МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

Направление 02.03.02 Фундаментальная информатика и

информационные технологии

**Отчет**

по учебной практике (проектно-технологической)

Тема Разработка мобильного приложения «Герои Рик и Морти»

Отчетный семестр 6

7.07.2025 – 19.07.2025

Обучающийся 3 курс, 61 группа,

Вафин А.Р.

Руководитель преп. И.Д. Коток

Воронеж 2025

**Введение**

Существует множество различных языков программирования, таких как: Java, C++, C и так далее. Однако большой интерес представляют относительно молодые языки, так как их синтаксис подстроен под современные реалии программного обес печения и новых технологий в сфере мобильных приложений. Одним из таких языков программирования является Kotlin. Kotlin — это статически типизированный язык программирования (тип переменной известен во время компиляции, то есть еще до запуска программы). В отличие от Java, где программы строятся на классах, основным строи тельным блоком программы на Kotlin является функция. Однако Kotlin также поддерживает объектно-ориентированный подход к программированию. Целью данной работы является изучение теоретических основ языка программирования Kotlin и их практическая реализация.

Для достижения поставленной цели требуется решить следующие задачи:

— изучить основные компоненты языка и его синтаксис;

— разработать приложение "Герои Рик и Морти" на основе языка Kotlin в среде Android Studio

**Описание практических задач**

Приложение "Герои Рик и Морти" – это мобильное приложение, которое позволяет отслеживать актуальную информацию о героях одноименного мультсериала.

Данное приложение берёт информацию из открытого API <https://rickandmortyapi.com/documentation/#introduction> и постранично (используя пагинацию) кэширует её в память телефона для доступа без интернета, актуализируя её при каждом запуске приложения.

Также в приложении добавлены следующие особенности:

- возможность фильтрации геров по их свойствам

- возможность добавления героев в избранное (локально);

- возможность выбора героя для своего профиля (локально);

- возможность посмотреть подробную информацию о каждом герое;

- возможность изменять главный цвет приложения (локально);

- возможность просматривать код приложения (переход в браузер);

- возможность сообщать о багах (по Gmail);

- поддержка темной и светлой темы;

- