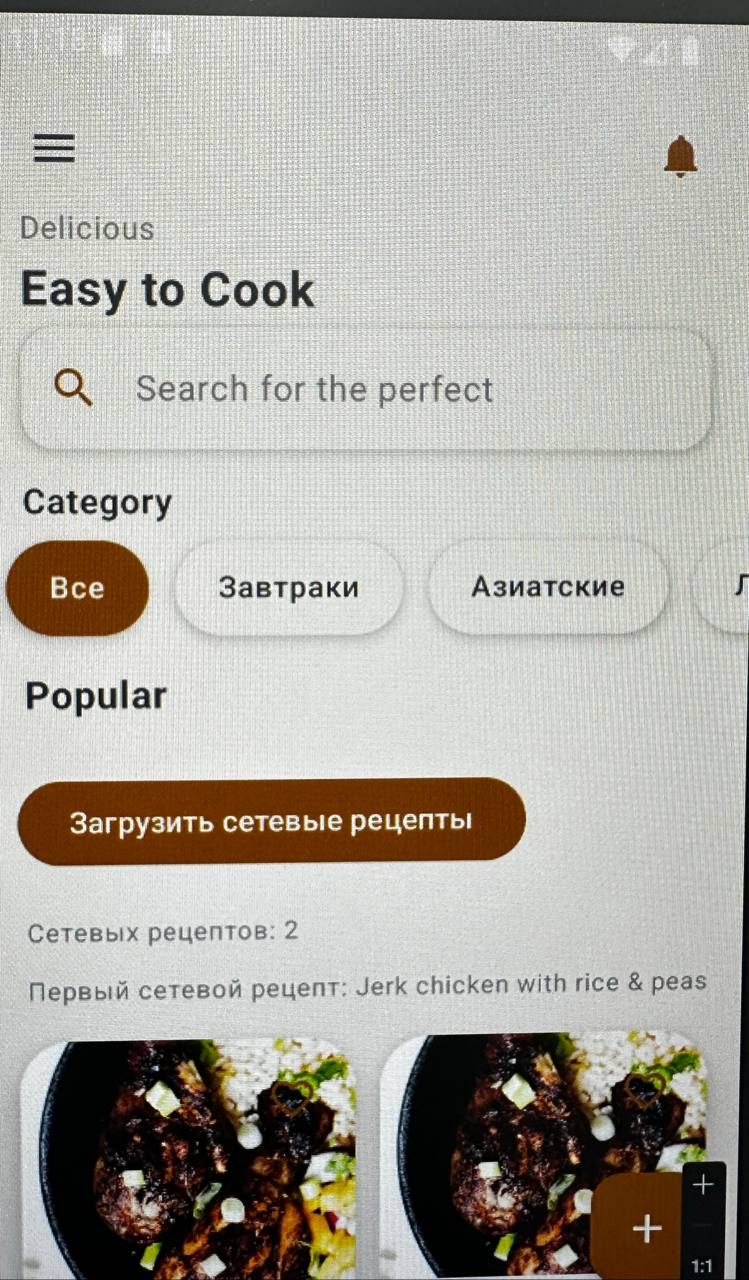


Рисунок А.1 - проверка сайта на выполнение

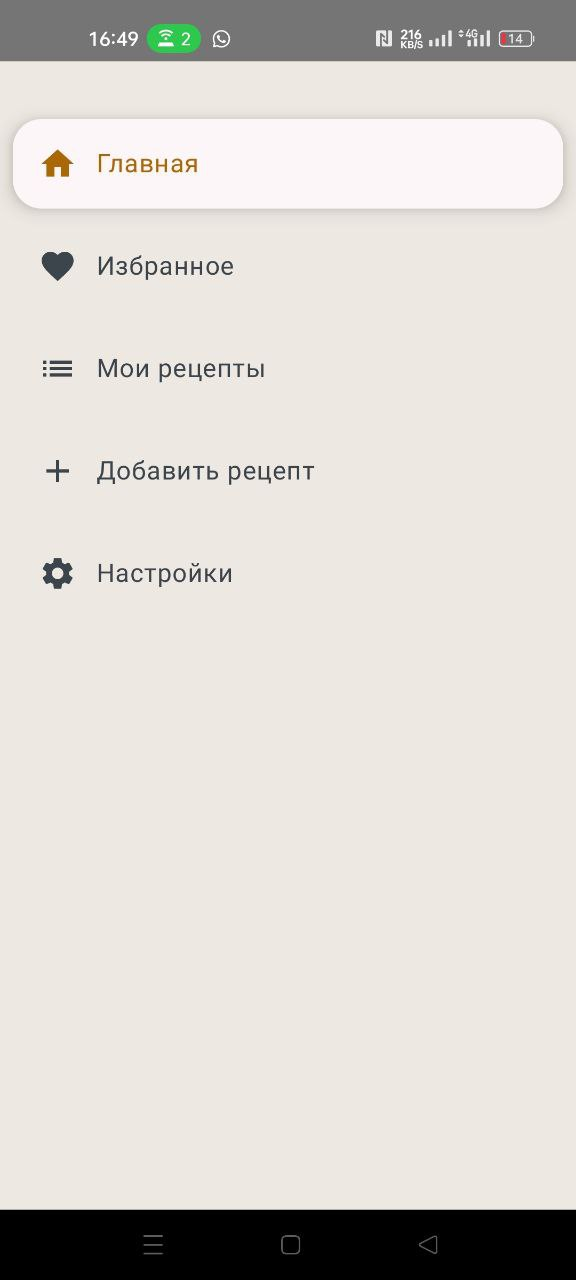


Рисунок А.2 - проверка сайта на выполнение



Рисунок А.3 - проверка сайта на выполнение



Рисунок А.4 - проверка сайта на выполнение

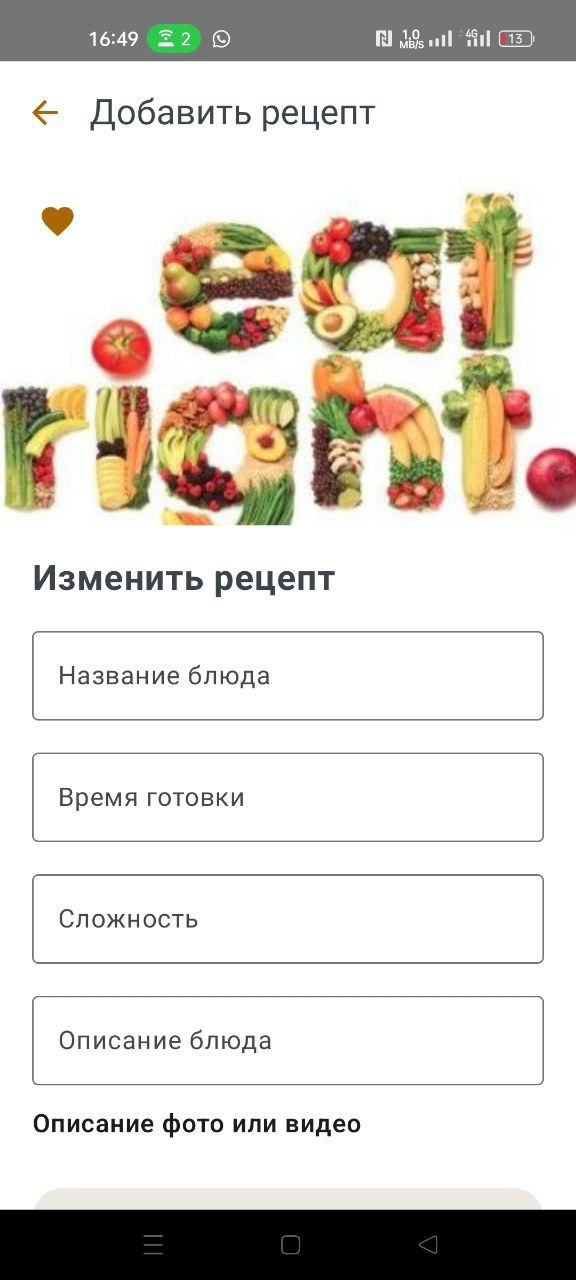


Рисунок А.5 - проверка сайта на выполнение

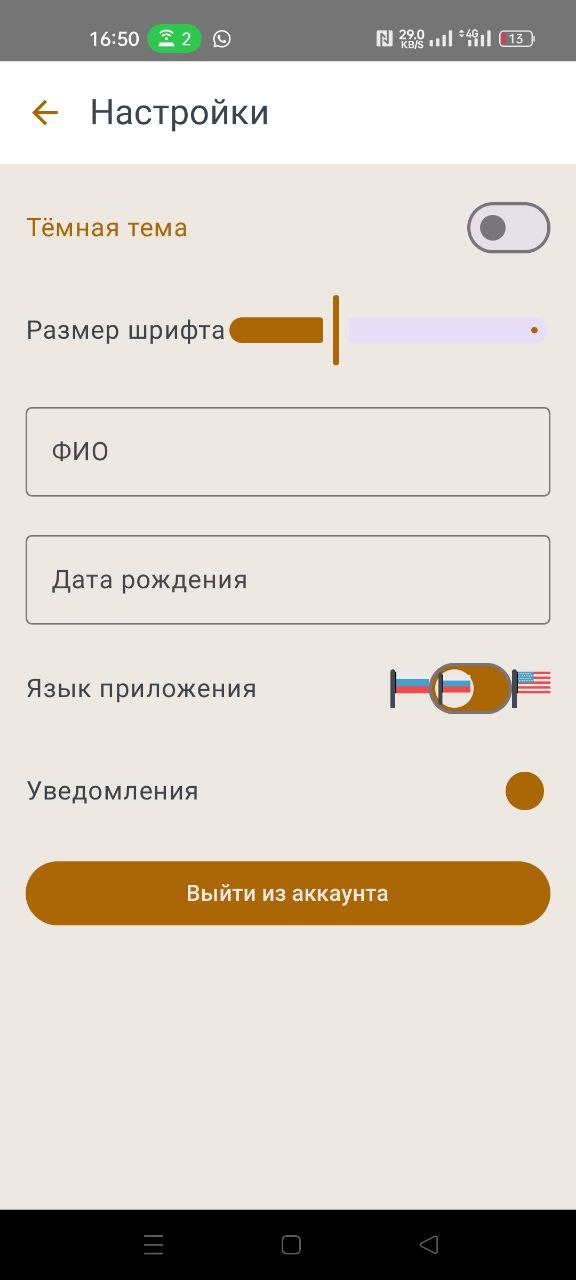


Рисунок А.6 - проверка сайта на выполнение

# **Приложение Б**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools">

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application

android:allowBackup="true"

android:dataExtractionRules="@xml/data\_extraction\_rules"

android:fullBackupContent="@xml/backup\_rules"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/Theme.TastyFood"

tools:targetApi="31">

<activity

android:name=".MainActivity"

android:exported="true"

android:label="@string/app\_name"

android:theme="@style/Theme.TastyFood">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

package com.students.tastyfood.data.local.dao

import androidx.room.\*

import com.students.tastyfood.data.local.entity.RecipeEntity

import kotlinx.coroutines.flow.Flow

@Dao

interface RecipeDao {

@Query("SELECT \* FROM recipes ORDER BY id DESC")

fun getAllRecipes(): Flow<List<RecipeEntity>>

@Query("SELECT \* FROM recipes WHERE isFavorite = 1")

fun getFavoriteRecipes(): Flow<List<RecipeEntity>>

@Query("SELECT \* FROM recipes WHERE id = :id")

suspend fun getRecipeById(id: Int): RecipeEntity?

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertRecipe(recipe: RecipeEntity)

@Delete

suspend fun deleteRecipe(recipe: RecipeEntity)

@Update

suspend fun updateRecipe(recipe: RecipeEntity)

@Query("DELETE FROM recipes")

suspend fun clearAll()

}

package com.students.tastyfood.data.local.db

import android.content.Context

import androidx.room.Database

import androidx.room.Room

import androidx.room.RoomDatabase

import androidx.room.TypeConverters

import com.students.tastyfood.data.local.dao.RecipeDao

import com.students.tastyfood.data.local.entity.RecipeEntity

import com.students.tastyfood.util.Converters

@Database(entities = [RecipeEntity::class], version = 3, exportSchema = false)

@TypeConverters(Converters::class)

abstract class RecipeDatabase : RoomDatabase() {

abstract fun recipeDao(): RecipeDao

companion object {

@Volatile

private var INSTANCE: RecipeDatabase? = null

fun getDatabase(context: Context): RecipeDatabase {

return INSTANCE ?: synchronized(this) {

val instance = Room.databaseBuilder(

context.applicationContext,

RecipeDatabase::class.java,

"recipe\_database"

).fallbackToDestructiveMigration(true).build()

INSTANCE = instance

instance

}

}

}

}

package com.students.tastyfood.data.local.entity

data class Ingredient(

val id: Int,

val name: String,

val imageUrl: String? = null,

val imageRes: Int? = null

)

package com.students.tastyfood.data.local.entity

import androidx.room.Entity

import androidx.room.PrimaryKey

@Entity(tableName = "recipes")

data class RecipeEntity(

@PrimaryKey(autoGenerate = true)

val id: Int = 0,

val title: String,

val imageUrl: String?,

val cookingTime: String,

val difficulty: Int, // 1..5

val description: String,

val rating: Float,

val isFavorite: Boolean = false,

val ingredients: List<String> = emptyList(),

val descriptionMedia: List<String> = emptyList(),

val category: String = "Все рецепты",

val steps: List<com.students.tastyfood.model.RecipeStep> = emptyList()

)package com.students.tastyfood.data.local.entity

import androidx.room.Entity

import androidx.room.PrimaryKey

@Entity(tableName = "recipes")

data class RecipeEntity(

@PrimaryKey(autoGenerate = true)

val id: Int = 0,

val title: String,

# 

val imageUrl: String?,

val cookingTime: String,

val difficulty: Int, // 1..5

val description: String,

val rating: Float,

val isFavorite: Boolean = false,

val ingredients: List<String> = emptyList(),

val descriptionMedia: List<String> = emptyList(),

val category: String = "Все рецепты",

val steps: List<com.students.tastyfood.model.RecipeStep> = emptyList()

)

package com.students.tastyfood.data.local

import android.content.Context

import androidx.datastore.preferences.core.edit

import androidx.datastore.preferences.core.stringSetPreferencesKey

import androidx.datastore.preferences.preferencesDataStore

import kotlinx.coroutines.flow.Flow

import kotlinx.coroutines.flow.map

private val Context.dataStore by preferencesDataStore(name = "categories")

object CategoriesDataStore {

private val CATEGORIES\_KEY = stringSetPreferencesKey("categories")

val defaultCategories = setOf("Все", "Завтраки", "Азиатские", "Ланч", "Обед", "Ужин", "Салаты", "Десерты", "Напитки", "Закуски", "Постные", "Вегетарианские", "Веганские", "Без глютена", "Без молока")

fun getCategories(context: Context): Flow<Set<String>> =

context.dataStore.data.map { prefs ->

prefs[CATEGORIES\_KEY] ?: defaultCategories

}

suspend fun addCategory(context: Context, category: String) {

# 

context.dataStore.edit { prefs ->

val current = prefs[CATEGORIES\_KEY]?.toMutableSet() ?: defaultCategories.toMutableSet()

current.add(category)

prefs[CATEGORIES\_KEY] = current

}

}

}

package com.students.tastyfood.data.remote

import retrofit2.http.GET

import retrofit2.http.Query

// Модель ответа для одного рецепта

data class MealResponse(

val meals: List<MealDto>?

)

data class MealDto(

val idMeal: String?,

val strMeal: String?,

val strCategory: String?,

val strArea: String?,

val strInstructions: String?,

val strMealThumb: String?,

val strTags: String?,

val strYoutube: String?

)

data class MealShortDto(

val idMeal: String?,

val strMeal: String?,

val strMealThumb: String?

)

data class MealShortResponse(

val meals: List<MealShortDto>?

)

interface TheMealDbApi {

// Получить случайный рецепт

@GET("random.php")

suspend fun getRandomMeal(): MealResponse

// Поиск по названию

@GET("search.php")

suspend fun searchMeals(@Query("s") query: String): MealResponse

// Получить по категории

@GET("filter.php")

suspend fun getMealsByCategory(@Query("c") category: String): MealShortResponse

// Получить подробный рецепт по id

@GET("lookup.php")

suspend fun lookupMealById(@Query("i") id: String): MealResponse

}

package com.students.tastyfood.data.remote

import android.util.Log

import retrofit2.Retrofit

import retrofit2.converter.gson.GsonConverterFactory

class TheMealDbRepository {

private val api: TheMealDbApi

init {

val retrofit = Retrofit.Builder()

.baseUrl("https://www.themealdb.com/api/json/v1/1/")

.addConverterFactory(GsonConverterFactory.create())

# 

.build()

api = retrofit.create(TheMealDbApi::class.java)

}

suspend fun getRandomMeal(): MealDto? {

return api.getRandomMeal().meals?.firstOrNull()

}

suspend fun searchMeals(query: String): List<MealDto> {

return api.searchMeals(query).meals ?: emptyList()

}

suspend fun getMealsByCategory(category: String): List<MealShortDto> {

return api.getMealsByCategory(category).meals ?: emptyList()

}

suspend fun getSomeMealsFromCategory(category: String, count: Int = 2): List<MealDto> {

val shortList = getMealsByCategory(category)

println("getSomeMealsFromCategory: shortList.size = ${shortList.size}")

Log.d("TheMealDbRepository", "Коротких рецептов по категории '$category': ${shortList.size}")

val selected = if (shortList.size > count) shortList.shuffled().take(count) else shortList

val result = mutableListOf<MealDto>()

for (shortMeal in selected) {

try {

println("lookupMealById: idMeal = ${shortMeal.idMeal}")

val detail = api.lookupMealById(shortMeal.idMeal ?: "").meals?.firstOrNull()

if (detail != null) result.add(detail)

} catch (e: Exception) {

Log.e("TheMealDbRepository", "Ошибка при получении подробного рецепта: ${shortMeal.idMeal}", e)

println("Ошибка при получении подробного рецепта: ${shortMeal.idMeal}, ${e.message}")

}

}

println("getSomeMealsFromCategory: result.size = ${result.size}")

Log.d("TheMealDbRepository", "Подробных рецептов получено: ${result.size}")

return result

}

override fun toString(): String = super.toString()

}

package com.students.tastyfood.data.repository

import com.students.tastyfood.data.local.dao.RecipeDao

import com.students.tastyfood.data.local.entity.RecipeEntity

import kotlinx.coroutines.flow.Flow

class RecipeRepository(private val recipeDao: RecipeDao) {

fun getAllRecipes(): Flow<List<RecipeEntity>> = recipeDao.getAllRecipes()

fun getFavoriteRecipes(): Flow<List<RecipeEntity>> = recipeDao.getFavoriteRecipes()

suspend fun getRecipeById(id: Int): RecipeEntity? = recipeDao.getRecipeById(id)

suspend fun insertRecipe(recipe: RecipeEntity) = recipeDao.insertRecipe(recipe)

suspend fun deleteRecipe(recipe: RecipeEntity) = recipeDao.deleteRecipe(recipe)

suspend fun updateRecipe(recipe: RecipeEntity) = recipeDao.updateRecipe(recipe)

suspend fun clearAll() = recipeDao.clearAll()

}

package com.students.tastyfood.model

data class Category(

val name: String,

val iconRes: Int? = null

)

package com.students.tastyfood.model

data class Ingredient(

val id: Int = 0,

val name: String,

val imageUrl: String? = null

)

package com.students.tastyfood.model

data class Recipe(

val id: Int = 0,

val title: String,

val description: String? = null,

val imageUrl: String? = null,

val ingredients: List<Ingredient> = emptyList(),

val steps: List<RecipeStep> = emptyList()

)

package com.students.tastyfood.model

data class RecipeStep(

val description: String,

val durationMinutes: Int = 5,

val imageUrl: String? = null

)

package com.students.tastyfood.ui.navigation

import androidx.compose.runtime.Composable

import androidx.compose.ui.platform.LocalContext

import androidx.lifecycle.viewmodel.compose.viewModel

import androidx.navigation.NavHostController

import androidx.navigation.compose.NavHost

import androidx.navigation.compose.composable

import androidx.navigation.compose.rememberNavController

import com.students.tastyfood.data.local.db.RecipeDatabase

import com.students.tastyfood.ui.screens.\*

import com.students.tastyfood.viewmodel.RecipeViewModel

import com.students.tastyfood.viewmodel.RecipeViewModelFactory

import com.students.tastyfood.viewmodel.SettingsViewModel

import androidx.compose.animation.AnimatedContent

import androidx.compose.animation.ExperimentalAnimationApi

import androidx.compose.animation.with

import androidx.compose.animation.slideInHorizontally

import androidx.compose.animation.slideOutHorizontally

import androidx.compose.animation.togetherWith

sealed class Screen(val route: String) {

object Splash : Screen("splash")

object Home : Screen("home")

object RecipeDetail : Screen("recipeDetail/{recipeId}")

object Favorites : Screen("favorites")

object MyRecipes : Screen("myrecipes")

object AddRecipe : Screen("addrecipe")

object Settings : Screen("settings")

object RecipeSteps : Screen("recipeSteps/{recipeId}/{stepId}")

}

@OptIn(ExperimentalAnimationApi::class)

@Composable

fun NavGraph(navController: NavHostController = rememberNavController(), settingsViewModel: SettingsViewModel) {

val context = LocalContext.current

val recipeDao = RecipeDatabase.getDatabase(context).recipeDao()

val recipeViewModelFactory = RecipeViewModelFactory(recipeDao)

val recipeViewModel: RecipeViewModel = viewModel(factory = recipeViewModelFactory)

NavHost(

navController = navController,

startDestination = Screen.Splash.route

) {

composable(Screen.Splash.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route) { \_ ->

SplashScreen(navController)

}

}

composable(Screen.Home.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route,

transitionSpec = {

slideInHorizontally { fullWidth -> fullWidth }.togetherWith(slideOutHorizontally { fullWidth -> -fullWidth })

}

) { \_ ->

HomeScreen(navController, recipeViewModel)

}

}

composable("recipeDetail/{recipeId}") { backStackEntry ->

val recipeId = backStackEntry.arguments?.getString("recipeId")?.toIntOrNull() ?: 0

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route,

transitionSpec = {

slideInHorizontally { fullWidth -> fullWidth }.togetherWith(slideOutHorizontally { fullWidth -> -fullWidth })

}

) { \_ ->

RecipeDetailScreen(navController, recipeId, recipeViewModel)

}

}

composable(Screen.Favorites.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route,

transitionSpec = {

slideInHorizontally { fullWidth -> fullWidth }.togetherWith(slideOutHorizontally { fullWidth -> -fullWidth })

}

) { \_ ->

FavoritesScreen(navController, recipeViewModel, onMenuClick = null)

}

}

composable(Screen.MyRecipes.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route,

transitionSpec = {

slideInHorizontally { fullWidth -> fullWidth }.togetherWith(slideOutHorizontally { fullWidth -> -fullWidth })

}

) { \_ ->

MyRecipesScreen(navController, recipeViewModel)

}

}

composable(Screen.AddRecipe.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route,

transitionSpec = {

slideInHorizontally { fullWidth -> fullWidth }.togetherWith(slideOutHorizontally { fullWidth -> -fullWidth })

}

) { \_ ->

AddRecipeScreen(navController, recipeViewModel)

}

}

composable(Screen.Settings.route) {

AnimatedContent(targetState = navController.currentBackStackEntry?.destination?.route) { \_ ->

SettingsScreen(navController, settingsViewModel = settingsViewModel)

}

}

composable("recipeSteps/{recipeId}/{stepId}") { backStackEntry ->

val recipeId = backStackEntry.arguments?.getString("recipeId")?.toIntOrNull() ?: 0

val stepId = backStackEntry.arguments?.getString("stepId")?.toIntOrNull() ?: 0

RecipeStepsScreen(navController, recipeId = recipeId, stepId = stepId, viewModel = recipeViewModel)

}

}

}

package com.students.tastyfood.ui.screens

import android.net.Uri

import androidx.activity.compose.rememberLauncherForActivityResult

import androidx.activity.result.contract.ActivityResultContracts

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.border

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.\*

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.LazyRow

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.shape.CircleShape

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.Favorite

import androidx.compose.material3.\*

import androidx.compose.runtime.\*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.clip

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.platform.LocalContext

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.navigation.NavController

import coil3.compose.rememberAsyncImagePainter

import coil3.request.ImageRequest

import com.students.tastyfood.data.local.entity.Ingredient

import com.students.tastyfood.data.local.entity.RecipeEntity

import com.students.tastyfood.R

import com.students.tastyfood.viewmodel.RecipeViewModel

import com.students.tastyfood.data.local.CategoriesDataStore

import kotlinx.coroutines.flow.collectLatest

import kotlinx.coroutines.launch

import java.io.File

import java.io.InputStream

import java.io.OutputStream

import android.content.Context

import androidx.compose.ui.graphics.Color.Companion.White

import androidx.compose.ui.res.stringResource

import com.students.tastyfood.model.RecipeStep

import com.students.tastyfood.util.RecipeStepSerializer

import com.students.tastyfood.ui.theme.\*

fun saveImageToInternalStorage(context: Context, uri: Uri): String? {

return try {

val inputStream: InputStream? = context.contentResolver.openInputStream(uri)

val fileName = "recipe\_${System.currentTimeMillis()}.jpg"

val file = File(context.filesDir, fileName)

val outputStream: OutputStream = file.outputStream()

inputStream?.copyTo(outputStream)

inputStream?.close()

outputStream.close()

file.absolutePath

} catch (\_: Exception) {

null

}

}

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun AddRecipeScreen(navController: NavController, viewModel: RecipeViewModel, onMenuClick: (() -> Unit)? = null) {

var title by remember { mutableStateOf("") }

var cookTime by remember { mutableStateOf("") }

var difficulty by remember { mutableStateOf("") }

var imageUri by remember { mutableStateOf<String?>(null) }

var selectedIngredients by remember { mutableStateOf(setOf<Int>()) }

var descriptionMedia by remember { mutableStateOf(listOf<String>()) }

var description by remember { mutableStateOf("") }

var showError by remember { mutableStateOf(false) }

var steps by remember { mutableStateOf(mutableListOf<RecipeStep>()) }

var newStepDescription by remember { mutableStateOf("") }

var newStepDuration by remember { mutableStateOf("") }

var newStepImageUri by remember { mutableStateOf<String?>(null) }

var newStepVideoUrl by remember { mutableStateOf("") }

val coroutineScope = rememberCoroutineScope()

val ingredients = listOf(

Ingredient(1, "Мясо", imageRes = R.drawable.meat),

# 

Ingredient(2, "Яйцо", imageRes = R.drawable.egg),

Ingredient(3, "Сыр", imageRes = R.drawable.cheese),

)

val context = LocalContext.current

val launcher = rememberLauncherForActivityResult(ActivityResultContracts.GetContent()) { uri ->

uri?.let { selectedUri ->

val savedPath = saveImageToInternalStorage(context, selectedUri)

)

imageUri = savedPath

}

}

val mediaLauncher = rememberLauncherForActivityResult(ActivityResultContracts.GetContent()) { uri ->

uri?.let { newUri ->

val savedPath = saveImageToInternalStorage(context, newUri)

savedPath?.let {

descriptionMedia = descriptionMedia + it

}

}

}

val stepImageLauncher = rememberLauncherForActivityResult(ActivityResultContracts.GetContent()) { uri ->

uri?.let { selectedUri ->

val savedPath = saveImageToInternalStorage(context, selectedUri)

newStepImageUri = savedPath

}

}

var selectedCategory by remember { mutableStateOf("Все") }

var newCategory by remember { mutableStateOf("") }

val categoriesState = remember { mutableStateOf(setOf<String>()) }

LaunchedEffect(Unit) {

CategoriesDataStore.getCategories(context).collectLatest { cats ->

categoriesState.value = cats

}

}

Scaffold(

topBar = {

TastyTopBar(title = "Добавить рецепт", onMenuClick = onMenuClick, onBackClick = if (onMenuClick == null) { { navController.popBackStack() } } else null)

},

containerColor = PastelBg

) { paddingValues ->

Box(

modifier = Modifier

.fillMaxSize()

.background(

brush = androidx.compose.ui.graphics.Brush.verticalGradient(

colors = listOf(PastelBg, White)

)

)

.padding(paddingValues)

) {

LazyColumn(

modifier = Modifier.fillMaxSize(),

horizontalAlignment = Alignment.CenterHorizontally

) {

item {

Box(modifier = Modifier.height(250.dp)) {

Box(

modifier = Modifier

.fillMaxWidth()

.height(250.dp)

.clickable { launcher.launch("image/\*") },

contentAlignment = Alignment.Center

) {

if (imageUri != null) {

Image(

painter = rememberAsyncImagePainter(

ImageRequest.Builder(LocalContext.current)

.data(imageUri)

.build()

),

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier.fillMaxSize()

)

} else {

Image(

painter = rememberAsyncImagePainter("https://i.pinimg.com/736x/fb/90/84/fb9084df5b28f78ba9ba7f27de810c70.jpg?text=Recipe+Image"),

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier.fillMaxSize()

)

}

}

Icon(

Icons.Default.Favorite,

contentDescription = null,

tint = PastelPink,

modifier = Modifier

.align(Alignment.TopStart)

.padding(16.dp)

.background(White, shape = CircleShape)

.padding(8.dp)

)

}

}

item {

Card(

modifier = Modifier

.fillMaxSize()

.offset(y = (-24).dp)

.clip(RoundedCornerShape(topStart = 28.dp, topEnd = 28.dp))

.background(White),

colors = CardDefaults.cardColors(containerColor = White)

) {

Column(

modifier = Modifier

.fillMaxSize()

.padding(20.dp),

verticalArrangement = Arrangement.spacedBy(12.dp)

) {

Text(

text = if (title.isBlank()) "Изменить рецепт" else title,

fontSize = 22.sp,

fontWeight = FontWeight.Bold,

color = TextColor

)

OutlinedTextField(

value = title,

onValueChange = { title = it },

label = { Text("Название блюда") },

modifier = Modifier.fillMaxWidth()

)

OutlinedTextField(

value = cookTime,

onValueChange = { cookTime = it },

label = { Text("Время готовки") },

modifier = Modifier.fillMaxWidth()

)

OutlinedTextField(

value = difficulty,

onValueChange = { difficulty = it },

label = { Text("Сложность") },

modifier = Modifier.fillMaxWidth()

)

OutlinedTextField(

value = description,

onValueChange = { description = it },

label = { Text("Описание блюда") },

modifier = Modifier.fillMaxWidth()

)

Text(stringResource(R.string.description\_media), fontWeight = FontWeight.SemiBold)

LazyRow(modifier = Modifier.fillMaxWidth()) {

items(descriptionMedia) { mediaUrl ->

Image(

painter = rememberAsyncImagePainter(mediaUrl),

contentDescription = null,

modifier = Modifier

.size(100.dp)

.padding(4.dp),

contentScale = ContentScale.Crop

)

}

}

Button(

onClick = { mediaLauncher.launch("image/\*") },

colors = ButtonDefaults.buttonColors(containerColor = PastelBg),

modifier = Modifier.fillMaxWidth()

) {

Text("Добавить файл", color = PastelPink)

}

Text("Выберите ингредиенты", fontWeight = FontWeight.SemiBold)

LazyRow(modifier = Modifier.fillMaxWidth())

items(ingredients) { ingredient ->

val selected = ingredient.id in selectedIngredients

Box(

modifier = Modifier

.padding(8.dp)

.size(64.dp)

.clip(CircleShape)

.background(if (selected) PastelPink else PastelBg)

.border(2.dp, if (selected) PastelPink else Color.LightGray, CircleShape)

.clickable {

selectedIngredients = if (selected)

selectedIngredients - ingredient.id

else

selectedIngredients + ingredient.id

},

contentAlignment = Alignment.Center

) {

if (ingredient.imageRes != null) {

Image(

painter = painterResource(id = ingredient.imageRes),

contentDescription = null,

modifier = Modifier.size(48.dp),

contentScale = ContentScale.Crop

)

} else {

Image(

painter = rememberAsyncImagePainter(ingredient.imageUrl),

contentDescription = null,

modifier = Modifier.size(48.dp),

contentScale = ContentScale.Crop

)

}

}

}

}

Text("Категории", fontWeight = FontWeight.SemiBold)

Row(verticalAlignment = Alignment.CenterVertically) {

var expanded by remember { mutableStateOf(false) }

OutlinedButton(onClick = { expanded = true }) {

Text(selectedCategory)

}

DropdownMenu(expanded = expanded, onDismissRequest = { expanded = false }) {

categoriesState.value.forEach { cat ->

DropdownMenuItem(

text = { Text(cat) },

onClick = {

selectedCategory = cat

expanded = false

}

)

}

}

Spacer(modifier = Modifier.width(8.dp))

OutlinedTextField(

value = newCategory,

onValueChange = { newCategory = it },

label = { Text(stringResource(R.string.new\_category)) },

singleLine = true,

modifier = Modifier.width(140.dp)

)

Button(

onClick = {

if (newCategory.isNotBlank()) {

coroutineScope.launch {

CategoriesDataStore.addCategory(context, newCategory)

selectedCategory = newCategory

newCategory = ""

}

}

},

enabled = newCategory.isNotBlank(),

modifier = Modifier.padding(start = 4.dp)

) {

Text("+")

}

}

Text("Шаги приготовления", fontWeight = FontWeight.SemiBold)

Column {

steps.forEachIndexed { idx, step ->

Card(

modifier = Modifier

.fillMaxWidth()

.padding(vertical = 4.dp),

colors = CardDefaults.cardColors(containerColor = PastelBg)

) {

Row(modifier = Modifier.padding(8.dp), verticalAlignment = Alignment.CenterVertically) {

if (!step.imageUrl.isNullOrBlank()) {

Image(

painter = rememberAsyncImagePainter(step.imageUrl),

contentDescription = null,

modifier = Modifier.size(56.dp).clip(RoundedCornerShape(8.dp)),

contentScale = ContentScale.Crop

)

}

Column(modifier = Modifier.weight(1f).padding(start = 8.dp)) {

Text(step.description, fontWeight = FontWeight.Medium)

Text("Время: ${step.durationMinutes} мин", fontSize = 12.sp, color = Color.Gray)

}

IconButton(onClick = { steps.removeAt(idx) }) {

Icon(Icons.Default.Favorite, contentDescription = "Удалить", tint = PastelPink)

}

}

}

}

Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.padding(top = 8.dp)) {

OutlinedTextField(

value = newStepDescription,

onValueChange = { newStepDescription = it },

label = { Text("Описание шага") },

modifier = Modifier.weight(1f)

)

Spacer(Modifier.width(8.dp))

OutlinedTextField(

value = newStepDuration,

onValueChange = { newStepDuration = it.filter { ch -> ch.isDigit() } },

label = { Text("Мин.") },

modifier = Modifier.width(70.dp)

)

Spacer(Modifier.width(8.dp))

Button(onClick = { stepImageLauncher.launch("image/\*") }, colors = ButtonDefaults.buttonColors(containerColor = PastelBg)) {

Text("Фото", color = PastelPink)

}

}

OutlinedTextField(

value = newStepVideoUrl,

onValueChange = { newStepVideoUrl = it },

label = { Text("Ссылка на видео или YouTube (необязательно)") },

modifier = Modifier.fillMaxWidth().padding(top = 8.dp)

)

if (newStepImageUri != null) {

Image(

painter = rememberAsyncImagePainter(newStepImageUri),

contentDescription = null,

modifier = Modifier.size(56.dp).clip(RoundedCornerShape(8.dp)),

contentScale = ContentScale.Crop

)

}

Button(

onClick = {

if (newStepDescription.isNotBlank() && newStepDuration.isNotBlank()) {

steps.add(

RecipeStep(

description = newStepDescription,

durationMinutes = newStepDuration.toIntOrNull() ?: 5,

imageUrl = if (newStepVideoUrl.isNotBlank()) newStepVideoUrl else newStepImageUri

)

)

newStepDescription = ""

newStepDuration = ""

newStepImageUri = null

newStepVideoUrl = ""

# 

}

},

enabled = newStepDescription.isNotBlank() && newStepDuration.isNotBlank(),

modifier = Modifier.padding(top = 8.dp)

) {

Text("Добавить шаг")

}

}

Spacer(modifier = Modifier.height(16.dp))

Button(

onClick = {

if (title.isBlank() || cookTime.isBlank() || difficulty.isBlank()) {

showError = true

} else {

coroutineScope.launch {

val selectedNames = ingredients.filter { it.id in selectedIngredients }.map { it.name }

val recipe = RecipeEntity(

title = title,

imageUrl = imageUri,

cookingTime = cookTime,

difficulty = difficulty.toIntOrNull() ?: 1,

description = description,

rating = 0f,

isFavorite = false,

ingredients = selectedNames,

# 

descriptionMedia = steps.map { RecipeStepSerializer.toJson(listOf(it)) },

category = selectedCategory,

steps = steps

)

viewModel.insertRecipe(recipe)

navController.popBackStack()

}

}

},

colors = ButtonDefaults.buttonColors(containerColor = PastelPink),

modifier = Modifier.fillMaxWidth(),

shape = RoundedCornerShape(16.dp)

) {

Text(stringResource(R.string.save), color = White)

}

if (showError) {

Text("Заполните все

поля", color = Color.Red)

}

}

}

}

}

}

}

}

package com.students.tastyfood.ui.screens

# 

import androidx.compose.animation.AnimatedVisibility

import androidx.compose.animation.fadeIn

import androidx.compose.animation.slideInVertically

import androidx.compose.foundation.Image

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.layout.width

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.filled.Add

import androidx.compose.material.icons.filled.Favorite

import androidx.compose.material.icons.filled.FavoriteBorder

import androidx.compose.material3.Card

import androidx.compose.material3.CardDefaults

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.FloatingActionButton

import androidx.compose.material3.Icon

import androidx.compose.material3.IconButton

import androidx.compose.material3.LinearProgressIndicator

# 

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.collectAsState

import androidx.compose.runtime.getValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.clip

import androidx.compose.ui.draw.shadow

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.graphics.Color.Companion.White

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.navigation.NavController

import coil3.compose.rememberAsyncImagePainter

import com.students.tastyfood.ui.theme.PastelPink

import com.students.tastyfood.ui.theme.TextColor

import com.students.tastyfood.viewmodel.RecipeViewModel

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun FavoritesScreen(navController: NavController, viewModel: RecipeViewModel, onMenuClick: (() -> Unit)? = null) {

val recipes by viewModel.recipes.collectAsState()

val favoriteRecipes = recipes.filter { it.isFavorite }

Scaffold(

topBar = {

# 

TastyTopBar(title = "Избранное", onMenuClick = onMenuClick, onBackClick = if (onMenuClick == null) { { navController.popBackStack() } } else null)

},

floatingActionButton = {

FloatingActionButton(onClick = { navController.navigate("addrecipe") }, containerColor = PastelPink) {

Icon(Icons.Default.Add, contentDescription = "Добавить рецепт", tint = White)

}

},

containerColor = White

) { paddingValues ->

if (favoriteRecipes.isEmpty()) {

Box(modifier = Modifier.fillMaxSize().padding(paddingValues), contentAlignment = Alignment.Center) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Icon(

imageVector = Icons.Default.FavoriteBorder,

contentDescription = null,

tint = PastelPink,

modifier = Modifier.size(64.dp)

)

Spacer(modifier = Modifier.height(12.dp))

Text("Нет избранных рецептов", color = PastelPink, fontWeight = FontWeight.Bold)

Text("Добавьте понравившиеся блюда в избранное", fontSize = 13.sp, color = Color.Gray)

}

# 

}

} else {

LazyColumn(modifier = Modifier.fillMaxSize().padding(paddingValues)) {

items(favoriteRecipes,

key = { it.id }) { recipe ->

AnimatedVisibility(

visible = true,

enter = fadeIn() + slideInVertically(initialOffsetY = { it / 2 }),

) {

Card(

modifier = Modifier

.fillMaxWidth()

.padding(8.dp)

.clickable { navController.navigate("recipeDetail/${recipe.id}") }

.shadow(12.dp, RoundedCornerShape(24.dp)),

shape = RoundedCornerShape(24.dp),

elevation = CardDefaults.cardElevation(0.dp),

colors = CardDefaults.cardColors(containerColor = Color(0xFFFDF6F9))

) {

Row(modifier = Modifier.padding(12.dp), verticalAlignment = Alignment.CenterVertically) {

Image(

painter = rememberAsyncImagePainter(recipe.imageUrl),

contentDescription = null,

modifier = Modifier.size(90.dp).clip(RoundedCornerShape(18.dp)),

# 

contentScale = ContentScale.Crop

)

Spacer(modifier = Modifier.width(12.dp))

Column(modifier = Modifier.weight(1f)) {

Text(recipe.title, fontWeight = FontWeight.Bold, color = TextColor)

Text("Время: ${recipe.cookingTime} мин", fontSize = 12.sp, color = PastelPink)

Text("★ ${recipe.rating}", fontSize = 12.sp, color = PastelPink)

LinearProgressIndicator(

progress = { recipe.difficulty / 5f },

modifier = Modifier

.fillMaxWidth(0.7f)

.height(6.dp)

.padding(top = 4.dp, bottom = 2.dp),

color = PastelPink,

trackColor = Color(0xFFF8EAF3),

strokeCap = androidx.compose.ui.graphics.StrokeCap.Round,

)

Text("Сложность: ${recipe.difficulty}/5", fontSize = 11.sp, color = PastelPink)

}

IconButton(

onClick = { viewModel.toggleFavorite(recipe) },

modifier = Modifier

) {

Icon(

# 

imageVector = if (recipe.isFavorite) Icons.Default.Favorite else Icons.Default.FavoriteBorder,

contentDescription = null,

tint = PastelPink

)

}

}

}

}

}

}

}

}

}

package com.students.tastyfood.ui.screens

import androidx.compose.animation.AnimatedVisibility

import androidx.compose.animation.fadeIn

import androidx.compose.animation.slideInVertically

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.PaddingValues

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxHeight

import androidx.compose.foundation.layout.fillMaxSize

# 

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.layout.width

import androidx.compose.foundation.lazy.LazyRow

import androidx.compose.foundation.lazy.grid.GridCells

import androidx.compose.foundation.lazy.grid.LazyVerticalGrid

import androidx.compose.foundation.lazy.grid.items

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.automirrored.filled.List

import androidx.compose.material.icons.filled.Add

import androidx.compose.material.icons.filled.Favorite

import androidx.compose.material.icons.filled.FavoriteBorder

import androidx.compose.material.icons.filled.Home

import androidx.compose.material.icons.filled.Menu

import androidx.compose.material.icons.filled.Notifications

import androidx.compose.material.icons.filled.Search

import androidx.compose.material.icons.filled.Settings

import androidx.compose.material3.AlertDialog

import androidx.compose.material3.Button

import androidx.compose.material3.ButtonDefaults

import androidx.compose.material3.Card

import androidx.compose.material3.CardDefaults

import androidx.compose.material3.DrawerValue

import androidx.compose.material3.FloatingActionButton

import androidx.compose.material3.Icon

# 

import androidx.compose.material3.IconButton

import androidx.compose.material3.LinearProgressIndicator

import androidx.compose.material3.ModalNavigationDrawer

import androidx.compose.material3.NavigationDrawerItem

import androidx.compose.material3.NavigationDrawerItemDefaults

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Text

import androidx.compose.material3.TextButton

import androidx.compose.material3.TextField

import androidx.compose.material3.TextFieldDefaults

import androidx.compose.material3.rememberDrawerState

import androidx.compose.runtime.Composable

import androidx.compose.runtime.LaunchedEffect

import androidx.compose.runtime.collectAsState

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.rememberCoroutineScope

import androidx.compose.runtime.setValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.draw.clip

import androidx.compose.ui.draw.shadow

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.graphics.StrokeCap

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.platform.LocalContext

import androidx.compose.ui.res.stringResource

import androidx.compose.ui.text.font.FontWeight

# 

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.navigation.NavController

import coil3.compose.rememberAsyncImagePainter

import com.students.tastyfood.R

import com.students.tastyfood.data.local.CategoriesDataStore

import com.students.tastyfood.data.local.entity.RecipeEntity

import com.students.tastyfood.data.remote.MealDto

import com.students.tastyfood.ui.theme.PastelBg

import com.students.tastyfood.ui.theme.PastelPink

import com.students.tastyfood.ui.theme.TextColor

import com.students.tastyfood.viewmm

s.tastyfood.viewmodel.RecipeViewModel

import kotlinx.coroutines.flow.collectLatest

import kotlinx.coroutines.launch

@Composable

fun HomeScreen(navController: NavController, viewModel: RecipeViewModel) {

val recipes by viewModel.recipes.collectAsState()

val randomMeals by viewModel.randomMeals.collectAsState()

var searchQuery by remember { mutableStateOf("") }

val context = LocalContext.current

var selectedCategory by remember { mutableStateOf("Все") }

val categoriesState = remember { mutableStateOf(setOf<String>()) }

val drawerState = rememberDrawerState(DrawerValue.Closed)

val scope = rememberCoroutineScope()

val menuItems = listOf(

Triple("Главная", "home", Icons.Default.Home),

Triple("Избранное", "favorites", Icons.Default.Favorite),

# 

Triple("Мои рецепты", "myrecipes", Icons.AutoMirrored.Filled.List),

Triple("Добавить рецепт", "addrecipe", Icons.Default.Add),

Triple("Настройки", "settings", Icons.Default.Settings)

)

var selectedMenu by remember { mutableStateOf("home") }

var showNetworkLoadButton by remember { mutableStateOf(true) }

var showMealDialog by remember { mutableStateOf<MealDto?>(null) }

LaunchedEffect(Unit) {

CategoriesDataStore.getCategories(context).collectLatest { cats ->

categoriesState.value = cats

}

}

LaunchedEffect(key1 = true) {

viewModel.loadRandomMeals(2)

}

val filteredRecipes = recipes

.filter { selectedCategory == "Все" || it.category == selectedCategory }

.filter { it.title.contains(searchQuery, ignoreCase = true) }

val combinedRecipes = remember(filteredRecipes, randomMeals) {

val local = filteredRecipes.map { it to null }

val remote = randomMeals.map { null to it }

local + remote

}

ModalNavigationDrawer(

drawerState = drawerState,

drawerContent = {

Column(modifier = Modifier.fillMaxHeight().background(PastelBg)) {

Spacer(Modifier.height(32.dp))

# 

menuItems.forEach { (title, route, icon) ->

val isSelected = selectedMenu == route

Box(

modifier = Modifier

.fillMaxWidth()

.padding(horizontal = 8.dp, vertical = 4.dp)

.then(

if (isSelected)

Modifier

.shadow(8.dp, RoundedCornerShape(18.dp))

.background(Color(0xFFFDF6F9), RoundedCornerShape(18.dp))

else Modifier

)

) {

NavigationDrawerItem(

label = {

Row(verticalAlignment = Alignment.CenterVertically) {

Icon(icon, contentDescription = null, tint = if (isSelected) PastelPink else TextColor)

Spacer(Modifier.width(12.dp))

Text(title, color = if (isSelected) PastelPink else TextColor)

}

},

selected = isSelected,

onClick = {

selectedMenu = route

scope.launch { drawerState.close() }

# 

if (route != "home") navController.navigate(route)

},

modifier = Modifier.fillMaxWidth().background(Color.Transparent),

colors = NavigationDrawerItemDefaults.colors(

selectedContainerColor = Color.Transparent,

unselectedContainerColor = Color.Transparent

)

)

}

}

}

},

content = {

Scaffol d(

floatingActionButton = {

FloatingActionButton(onClick = { navController.navigate("addrecipe") }, containerColor = PastelPink) {

Icon(Icons.Default.Add, contentDescription = "Добавить рецепт", tint = PastelBg)

}

},

content = { paddingValues ->

Box(

modifier = Modifier

.fillMaxSize()

.background(PastelBg)

.padding(paddingValues)

) {

# 

Column(modifier = Modifier.fillMaxSize()) {

Row(

modifier = Modifier

.fillMaxWidth()

.padding(top = 12.dp, start = 12.dp, end = 12.dp),

verticalAlignment = Alignment.CenterVertically

) {

IconButton(onClick = { scope.launch { drawerState.open() } }) {

Icon(Icons.Default.Menu, contentDescription = null, tint = TextColor)

}

Spacer(modifier = Modifier.weight(1f))

IconButton(onClick = { }) {

Icon(Icons.Default.Notifications, contentDescription = null, tint = PastelPink)

}

}

Text(

text = stringResource(R.string.delicious),

fontSize = 14.sp,

color = Color.Gray,

modifier = Modifier.padding(start = 20.dp)

)

Text(

text = stringResource(R.string.easy\_to\_cook),

fontSize = 22.sp,

fontWeight = FontWeight.Bold,

# 

color = TextColor,

modifier = Modifier.padding(start = 20.dp, top = 2.dp, bottom = 4.dp)

)

// Search bar styled like card

Card(

shape = RoundedCornerShape(16.dp),

modifier = Modifier

.fillMaxWidth()

.padding(horizontal = 20.dp),

elevation = CardDefaults.cardElevation(4.dp)

) {

TextField(

value = searchQuery,

onValueChange = { searchQuery = it },

placeholder = { Text(stringResource(R.string.search\_perfect), color = Color.Gray) },

leadingIcon = {

Icon(Icons.Default.Search, contentDescription = null, tint = PastelPink)

},

modifier = Modifier.fillMaxWidth(),

colors = TextFieldDefaults.colors(

unfocusedContainerColor = PastelBg,

focusedContainerColor = PastelBg,

focusedIndicatorColor = Color.Transparent,

unfocusedIndicatorColor = Color.Transparent

),

# 

singleLine = true

)

}

Text(

text = stringResource(R.string.category),

fontWeight = FontWeight.SemiBold,

color = TextColor,

fontSize =

= 1

fontSize = 16.sp,

modifier = Modifier.padding(start = 20.dp, top = 12.dp, bottom = 6.dp)

)

LazyRow(

contentPadding = PaddingValues(horizontal = 12.dp),

horizontalArrangement = Arrangement.spacedBy(12.dp)

) {

items(categoriesState.value.toList()) { category ->

val selected = category == selectedCategory

Card(

shape = RoundedCornerShape(24.dp),

elevation = CardDefaults.cardElevation(4.dp),

colors = CardDefaults.cardColors(

containerColor = if (selected) PastelPink else PastelBg

),

modifier = Modifier

# 

.clickable { selectedCategory = category }

) {

Text(

text = category,

color = if (selected) PastelBg else TextColor,

fontWeight = FontWeight.Medium,

modifier = Modifier.padding(horizontal = 20.dp, vertical = 10.dp),

fontSize = 14.sp

)

}

}

}

Text(

text = stringResource(R.string.popular),

fontWeight = FontWeight.Bold,

color = TextColor,

fontSize = 18.sp,

modifier = Modifier.padding(start = 20.dp, top = 16.dp, bottom = 8.dp)

)

if (showNetworkLoadButton) {

Button(

onClick = {

viewModel.loadRandomMeals(2)

showNetworkLoadButton = false

},

modifier = Modifier.padding(16.dp),colors = ButtonDefaults.buttonColors(

# 

containerColor = PastelPink,

contentColor = PastelBg

)

) {

Text("Загрузить сетевые рецепты")

}

}

Text(

text = "Сетевых рецептов: ${randomMeals.size}",

color = Color.Gray,

fontSize = 12.sp,

modifier = Modifier.padding(start = 20.dp, bottom = 4.dp)

)

if (randomMeals.isNotEmpty()) {

Text(

text = "Первый сетевой рецепт: ${randomMeals.firstOrNull()?.strMeal ?: "-"}",

color = Color.Gray,

fontSize = 12.sp,

modifier = Modifier.padding(start = 20.dp, bottom = 4.dp)

)

}

if (combinedRecipes.isEmpty()) {

Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Icon(imageVector = Icons.Default.FavoriteBorder,

contentDescription = null,

# 

tint = PastelPink,

modifier = Modifier.size(64.dp)

)

Spacer(modifier = Modifier.height(12.dp))

Text("Нет рецептов", color = PastelPink, fontWeight = FontWeight.Bold)

Text("Добавьте свой первый рецепт или измените фильтры", fontSize = 13.sp, color = Color.Gray)

}

}

} else {

LazyVerticalGrid(

columns = GridCells.Fixed(2),

contentPadding = PaddingValues(horizontal = 16.dp, vertical = 8.dp),

verticalArrangement = Arrangement.spacedBy(12.dp),

horizontalArrangement = Arrangement.spacedBy(12.dp),

modifier = Modifier.fillMaxSize()

) {

items(combinedRecipes) { (localRecipe, remoteRecipe) ->

AnimatedVisibility(

visible = true,

enter = fadeIn() + slideInVertically(initialOffsetY = { it / 2 }),

) {

if (localRecipe != null) {

Card(

modifier = Modifier

# 

.fillMaxWidth()

.height(220.dp)

.clickable { navController.navigate("recipeDetail/${localRecipe.id}") }

.shadow(12.dp, RoundedCornerShape(24.dp)),

shape = RoundedCornerShape(24.dp),

elevation = CardDefaults.cardElevation(0.dp),

colors = CardDefaults.cardColors(containerColor = Color(0xFFFDF6F9))

) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Box(modifier = Modifier.fillMaxWidth()) {

val painter = rememberAsyncImagePainter(localRecipe.imageUrl)

if (!localRecipe.imageUrl.isNullOrBlank()) {

Image(

painter = painter,

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.clip(RoundedCornerShape(topStart = 24.dp, topEnd = 24.dp))

)

# 

if (painter.state is coil3.compose.AsyncImagePainter.State.Error) {

Box(

modifier =

Modif

ier

.height(120.dp)

.fillMaxWidth()

.background(Color(0xFFF8EAF3)),

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color(0xFFB39EB5))

}

}

} else {

Box(

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.background(Color(0xFFF8EAF3)),

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color(0xFFB39EB5))

}

# 

}

IconButton(

onClick = { viewModel.toggleFavorite(localRecipe) },

modifier = Modifier

.align(Alignment.TopEnd)

.padding(6.dp)

) {

Icon(

imageVector = if (localRecipe.isFavorite) Icons.Default.Favorite else Icons.Default.FavoriteBorder,

contentDescription = null,

tint = PastelPink

)

}

}

Spacer(modifier = Modifier.height(6.dp))

Text(localRecipe.title, fontWeight = FontWeight.Bold, fontSize = 15.sp, color = TextColor)

Text("${localRecipe.cookingTime} мин", fontSize = 12.sp, color = PastelPink)

Text("★ ${localRecipe.rating}", fontSize = 12.sp, color = PastelPink)

LinearProgressIndicator(

progress = { localRecipe.difficulty / 5f },

modifier = Modifier

.fillMaxWidth(0.7f)

.height(6.dp)

# 

.padding(top = 4.dp, bottom = 2.dp),

color = PastelPink,

trackColor = Color(0xFFF8EAF3),

strokeCap = StrokeCap.Round,

)

Text("Сложность: ${localRecipe.difficulty}/5", fontSize = 11.sp, color = PastelPink)

}

}

} else if (remoteRecipe != null) {

Card(

modifier = Modifier

.fillMaxWidth()

.height(220.dp)

.clickable { showMealDialog = remoteRecipe }

.shadow(12.dp, RoundedCornerShape(24.dp)),

shape = RoundedCornerShape(24.dp),

elevation = CardDefaults.cardElevation(0.dp),

colors = CardDefaults.cardColors(containerColor = Color(0xFFFDF6F9))

) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Box(modifier = Modifier.fillMaxWidth()) {

val painter = rememberAsyncImagePainter(remoteRecipe.strMealThumb)

# 

if (!remoteRecipe.strMealThumb.isNullOrBlank()) {

Image(

painter = painter,

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.clip(RoundedCornerShape(topStart = 24.dp, topEnd = 24.dp))

)

if (painter.state is coil3.compose.AsyncImagePainter.State.Error) {

Box(

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.background(Color(0xFFF8EAF3)),

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color(0xFFB39EB5))

}

}

} else {

Box(

# 

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.background(Color(0xFFF8EAF3)),

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color(0xFFB39EB5))

}

}

IconButton(

onClick = {

val entity = mealDtoToRecipeEntity(remoteRecipe)

viewModel.insertRecipe(entity)

},

modifier = Modifier

.align(Alignment.TopEnd)

.padding(6.dp)

) {

Icon(

imageVector = Icons.Default.FavoriteBorder,

contentDescription = null,

tint = PastelPink

)

}

}

# 

Spacer(modifier = Modifier.height(6.dp))

Text(remoteRecipe.strMeal ?: "Без названия", fontWeight = FontWeight.Bold, fontSize = 15.sp, color = TextColor)

Text(remoteRecipe.strCategory ?: "", fontSize = 12.sp, color = PastelPink)

Text(remoteRecipe.strArea ?: "", fontSize = 12.sp, color = PastelPink)

}

}

}

}

}

}

}

}

}

}

)

}

)

if (showMealDialog != null) {

AlertDialog(

onDismissRequest = { showMealDialog = null },

title = { Text(showMealDialog?.strMeal ?: "Рецепт") },

text = {

Column {

if (!showMealDialog?.strMealThumb.isNullOrBlank()) {

# 

Image(

painter = rememberAsyncImagePainter(showMealDialog?.strMealThumb),

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier

.fillMaxWidth()

.height(180.dp)

)

}

Text(showMealDialog?.strInstructions ?: "Нет описания")

}

},

confirmButton = {

TextButton(onClick = { showMealDialog = null }) {

Text("Закрыть")

}

}

)

}

}

@Composable

fun RecipeCardGridItem(

recipe: RecipeEntity,

onClick: () -> Unit,

onFavoriteClick: () -> Unit,

pastelPink: Color,

textColor: Color

# 

) {

Card(

modifier = Modifier

.fillMaxWidth()

.height(220.dp)

.clickable { onClick() },

shape = RoundedCornerShape(16.dp),

elevation = CardDefaults.cardElevation(6.dp),

colors = CardDefaults.cardColors(containerColor = PastelBg)

) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Box(modifier =

Modifier.fillMaxWidth()) {

val painter = rememberAsyncImagePainter(recipe.imageUrl)

if (!recipe.imageUrl.isNullOrBlank()) {

Image(

painter = painter,

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

)

if (painter.state is coil3.compose.AsyncImagePainter.State.Error) {

Box(

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.background(Color.LightGray),

# 

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color.DarkGray)

}

}

} else {

Box(

modifier = Modifier

.height(120.dp)

.fillMaxWidth()

.background(Color.LightGray),

contentAlignment = Alignment.Center

) {

Text("Нет фото", color = Color.DarkGray)

}

}

IconButton(

onClick = onFavoriteClick,

modifier = Modifier

.align(Alignment.TopEnd)

.padding(6.dp)

) {

Icon(

imageVector = if (recipe.isFavorite) Icons.Default.Favorite else Icons.Default.FavoriteBorder,

contentDescription = null,

tint = PastelPink

)

}

# 

}

Spacer(modifier = Modifier.height(6.dp))

Text(recipe.title, fontWeight = FontWeight.Bold, fontSize = 15.sp, color = TextColor)

Text("${recipe.cookingTime} мин", fontSize = 12.sp, color = PastelPink)

Text("★ ${recipe.rating}", fontSize = 12.sp, color = PastelPink)

LinearProgressIndicator(

progress = { recipe.difficulty / 5f },

modifier = Modifier

.fillMaxWidth(0.7f)

.height(6.dp)

.padding(top = 4.dp, bottom = 2.dp),

color = PastelPink,

trackColor = Color(0xFFF8EAF3),

strokeCap = StrokeCap.Round,

)

Text("Сложность: ${recipe.difficulty}/5", fontSize = 11.sp, color = PastelPink)

}

}

}

fun mealDtoToRecipeEntity(meal: MealDto): RecipeEntity {

return RecipeEntity(

id = 0, // автоинкремент

title = meal.strMeal ?: "Без названия",

imageUrl = meal.strMealThumb,

cookingTime = "?", // TheMealDB не содержит времени готовки

difficulty = 1, // Можно добавить рандом или фиксированное значение

# 

description = meal.strInstructions ?: "",

rating = 0f,

isFavorite = true,

ingredients = emptyList(), // Можно доработать парсинг ингредиентов

descriptionMedia = emptyList(),

category = meal.strCategory ?: "Сеть"

)

}

package com.students.tastyfood.ui.screens

import androidx.compose.animation.AnimatedVisibility

import androidx.compose.animation.fadeIn

import androidx.compose.animation.slideInVertically

import androidx.compose.foundation.Image

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.PaddingValues

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.automirrored.filled.ArrowBack

# 

import androidx.compose.material.icons.filled.Add

import androidx.compose.material.icons.filled.Menu

import androidx.compose.material3.Card

import androidx.compose.material3.CardDefaults

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.FloatingActionButton

import androidx.compose.material3.Icon

import androidx.compose.material3.IconButton

import androidx.compose.material3.LinearProgressIndicator

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Text

import androidx.compose.material3.TopAppBar

import androidx.compose.material3.TopAppBarDefaults

import androidx.compose.runtime.Composable

import androidx.compose.runtime.collectAsState

import androidx.compose.runtime.getValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.graphics.Color.Companion.White

import androidx.compose.ui.graphics.StrokeCap

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.navigation.NavController

import coil3.compose.rememberAsyncImagePainter

import com.students.tastyfood.ui.theme.PastelPink

import com.students.tastyfood.ui.theme.TextColor

# 

import com.students.tastyfood.viewmodel.RecipeViewModel

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun TastyTopBar(

title: String,

onMenuClick: (() -> Unit)? = null,

onBackClick: (() -> Unit)? = null

) {

TopAppBar(

title = { Text(title, color = TextColor) },

navigationIcon = {

when {

onMenuClick != null -> {

IconButton(onClick = onMenuClick) {

Icon(Icons.Default.Menu, contentDescription = "Меню", tint = PastelPink)

}

}

onBackClick != null -> {

IconButton(onClick = onBackClick) {

Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Назад", tint = PastelPink)

}

}

}

},

colors = TopAppBarDefaults.topAppBarColors(

containerColor = White

)

# 

)

}

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun MyRecipesScreen(navController: NavController, viewModel: RecipeViewModel, onMenuClick: (() -> Unit)? = null) {

val myRecipes by viewModel.recipes.collectAsState()

Scaffold(

topBar = {

TastyTopBar(title = "Мои рецепты", onMenuClick = onMenuClick, onBackClick = if (onMenuClick == null) { { navController.popBackStack() } } else null)

},

floatingActionButton = {

FloatingActionButton(onClick = { navController.navigate("addrecipe") }, containerColor = PastelPink) {

Icon(Icons.Default.Add, contentDescription = "Добавить рецепт", tint = White)

}

},

containerColor = White

) { paddingValues ->

if

(myRecipes.isEmpty()) {

Box(modifier = Modifier.fillMaxSize().padding(paddingValues), contentAlignment = Alignment.Center) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

Icon(

imageVector = Icons.Default.Add,

# 

contentDescription = null,

tint = PastelPink,

modifier = Modifier.size(64.dp)

)

Spacer(modifier = Modifier.height(12.dp))

Text("У вас нет своих рецептов", color = PastelPink, fontWeight = FontWeight.Bold)

Text("Добавьте свой первый рецепт!", fontSize = 13.sp, color = Color.Gray)

}

}

} else {

LazyColumn(

modifier = Modifier

.padding(paddingValues)

.fillMaxSize(),

contentPadding = PaddingValues(16.dp),

verticalArrangement = Arrangement.spacedBy(12.dp)

) {

items(myRecipes, key = { it.id }) { recipe ->

AnimatedVisibility(

visible = true,

enter = fadeIn() + slideInVertically(initialOffsetY = { it / 2 }),

) {

Card(

modifier = Modifier

.fillMaxWidth()

.clickable { navController.navigate("recipeDetail/${recipe.id}") },

# 

shape = MaterialTheme.shapes.medium,

elevation = CardDefaults.cardElevation(4.dp)

) {

Row(modifier = Modifier.padding(12.dp)) {

Image(

painter = rememberAsyncImagePainter(recipe.imageUrl),

contentDescription = null,

modifier = Modifier

.size(80.dp)

.padding(end = 12.dp)

)

Column(modifier = Modifier.align(Alignment.CenterVertically)) {

Text(recipe.title, fontSize = 16.sp, fontWeight = FontWeight.Bold, color = TextColor)

Spacer(modifier = Modifier.height(4.dp))

Text("Добавлен вами", fontSize = 12.sp, color = PastelPink)

LinearProgressIndicator(

progress = { recipe.difficulty / 5f },

modifier = Modifier

.fillMaxWidth(0.6f)

.height(6.dp)

.padding(top = 4.dp, bottom = 2.dp),

color = PastelPink,

trackColor = Color.LightGray,

strokeCap = StrokeCap.Round,

)

# 

Text("Сложность: ${recipe.difficulty}/5", fontSize = 11.sp, color = Color.Gray)

}

}

}

}

}

}

}

}

}

package com.students.tastyfood.ui.screens

import androidx.compose.foundation.Image

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.lazy.LazyRow

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.foundation.shape.CircleShape

import androidx.compose.foundation.shape.RoundedCornerShape

import androidx.compose.material.icons.Icons

# 

import androidx.compose.material.icons.automirrored.filled.ArrowBack

import androidx.compose.material3.Button

import androidx.compose.material3.Icon

import androidx.compose.material3.IconButton

import androidx.compose.material3.LinearProgressIndicator

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.LaunchedEffect

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.rememberCoroutineScope

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.graphics.StrokeCap

import androidx.compose.ui.layout.ContentScale

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.compose.ui.draw.clip

import androidx.compose.ui.draw.shadow

import androidx.navigation.NavController

import coil3.compose.rememberAsyncImagePainter

import com.students.tastyfood.data.local.entity.Ingredient

import com.students.tastyfood.data.local.entity.RecipeEntity

import com.students.tastyfood.viewmodel.RecipeViewModel

import kotlinx.coroutines.flow.collectLatest

import com.halilibo.richtext.commonmark.Markdown

# 

import com.halilibo.richtext.ui.material3.RichText

@Composable

fun RecipeDetailScreen(navController: NavController, recipeId: Int, viewModel: RecipeViewModel) {

rememberCoroutineScope()

val recipeState = remember { mutableStateOf<RecipeEntity?>(null) }

LaunchedEffect(recipeId) {

viewModel.getRecipeById(recipeId).collectLatest { recipe ->

recipeState.value = recipe

}

}

val recipe = recipeState.value

recipe?.let {

LazyColumn(

modifier = Modifier

.fillMaxSize()

.background(Color(0xFFFFF6FA)) // светлый бело-розовый фон

) {

item {

// Кнопка "Назад" без белого фона и тени

IconButton(

onClick = { navController.popBackStack() },

modifier = Modifier.padding(12.dp)

) {

Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Назад", tint = Color(0xFFF96163))

}

Row(

modifier = Modifier

# 

.fillMaxWidth()

.padding(horizontal = 20.dp, vertical = 0.dp),

verticalAlignment = Alignment.CenterVertically

) {

Spacer(modifier = Modifier.weight(1f))

IconButton(

onClick = { /\* TODO: обработка избранного \*/ },

modifier = Modifier

) {

Icon(

painter = painterResource(id = com.students.tastyfood.R.drawable.ic\_favorite),

contentDescription = "В избранное",

tint = if (it.isFavorite) Color(0xFFF96163) else Color(0xFFBDBDBD)

)

}

}

Column(

modifier = Modifier

.padding(horizontal = 20.dp, vertical = 8.dp)

.shadow(12.dp, RoundedCornerShape(32.dp))

.background(Color.White, RoundedCornerShape(32.dp))

) {

Image(

painter = rememberAsyncImagePainter(model = it.imageUrl),

contentDescription = null,

contentScale = ContentScale.Crop,

modifier = Modifier

# 

.fillMaxWidth()

.height(200.dp)

.clip(RoundedCornerShape(topStart = 32.dp, topEnd = 32.dp))

)

Text(

text = it.title,

fontWeight = FontWeight.Bold,

fontSize = 24.sp,

color = Color(0xFFF96163),

modifier = Modifier.padding(16.dp)

)

}

val allIngredients = listOf(

Ingredient(1, "Мясо", imageRes = com.students.tastyfood.R.drawable.meat),

Ingredient(2, "Яйцо", imageRes = com.students.tastyfood.R.drawable.egg),

Ingredient(3, "Сыр", imageRes = com.students.tastyfood.R.drawable.cheese)

)

val recipeIngredients = allIngredients.filter { it.name in recipe.ingredients }

if (recipeIngredients.isNotEmpty()) {

Text("Ингредиенты:", fontWeight = FontWeight.SemiBold, fontSize = 16.sp, color = Color(0xFFF96163), modifier = Modifier.padding(start = 16.dp, top = 8.dp))

LazyRow(modifier = Modifier.padding(start = 16.dp, top = 4.dp, bottom = 8.dp)) {

items(recipeIngredients) { ingredient ->

# 

Box(

modifier = Modifier

.padding(end = 12.dp)

.size(56.dp)

.clip(CircleShape)

.background(Color.White),

contentAlignment = Alignment.Center

) {

if (ingredient.imageRes != null) {

Image(

painter = painterResource(id = ingredient.imageRes),

contentDescription = ingredient.name,

modifier = Modifier.size(36.dp)

)

}

}

}

}

}

LinearProgressIndicator(

progress = { it.difficulty / 5f },

modifier = Modifier

.fillMaxWidth(0.7f)

.height(8.dp)

.padding(start = 16.dp, top = 4.dp, bottom = 2.dp)

.shadow(2.dp, RoundedCornerShape(8.dp)),

color = Color(0xFFF96163),

trackColor = Color(0xFFFFE0EC),

strokeCap = StrokeCap.Round,

# 

)

Text("Сложность: ${it.difficulty}/5", fontSize = 13.sp, color = Color(0xFFF96163), modifier = Modifier.padding(start = 16.dp, bottom = 8.dp))

Text(

text = "Время готовки: ${recipe.cookingTime} мин",

fontSize = 16.sp,

color = Color(0xFFB23A48),

modifier = Modifier.padding(horizontal = 16.dp)

)

Text(

text = "Оценка: ${it.rating} ⭐",

fontSize = 16.sp,

color = Color(0xFFB23A48),

modifier = Modifier.padding(horizontal = 16.dp, vertical = 4.dp)

)

val stepsList = it.steps

Text(

text = "Шагов: ${stepsList.size}",

color = Color(0xFFB23A48),

fontSize = 14.sp,

modifier = Modifier.padding(horizontal = 16.dp, vertical = 4.dp)

)

val hasSteps = stepsList.isNotEmpty()

val stepId = if (hasSteps) 0 else 0

Text(text = "Описание:", color = Color(0xFFF96163), fontWeight = FontWeight.SemiBold, modifier = Modifier.padding(start = 16.dp, top = 8.dp))

RichText(

modifier = Modifier.padding(16.dp)

) {

# 

Markdown(it.description)

}

Button(

onClick = {

if (hasSteps) {

navController.navigate("recipeSteps/${it.id}/$stepId")

}

},

enabled = hasSteps,

modifier = Modifier

.padding(16.dp)

.fillMaxWidth()

.shadow(8.dp, RoundedCornerShape(24.dp)),

shape = RoundedCornerShape(24.dp),

colors = androidx.compose.material3.ButtonDefaults.buttonColors(

containerColor = Color(0xFFF96163),

contentColor = Color.White

)

) {

Text("Приступить", fontWeight = FontWeight.Bold, fontSize = 18.sp)

}

}

}

}

}

package com.students.tastyfood.ui.screens

package com.students.tastyfood.ui.screens

# 

import androidx.compose.foundation.Image

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.material3.Button

import androidx.compose.material3.ButtonDefaults

import androidx.compose.material3.ExperimentalMaterial3Api

import androidx.compose.material3.OutlinedTextField

import androidx.compose.material3.Scaffold

import androidx.compose.material3.Slider

import androidx.compose.material3.SliderDefaults

import androidx.compose.material3.Switch

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

import androidx.compose.runtime.setValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.res.painterResource

import androidx.compose.ui.res.stringResource

import androidx.compose.ui.unit.dp

import androidx.navigation.NavController

import com.students.tastyfood.R

# 

import com.students.tastyfood.ui.theme.PastelBg

import com.students.tastyfood.ui.theme.PastelPink

import com.students.tastyfood.ui.theme.TextColor

import com.students.tastyfood.viewmodel.SettingsViewModel

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun SettingsScreen(navController: NavController, settingsViewModel: SettingsViewModel, onMenuClick: (() -> Unit)? = null) {

val isDarkTheme = settingsViewModel.isDarkTheme

val fontSize = settingsViewModel.fontSize

var fullName by remember { mutableStateOf("") }

var birthDate by remember { mutableStateOf("") }

Scaffold(

topBar = {

TastyTopBar(

title = "Настройки",

onMenuClick = onMenuClick,

onBackClick = if (onMenuClick == null) { { navController.popBackStack() } } else null

)

},

containerColor = PastelBg

) { paddingValues ->

Column(

modifier = Modifier

.padding(paddingValues)

.padding(16.dp)

.fillMaxSize(),

verticalArrangement = Arrangement.spacedBy(16.dp)

# 

) {

Row(

verticalAlignment = Alignment.CenterVertically,

horizontalArrangement = Arrangement.SpaceBetween,

modifier = Modifier.fillMaxWidth()

) {

Text("Тёмная тема", color = PastelPink)

Switch(checked = isDarkTheme, onCheckedChange = { settingsViewModel.isDarkTheme = it },

colors = androidx.compose.material3.SwitchDefaults.colors(checkedThumbColor = PastelPink))

}

Row(

verticalAlignment = Alignment.CenterVertically,

horizontalArrangement = Arrangement.SpaceBetween,

modifier = Modifier.fillMaxWidth()

) {

Text("Размер шрифта", color = TextColor)

Slider(value = fontSize, colors = SliderDefaults.colors(thumbColor = PastelPink, activeTrackColor = PastelPink) , onValueChange = { settingsViewModel.fontSize = it }, valueRange = 12f..24f)

}

OutlinedTextField(

value = fullName,

onValueChange = { fullName = it },

label = { Text("ФИО") },

modifier = Modifier.fillMaxWidth()

)

# 

OutlinedTextField(

value = birthDate,

onValueChange = { birthDate = it },

label = { Text("Дата рождения") },

modifier = Modifier.fillMaxWidth()

)

Row(

verticalAlignment = Alignment.CenterVertically,

horizontalArrangement = Arrangement.SpaceBetween,

modifier

= Modifier.fillMaxWidth()

) {

Text("Язык приложения", color = TextColor)

Row(verticalAlignment = Alignment.CenterVertically) {

Image(

painter = painterResource(id = R.drawable.ic\_flag\_ru),

contentDescription = stringResource(id = R.string.lang\_russian),

modifier = Modifier.size(24.dp)

)

Switch(

checked = settingsViewModel.isEnglish,

onCheckedChange = { settingsViewModel.isEnglish = it },

colors = androidx.compose.material3.SwitchDefaults.colors(

checkedThumbColor = PastelPink,

uncheckedThumbColor = PastelBg,

checkedTrackColor = PastelBg,

uncheckedTrackColor = PastelPink

),

thumbContent = {

# 

if (settingsViewModel.isEnglish) {

Image(

painter = painterResource(id = R.drawable.ic\_flag\_us),

contentDescription = stringResource(id = R.string.lang\_english),

modifier = Modifier.size(20.dp)

)

} else {

Image(

painter = painterResource(id = R.drawable.ic\_flag\_ru),

contentDescription = stringResource(id = R.string.lang\_russian),

modifier = Modifier.size(20.dp)

)

}

}

)

Image(

painter = painterResource(id = R.drawable.ic\_flag\_us),

contentDescription = stringResource(id = R.string.lang\_english),

modifier = Modifier.size(24.dp)

)

}

}

Row(

verticalAlignment = Alignment.CenterVertically,

horizontalArrangement = Arrangement.SpaceBetween,

modifier = Modifier.fillMaxWidth()

) {

# 

Text("Уведомления", color = TextColor)

Switch(checked = true, onCheckedChange = { /\* TODO: Реализовать \*/ },

colors = androidx.compose.material3.SwitchDefaults.colors(

checkedThumbColor = PastelPink,

uncheckedThumbColor = PastelBg,

checkedTrackColor = PastelBg,

uncheckedTrackColor = PastelPink

)

)

}

Button(

onClick = { /\* TODO: Реализовать выход из аккаунта \*/ },

modifier = Modifier.fillMaxWidth(),

colors = ButtonDefaults.buttonColors(containerColor = PastelPink)

) {

Text("Выйти из аккаунта", color = PastelBg)

}

}

}

}

package com.students.tastyfood.ui.screens

import androidx.compose.foundation.background

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.size

# 

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Text

import androidx.compose.runtime.Composable

import androidx.compose.runtime.LaunchedEffect

import androidx.compose.runtime.getValue

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.graphics.Color

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.navigation.NavController

import com.airbnb.lottie.compose.LottieAnimation

import com.airbnb.lottie.compose.LottieCompositionSpec

import com.airbnb.lottie.compose.animateLottieCompositionAsState

import com.airbnb.lottie.compose.rememberLottieComposition

import com.students.tastyfood.R

import kotlinx.coroutines.delay

@Composable

fun SplashScreen(navController: NavController) {

val composition by rememberLottieComposition(LottieCompositionSpec.RawRes(R.raw.food))

val progress by animateLottieCompositionAsState(composition, iterations = 1)

LaunchedEffect(progress) {

if (progress == 1f) {

delay(30)

navController.navigate("home") {

popUpTo(0)

}

# 

}

}

Box(

modifier = Modifier

.fillMaxSize()

.background(Color(0xFFFFFFFF)),

contentAlignment = Alignment.Center

) {

Column(horizontalAlignment = Alignment.CenterHorizontally) {

LottieAnimation(

composition = composition,

progress = { progress },

modifier = Modifier.size(200.dp)

)

Spacer(modifier = Modifier.height(220.dp))

Text(

text = "Cook Like a Chef\nTasty Food @2025",

color = Color(0xFF4F4F4F),

fontSize = 16.sp,

style = MaterialTheme.typography.headlineMedium

)

}

}

}

package com.students.tastyfood.ui.theme

import androidx.compose.ui.graphics.Color

val Purple80 = Color(0xFFD0BCFF)

val PurpleGrey80 = Color(0xFFCCC2DC)

val Pink80 = Color(0xFFEFB8C8)

# 

val Purple40 = Color(0xFF6650a4)

val PurpleGrey40 = Color(0xFF625b71)

val Pink40 = Color(0xFF7D5260)

val TextColor = Color(0xFF3C444C)

val PastelBg = Color(0xFFEEE8E2)

val PastelPink = Color(0xFFAB6606)

val PastelBlue = Color(0xFF3C444C)

val PastelYellow = Color(0xFFE2D5A3)

val PastelGreen = Color(0xFFB8E4C9)

val PastelPurple = Color(0xFFD0BCFF)

val PastelOrange = Color(0xFFFFCBA4)

val PastelRed = Color(0xFFF96163)

val PastelTeal = Color(0xFFB8E4C9)

val PastelCyan = Color(0xFF3C444C)

val PastelLime = Color(0xFFE2D5A3)

package com.students.tastyfood.ui.theme

import android.app.Activity

import android.os.Build

import androidx.compose.foundation.isSystemInDarkTheme

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.darkColorScheme

import androidx.compose.material3.dynamicDarkColorScheme

import androidx.compose.material3.dynamicLightColorScheme

import androidx.compose.material3.lightColorScheme

import androidx.compose.runtime.Composable

import androidx.compose.ui.platform.LocalContext

private val DarkColorScheme = darkColorScheme(

primary = Purple80,

secondary = PurpleGrey80,

# 

tertiary = Pink80

)

private val LightColorScheme = lightColorScheme(

primary = Purple40,

secondary = PurpleGrey40,

tertiary = Pink40

/\* Other default colors to override

background = Color(0xFFFFFBFE),

surface = Color(0xFFFFFBFE),

onPrimary = Color.White,

onSecondary = Color.White,

onTertiary = Color.White,

onBackground = Color(0xFF1C1B1F),

onSurface = Color(0xFF1C1B1F),

\*/

)

@Composable

fun TastyFoodTheme(

darkTheme: Boolean = isSystemInDarkTheme(),

// Dynamic color is available on Android 12+

dynamicColor: Boolean = false,

content: @Composable () -> Unit

) {

val colorScheme = when {

dynamicColor && Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S -> {

val context = LocalContext.current

if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)

# 

}

darkTheme -> DarkColorScheme

else -> LightColorScheme

}

MaterialTheme(

colorScheme = colorScheme,

typography = Typography,

content = content

)

}

package com.students.tastyfood.ui.theme

import androidx.compose.material3.Typography

import androidx.compose.ui.text.TextStyle

import androidx.compose.ui.text.font.FontFamily

import androidx.compose.ui.text.font.FontWeight

import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with

val Typography = Typography(

bodyLarge = TextStyle(

fontFamily = FontFamily.Default,

fontWeight = FontWeight.Normal,

fontSize = 16.sp,

lineHeight = 24.sp,

letterSpacing = 0.5.sp

)

/\* Other default text styles to override

titleLarge = TextStyle(

fontFamily = FontFamily.Default,

fontWeight = FontWeight.Normal,

# 

fontSize = 22.sp,

lineHeight = 28.sp,

letterSpacing = 0.sp

),

labelSmall = TextStyle(

fontFamily = FontFamily.Default,

fontWeight = FontWeight.Medium,

fontSize = 11.sp,

lineHeight = 16.sp,

letterSpacing = 0.5.sp

)

\*/

)

package com.students.tastyfood.util

import androidx.room.TypeConverter

import com.google.gson.Gson

import com.google.gson.reflect.TypeToken

import com.students.tastyfood.model.RecipeStep

class Converters {

@TypeConverter

fun fromString(value: String?): List<String> {

return value?.split("||")?.filter { it.isNotEmpty() } ?: emptyList()

}

@TypeConverter

fun listToString(list: List<String>?): String {

return list?.joinToString("||") ?: ""

}

@TypeConverter

fun fromRecipeStepList(list: List<RecipeStep>?): String {

# 

return Gson().toJson(list)

}

@TypeConverter

fun toRecipeStepList(json: String?): List<RecipeStep> {

if (json.isNullOrEmpty()) return emptyList()

val type = object : TypeToken<List<RecipeStep>>() {}.type

return Gson().fromJson(json, type)

}

}

package com.students.tastyfood.util

import com.google.gson.Gson

import com.google.gson.reflect.TypeToken

import com.students.tastyfood.model.RecipeStep

object RecipeStepSerializer {

private val gson = Gson()

fun toJson(steps: List<RecipeStep>): String = gson.toJson(steps)

fun fromJson(json: String): List<RecipeStep> =

gson.fromJson(json, object : TypeToken<List<RecipeStep>>() {}.type)

}

package com.students.tastyfood.viewmodel

import android.util.Log

import androidx.lifecycle.ViewModel

import androidx.lifecycle.viewModelScope

import com.students.tastyfood.data.local.dao.RecipeDao

import com.students.tastyfood.data.local.entity.RecipeEntity

import com.students.tastyfood.data.remote.MealDto

import com.students.tastyfood.data.remote.TheMealDbRepository

import kotlinx.coroutines.flow.\*

import kotlinx.coroutines.launch

# 

class RecipeViewModel(private val recipeDao: RecipeDao) : ViewModel() { val recipes: StateFlow<List<RecipeEntity>> = recipeDao.getAllRecipes()

.stateIn(viewModelScope, SharingStarted.WhileSubscribed(5000), emptyList())

private val theMealDbRepository = TheMealDbRepository()

private val \_randomMeals = MutableStateFlow<List<MealDto>>(emptyList())

val randomMeals: StateFlow<List<MealDto>> = \_randomMeals

fun toggleFavorite(recipe: RecipeEntity) {

viewModelScope.launch {

recipeDao.updateRecipe(recipe.copy(isFavorite = !recipe.isFavorite))

}

}

fun getRecipeById(id: Int): Flow<RecipeEntity?> {

return flow {

emit(recipeDao.getRecipeById(id))

}

}

fun insertRecipe(recipe: RecipeEntity) {

viewModelScope.launch {

recipeDao.insertRecipe(recipe)

}

}

fun loadRandomMeals(count: Int = 2) {

println("loadRandomMeals called")

viewModelScope.launch {

try {

Log.d("RecipeViewModel", "Запуск загрузки сетевых рецептов")

# 

val meals = theMealDbRepository.getSomeMealsFromCategory("Chicken", count)

Log.d("RecipeViewModel", "Загружено сетевых рецептов: ${meals.size}")

\_randomMeals.value = meals

} catch (e: Exception) {

Log.e("RecipeViewModel", "Ошибка загрузки сетевых рецептов", e)

println("Ошибка загрузки сетевых рецептов: ${e.message}")

}

}

}

}

package com.students.tastyfood.viewmodel

import androidx.lifecycle.ViewModel

import androidx.lifecycle.ViewModelProvider

import com.students.tastyfood.data.local.dao.RecipeDao

class RecipeViewModelFactory(private val recipeDao: RecipeDao) : ViewModelProvider.Factory {

override fun <T : ViewModel> create(modelClass: Class<T>): T {

if (modelClass.isAssignableFrom(RecipeViewModel::class.java)) {

@Suppress("UNCHECKED\_CAST")

return RecipeViewModel(recipeDao) as T

}

throw IllegalArgumentException("Unknown ViewModel class")

}

}

package com.students.tastyfood.viewmodel

# 

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableFloatStateOf

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.setValue

import androidx.lifecycle.ViewModel

class SettingsViewModel : ViewModel() {

var isDarkTheme by mutableStateOf(false)

var fontSize by mutableFloatStateOf(16f)

var isEnglish by mutableStateOf(false)

}

package com.students.tastyfood

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.activity.viewModels

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.material3.MaterialTheme

import androidx.compose.material3.Surface

import androidx.compose.ui.Modifier

import com.students.tastyfood.ui.navigation.NavGraph

import com.students.tastyfood.ui.theme.TastyFoodTheme

import com.students.tastyfood.viewmodel.SettingsViewModel

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

val settingsViewModel: SettingsViewModel by viewModels()

setContent {

TastyFoodTheme(darkTheme = settingsViewModel.isDarkTheme) {

Surface(

# 

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

NavGraph(settingsViewModel = settingsViewModel)

}

}

}

}

}

package com.students.tastyfood

import androidx.test.platform.app.InstrumentationRegistry

import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test

import org.junit.runner.RunWith

import org.junit.Assert.\*

/\*\*

\* Instrumented test, which will execute on an Android device.

\*

\* See [testing documentation](http://d.android.com/tools/testing).

\*/

@RunWith(AndroidJUnit4::class)

class ExampleInstrumentedTest {

@Test

fun useAppContext() {

// Context of the app under test.

val appContext = InstrumentationRegistry.getInstrumentation().targetContext

assertEquals("com.students.tastyfood", appContext.packageName)

}

# 

}

package com.students.tastyfood

import org.junit.Test

import org.junit.Assert.\*

/\*\*

\* Example local unit test, which will execute on the development machine (host).

\*

\* See [testing documentation](http://d.android.com/tools/testing).

\*/

class ExampleUnitTest {

@Test

fun addition\_isCorrect() {

assertEquals(4, 2 + 2)

}

}

package com.students.tastyfood.`data`.local.dao

import androidx.room.EntityDeleteOrUpdateAdapter

import androidx.room.EntityInsertAdapter

import androidx.room.RoomDatabase

import androidx.room.coroutines.createFlow

import androidx.room.util.getColumnIndexOrThrow

import androidx.room.util.performSuspending

import androidx.sqlite.SQLiteStatement

import com.students.tastyfood.`data`.local.entity.RecipeEntity

import com.students.tastyfood.model.RecipeStep

import com.students.tastyfood.util.Converters

import javax.`annotation`.processing.Generated

import kotlin.Boolean

# 

import kotlin.Float

import kotlin.Int

import kotlin.String

import kotlin.Suppress

import kotlin.Unit

import kotlin.collections.List

import kotlin.collections.MutableList

import kotlin.collections.mutableListOf

import kotlin.reflect.KClass

import kotlinx.coroutines.flow.Flow

@Generated(value = ["androidx.room.RoomProcessor"])

@Suppress(names = ["UNCHECKED\_CAST", "DEPRECATION", "REDUNDANT\_PROJECTION", "REMOVAL"])

public class RecipeDao\_Impl(

\_\_db: RoomDatabase,

) : RecipeDao {

private val \_\_db: RoomDatabase

private val \_\_insertAdapterOfRecipeEntity: EntityInsertAdapter<RecipeEntity>

private val \_\_converters: Converters = Converters()

private val \_\_deleteAdapterOfRecipeEntity: EntityDeleteOrUpdateAdapter<RecipeEntity>

private val \_\_updateAdapterOfRecipeEntity: EntityDeleteOrUpdateAdapter<RecipeEntity>

init {

this.\_\_db = \_\_db

this.\_\_insertAdapterOfRecipeEntity = object : EntityInsertAdapter<RecipeEntity>() {

protected override fun createQuery(): String =

# 

"INSERT OR REPLACE INTO `recipes` (`id`,`title`,`imageUrl`,`cookingTime`,`difficulty`,`description`,`rating`,`isFavorite`,`ingredients`,`descriptionMedia`,`category`,`steps`) VALUES (nullif(?, 0),?,?,?,?,?,?,?,?,?,?,?)"

protected override fun bind(statement: SQLiteStatement, entity: RecipeEntity) {

statement.bindLong(1, entity.id.toLong())

statement.bindText(2, entity.title)

val \_tmpImageUrl: String? = entity.imageUrl

if (\_tmpImageUrl == null) {

statement.bindNull(3)

} else {

statement.bindText(3, \_tmpImageUrl)

}

statement.bindText(4, entity.cookingTime)

statement.bindLong(5, entity.difficulty.toLong())

statement.bindText(6, entity.description)

statement.bindDouble(7, entity.rating.toDouble())

val \_tmp: Int = if (entity.isFavorite) 1 else 0

statement.bindLong(8, \_tmp.toLong())

val \_tmp\_1: String = \_\_converters.listToString(entity.ingredients)

statement.bindText(9, \_tmp\_1)

val \_tmp\_2: String = \_\_converters.listToString(entity.descriptionMedia)

statement.bindText(10, \_tmp\_2)

statement.bindText(11, entity.category)

val \_tmp\_3: String = \_\_converters.fromRecipeStepList(entity.steps)

statement.bindText(12, \_tmp\_3)

}

}

# 

this.\_\_deleteAdapterOfRecipeEntity = object : EntityDeleteOrUpdateAdapter<RecipeEntity>() {

protected override fun createQuery(): String = "DELETE FROM `recipes` WHERE `id` = ?"

protected override fun bind(statement: SQLiteStatement, entity: RecipeEntity) {

statement.bindLong(1, entity.id.toLong())

}

}

this.\_\_updateAdapterOfRecipeEntity = object : EntityDeleteOrUpdateAdapter<RecipeEntity>() {

protected override fun createQuery(): String =

"UPDATE OR ABORT recipes SET id = ?,`title` = ?,`imageUrl` = ?,`cookingTime` = ?,`difficulty` = ?,`description` = ?,`rating` = ?,`isFavorite` = ?,`ingredients` = ?,`descriptionMedia` = ?,`category` = ?,`steps` = ? WHERE id = ?"

protected override fun bind(statement: SQLiteStatement, entity: RecipeEntity) {

statement.bindLong(1, entity.id.toLong())

statement.bindText(2, entity.title)

val \_tmpImageUrl: String? = entity.imageUrl

if (\_tmpImageUrl == null) {

statement.bindNull(3)

} else {

statement.bindText(3, \_tmpImageUrl)

}

statement.bindText(4, entity.cookingTime)

statement.bindLong(5,

ement.bindLong

# 

(5, entity.difficulty.toLong())

statement.bindText(6, entity.description)

statement.bindDouble(7, entity.rating.toDouble())

val \_tmp: Int = if (entity.isFavorite) 1 else 0

statement.bindLong(8, \_tmp.toLong())

val \_tmp\_1: String = \_\_converters.listToString(entity.ingredients)

statement.bindText(9, \_tmp\_1)

val \_tmp\_2: String = \_\_converters.listToString(entity.descriptionMedia)

statement.bindText(10, \_tmp\_2)

statement.bindText(11, entity.category)

val \_tmp\_3: String = \_\_converters.fromRecipeStepList(entity.steps)

statement.bindText(12, \_tmp\_3)

statement.bindLong(13, entity.id.toLong())

}

}

}

public override suspend fun insertRecipe(recipe: RecipeEntity): Unit = performSuspending(\_\_db,

false, true) { \_connection ->

\_\_insertAdapterOfRecipeEntity.insert(\_connection, recipe)

}

public override suspend fun deleteRecipe(recipe: RecipeEntity): Unit = performSuspending(\_\_db,

false, true) { \_connection ->

\_\_deleteAdapterOfRecipeEntity.handle(\_connection, recipe)

}

public override suspend fun updateRecipe(recipe: RecipeEntity): Unit = performSuspending(\_\_db,

false, true) { \_connection ->

# 

\_\_updateAdapterOfRecipeEntity.handle(\_connection, recipe)

}

public override fun getAllRecipes(): Flow<List<RecipeEntity>> {

val \_sql: String = "SELECT \* FROM recipes ORDER BY id DESC"

return createFlow(\_\_db, false, arrayOf("recipes")) { \_connection ->

val \_stmt: SQLiteStatement = \_connection.prepare(\_sql)

try {

val \_columnIndexOfId: Int = getColumnIndexOrThrow(\_stmt, "id")

val \_columnIndexOfTitle: Int = getColumnIndexOrThrow(\_stmt, "title")

val \_columnIndexOfImageUrl: Int = getColumnIndexOrThrow(\_stmt, "imageUrl")

val \_columnIndexOfCookingTime: Int = getColumnIndexOrThrow(\_stmt, "cookingTime")

val \_columnIndexOfDifficulty: Int = getColumnIndexOrThrow(\_stmt, "difficulty")

val \_columnIndexOfDescription: Int = getColumnIndexOrThrow(\_stmt, "description")

val \_columnIndexOfRating: Int = getColumnIndexOrThrow(\_stmt, "rating")

val \_columnIndexOfIsFavorite: Int = getColumnIndexOrThrow(\_stmt, "isFavorite")

val \_columnIndexOfIngredients: Int = getColumnIndexOrThrow(\_stmt, "ingredients")

val \_columnIndexOfDescriptionMedia: Int = getColumnIndexOrThrow(\_stmt, "descriptionMedia")

val \_columnIndexOfCategory: Int = getColumnIndexOrThrow(\_stmt, "category")

val \_columnIndexOfSteps: Int = getColumnIndexOrThrow(\_stmt, "steps")

val \_result: MutableList<RecipeEntity> = mutableListOf()

# 

while (\_stmt.step()) {

val \_item: RecipeEntity

val \_tmpId: Int

\_tmpId = \_stmt.getLong(\_columnIndexOfId).toInt()

val \_tmpTitle: String

\_tmpTitle = \_stmt.getText(\_columnIndexOfTitle)

val \_tmpImageUrl: String?

if (\_stmt.isNull(\_columnIndexOfImageUrl)) {

\_tmpImageUrl = null

} else {

\_tmpImageUrl = \_stmt.getText(\_columnIndexOfImageUrl)

}

val \_tmpCookingTime: String

\_tmpCookingTime = \_stmt.getText(\_columnIndexOfCookingTime)

val \_tmpDifficulty: Int

\_tmpDifficulty = \_stmt.getLong(\_columnIndexOfDifficulty).toInt()

val \_tmpDescription: String

\_tmpDescription = \_stmt.getText(\_columnIndexOfDescription)

val \_tmpRating: Float

\_tmpRating = \_stmt.getDouble(\_columnIndexOfRating).toFloat()

val \_tmpIsFavorite: Boolean

val \_tmp: Int

\_tmp = \_stmt.getLong(\_columnIndexOfIsFavorite).toInt()

\_tmpIsFavorite = \_tmp != 0

val \_tmpIngredients: List<String>

val \_tmp\_1: String

\_tmp\_1 = \_stmt.getText(\_columnIndexOfIngredients)

\_tmpIngredients = \_\_converters.fromString(\_tmp\_1)

val \_tmpDescriptionMedia: List<String>

# 

val \_tmp\_2: String

\_tmp\_2 = \_stmt.getText(\_columnIndexOfDescriptionMedia)

\_tmpDescriptionMedia = \_\_converters.fromString(\_

tmp\_2)

val \_tmpCategory: String

\_tmpCategory = \_stmt.getText(\_columnIndexOfCategory)

val \_tmpSteps: List<RecipeStep>

val \_tmp\_3: String

\_tmp\_3 = \_stmt.getText(\_columnIndexOfSteps)

\_tmpSteps = \_\_converters.toRecipeStepList(\_tmp\_3)

\_item = RecipeEntity(\_tmpId,\_tmpTitle,\_tmpImageUrl,\_tmpCookingTime,\_tmpDifficulty,\_tmpDescription,\_tmpRating,\_tmpIsFavorite,\_tmpIngredients,\_tmpDescriptionMedia,\_tmpCategory,\_tmpSteps)

\_result.add(\_item)

}

\_result

} finally {

\_stmt.close()

}

}

}

public override fun getFavoriteRecipes(): Flow<List<RecipeEntity>> {

val \_sql: String = "SELECT \* FROM recipes WHERE isFavorite = 1"

return createFlow(\_\_db, false, arrayOf("recipes")) { \_connection ->

val \_stmt: SQLiteStatement = \_connection.prepare(\_sql)

try {

val \_columnIndexOfId: Int = getColumnIndexOrThrow(\_stmt, "id")

val \_columnIndexOfTitle: Int = getColumnIndexOrThrow(\_stmt, "title")

# 

val \_columnIndexOfImageUrl: Int = getColumnIndexOrThrow(\_stmt, "imageUrl")

val \_columnIndexOfCookingTime: Int = getColumnIndexOrThrow(\_stmt, "cookingTime")

val \_columnIndexOfDifficulty: Int = getColumnIndexOrThrow(\_stmt, "difficulty")

val \_columnIndexOfDescription: Int = getColumnIndexOrThrow(\_stmt, "description")

val \_columnIndexOfRating: Int = getColumnIndexOrThrow(\_stmt, "rating")

val \_columnIndexOfIsFavorite: Int = getColumnIndexOrThrow(\_stmt, "isFavorite")

val \_columnIndexOfIngredients: Int = getColumnIndexOrThrow(\_stmt, "ingredients")

val \_columnIndexOfDescriptionMedia: Int = getColumnIndexOrThrow(\_stmt, "descriptionMedia")

val \_columnIndexOfCategory: Int = getColumnIndexOrThrow(\_stmt, "category")

val \_columnIndexOfSteps: Int = getColumnIndexOrThrow(\_stmt, "steps")

val \_result: MutableList<RecipeEntity> = mutableListOf()

while (\_stmt.step()) {

val \_item: RecipeEntity

val \_tmpId: Int

\_tmpId = \_stmt.getLong(\_columnIndexOfId).toInt()

val \_tmpTitle: String

\_tmpTitle = \_stmt.getText(\_columnIndexOfTitle)

val \_tmpImageUrl: String?

if (\_stmt.isNull(\_columnIndexOfImageUrl)) {

\_tmpImageUrl = null

# 

} else {

\_tmpImageUrl = \_stmt.getText(\_columnIndexOfImageUrl)

}

val \_tmpCookingTime: String

\_tmpCookingTime = \_stmt.getText(\_columnIndexOfCookingTime)

val \_tmpDifficulty: Int

\_tmpDifficulty = \_stmt.getLong(\_columnIndexOfDifficulty).toInt()

val \_tmpDescription: String

\_tmpDescription = \_stmt.getText(\_columnIndexOfDescription)

val \_tmpRating: Float

\_tmpRating = \_stmt.getDouble(\_columnIndexOfRating).toFloat()

val \_tmpIsFavorite: Boolean

val \_tmp: Int

\_tmp = \_stmt.getLong(\_columnIndexOfIsFavorite).toInt()

\_tmpIsFavorite = \_tmp != 0

val \_tmpIngredients: List<String>

val \_tmp\_1: String

\_tmp\_1 = \_stmt.getText(\_columnIndexOfIngredients)

\_tmpIngredients = \_\_converters.fromString(\_tmp\_1)

val \_tmpDescriptionMedia: List<String>

val \_tmp\_2: String

\_tmp\_2 = \_stmt.getText(\_columnIndexOfDescriptionMedia)

\_tmpDescriptionMedia = \_\_converters.fromString(\_tmp\_2)

val \_tmpCategory: String

\_tmpCategory = \_stmt.getText(\_columnIndexOfCategory)

val \_tmpSteps: List<RecipeStep>

val \_tmp\_3: String

\_tmp\_3 = \_stmt.getText(\_columnIndexOfSteps)

\_tmpSteps = \_\_converters.toRecipeStepList(\_tmp\_3)

# 

\_item =

RecipeEntity(\_tmpId,\_tmpTitle,\_tmpImageUrl,\_tmpCookingTime,\_tmpDifficulty,\_tmpDescription,\_tmpRating,\_tmpIsFavorite,\_tmpIngredients,\_tmpDescriptionMedia,\_tmpCategory,\_tmpSteps)

\_result.add(\_item)

}

\_result

} finally {

\_stmt.close()

}

}

}

public override suspend fun getRecipeById(id: Int): RecipeEntity?

{

val \_sql: String = "SELECT \* FROM recipes WHERE id = ?"

return performSuspending(\_\_db, true, false) { \_connection ->

val \_stmt: SQLiteStatement = \_connection.prepare(\_sql)

try {

var \_argIndex: Int = 1

\_stmt.bindLong(\_argIndex, id.toLong())

val \_columnIndexOfId: Int = getColumnIndexOrThrow(\_stmt, "id")

val \_columnIndexOfTitle: Int = getColumnIndexOrThrow(\_stmt, "title")

val \_columnIndexOfImageUrl: Int = getColumnIndexOrThrow(\_stmt, "imageUrl")

val \_columnIndexOfCookingTime: Int = getColumnIndexOrThrow(\_stmt, "cookingTime")

val \_columnIndexOfDifficulty: Int = getColumnIndexOrThrow(\_stmt, "difficulty")

# 

val \_columnIndexOfDescription: Int = getColumnIndexOrThrow(\_stmt, "description")

val \_columnIndexOfRating: Int = getColumnIndexOrThrow(\_stmt, "rating")

val \_columnIndexOfIsFavorite: Int = getColumnIndexOrThrow(\_stmt, "isFavorite")

val \_columnIndexOfIngredients: Int = getColumnIndexOrThrow(\_stmt, "ingredients")

val \_columnIndexOfDescriptionMedia: Int = getColumnIndexOrThrow(\_stmt, "descriptionMedia")

val \_columnIndexOfCategory: Int = getColumnIndexOrThrow(\_stmt, "category")

val \_columnIndexOfSteps: Int = getColumnIndexOrThrow(\_stmt, "steps")

val \_result: RecipeEntity?

if (\_stmt.step()) {

val \_tmpId: Int

\_tmpId = \_stmt.getLong(\_columnIndexOfId).toInt()

val \_tmpTitle: String

\_tmpTitle = \_stmt.getText(\_columnIndexOfTitle)

val \_tmpImageUrl: String?

if (\_stmt.isNull(\_columnIndexOfImageUrl)) {

\_tmpImageUrl = null

} else {

\_tmpImageUrl = \_stmt.getText(\_columnIndexOfImageUrl)

}

val \_tmpCookingTime: String

\_tmpCookingTime = \_stmt.getText(\_columnIndexOfCookingTime)

val \_tmpDifficulty: Int

\_tmpDifficulty = \_stmt.getLong(\_columnIndexOfDifficulty).toInt()

# 

val \_tmpDescription: String

\_tmpDescription = \_stmt.getText(\_columnIndexOfDescription)

val \_tmpRating: Float

\_tmpRating = \_stmt.getDouble(\_columnIndexOfRating).toFloat()

val \_tmpIsFavorite: Boolean

val \_tmp: Int

\_tmp = \_stmt.getLong(\_columnIndexOfIsFavorite).toInt()

\_tmpIsFavorite = \_tmp != 0

val \_tmpIngredients: List<String>

val \_tmp\_1: String

\_tmp\_1 = \_stmt.getText(\_columnIndexOfIngredients)

\_tmpIngredients = \_\_converters.fromString(\_tmp\_1)

val \_tmpDescriptionMedia: List<String>

val \_tmp\_2: String

\_tmp\_2 = \_stmt.getText(\_columnIndexOfDescriptionMedia)

\_tmpDescriptionMedia = \_\_converters.fromString(\_tmp\_2)

val \_tmpCategory: String

\_tmpCategory = \_stmt.getText(\_columnIndexOfCategory)

val \_tmpSteps: List<RecipeStep>

val \_tmp\_3: String

\_tmp\_3 = \_stmt.getText(\_columnIndexOfSteps)

\_tmpSteps = \_\_converters.toRecipeStepList(\_tmp\_3)

\_result = RecipeEntity(\_tmpId,\_tmpTitle,\_tmpImageUrl,\_tmpCookingTime,\_tmpDifficulty,\_tmpDescription,\_tmpRating,\_tmpIsFavorite,\_tmpIngredients,\_tmpDescriptionMedia,\_tmpCategory,\_tmpSteps)

} else {

\_result = null

}

# 

\_result

} finally {

\_stmt.close()

}

}

}

public override suspend fun clearAll() {

val \_sql: String = "DELETE FROM recipes"

return performSuspending(\_\_db, false, true) { \_connection ->

val \_stmt: SQLiteStatement = \_connection.prepare(\_sql)

try {

\_stmt.step()

} finally {

\_stmt.close()

}

}

}

public companion object {

public fun getRequiredConverters(): List<KClass<\*>> = emptyList()

}

}

package com.students.tastyfood.`data`.local.db

import androidx.room.InvalidationTracker

import androidx.room.RoomOpenDelegate

import androidx.room.migration.AutoMigrationSpec

import androidx.room.migration.Migration

import androidx.room.util.TableInfo

import androidx.room.util.TableInfo.Companion.read

import androidx.room.util.dropFtsSyncTriggers

# 

import androidx.sqlite.SQLiteConnection

import androidx.sqlite.execSQL

import com.students.tastyfood.`data`.local.dao.RecipeDao

import com.students.tastyfood.`data`.local.dao.RecipeDao\_Impl

import javax.`annotation`.processing.Generated

import kotlin.Lazy

import kotlin.String

import kotlin.Suppress

import kotlin.collections.List

import kotlin.collections.Map

import kotlin.collections.MutableList

import kotlin.collections.MutableMap

import kotlin.collections.MutableSet

import kotlin.collections.Set

import kotlin.collections.mutableListOf

import kotlin.collections.mutableMapOf

import kotlin.collections.mutableSetOf

import kotlin.reflect.KClass

@Generated(value = ["androidx.room.RoomProcessor"])

@Suppress(names = ["UNCHECKED\_CAST", "DEPRECATION", "REDUNDANT\_PROJECTION", "REMOVAL"])

public class RecipeDatabase\_Impl : RecipeDatabase() {

private val \_recipeDao: Lazy<RecipeDao> = lazy {

RecipeDao\_Impl(this)

}

protected override fun createOpenDelegate(): RoomOpenDelegate {

val \_openDelegate: RoomOpenDelegate = object : RoomOpenDelegate(3,

"d381f4341ad2d8fc50dfef1f547d7308", "5a27caa88901b7a389f5046f391187df") {

# 

public override fun createAllTables(connection: SQLiteConnection) {

connection.execSQL("CREATE TABLE IF NOT EXISTS `recipes` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `title` TEXT NOT NULL, `imageUrl` TEXT, `cookingTime` TEXT NOT NULL, `difficulty` INTEGER NOT NULL, `description` TEXT NOT NULL, `rating` REAL NOT NULL, `isFavorite` INTEGER NOT NULL, `ingredients` TEXT NOT NULL, `descriptionMedia` TEXT NOT NULL, `category` TEXT NOT NULL, `steps` TEXT NOT NULL)")

connection.execSQL("CREATE TABLE IF NOT EXISTS room\_master\_table (id INTEGER PRIMARY KEY,identity\_hash TEXT)")

connection.execSQL("INSERT OR REPLACE INTO room\_master\_table (id,identity\_hash) VALUES(42, 'd381f4341ad2d8fc50dfef1f547d7308')")

}

public override fun dropAllTables(connection: SQLiteConnection) {

connection.execSQL("DROP TABLE IF EXISTS `recipes`")

}

public override fun onCreate(connection: SQLiteConnection) {

}

public override fun onOpen(connection: SQLiteConnection) {

internalInitInvalidationTracker(connection)

}

public override fun onPreMigrate(connection: SQLiteConnection) {

dropFtsSyncTriggers(connection)

}

public override fun onPostMigrate(connection: SQLiteConnection) {

}

public override fun onValidateSchema(connection: SQLiteConnection):

RoomOpenDelegate.ValidationResult {

# 

val \_columnsRecipes: MutableMap<String, TableInfo.Column> = mutableMapOf()

\_columnsRecipes.put("id", TableInfo.Column("id", "INTEGER", true, 1, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("title", TableInfo.Column("title", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("imageUrl", TableInfo.Column("imageUrl", "TEXT", false, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("cookingTime", TableInfo.Column("cookingTime", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("difficulty", TableInfo.Column("difficulty", "INTEGER", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("description", TableInfo.Column("description", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("rating", TableInfo.Column("rating", "REAL", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("isFavorite", TableInfo.Column("isFavorite", "INTEGER", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("ingredients", TableInfo.Column("ingredients", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

# 

\_columnsRecipes.put("descriptionMedia", TableInfo.Column("descriptionMedia", "TEXT", true,

0, null, TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("category", TableInfo.Column("category", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

\_columnsRecipes.put("steps", TableInfo.Column("steps", "TEXT", true, 0, null,

TableInfo.CREATED\_FROM\_ENTITY))

val \_foreignKeysRecipes: MutableSet<TableInfo.ForeignKey> = mutableSetOf()

val \_indicesRecipes: MutableSet<TableInfo.Index> = mutableSetOf()

val \_infoRecipes: TableInfo = TableInfo("recipes", \_columnsRecipes, \_foreignKeysRecipes,

\_indicesRecipes)

val \_existingRecipes: TableInfo = read(connection, "recipes")

if (!\_infoRecipes.equals(\_existingRecipes)) {

return RoomOpenDelegate.ValidationResult(false, """

|recipes(com.students.tastyfood.data.local.entity.RecipeEntity).

| Expected:

|""".trimMargin() + \_infoRecipes + """

|

| Found:

|""".trimMargin() + \_existingRecipes)

}

return RoomOpenDelegate.ValidationResult(true, null)

}

}

return \_openDelegate

# 

}

protected override fun createInvalidationTracker(): InvalidationTracker {

val \_shadowTablesMap: MutableMap<String, String> = mutableMapOf()

val \_viewTables: MutableMap<String, Set<String>> = mutableMapOf()

return InvalidationTracker(this, \_shadowTablesMap, \_viewTables, "recipes")

}

public override fun clearAllTables() {

super.performClear(false, "recipes")

}

protected override fun getRequiredTypeConverterClasses(): Map<KClass<\*>, List<KClass<\*>>> {

val \_typeConvertersMap: MutableMap<KClass<\*>, List<KClass<\*>>> = mutableMapOf()

\_typeConvertersMap.put(RecipeDao::class, RecipeDao\_Impl.getRequiredConverters())

return \_typeConvertersMap

}

public override fun getRequiredAutoMigrationSpecClasses(): Set<KClass<out AutoMigrationSpec>> {

val \_autoMigrationSpecsSet: MutableSet<KClass<out AutoMigrationSpec>> = mutableSetOf()

return \_autoMigrationSpecsSet

}

public override

fun createAutoMigrations(autoMigrationSpecs: Map<KClass<out AutoMigrationSpec>, AutoMigrationSpec>):

List<Migration> {

val \_autoMigrations: MutableList<Migration> = mutableListOf()

return \_autoMigrations

# 

}

public override fun recipeDao(): RecipeDao = \_recipeDao.value

}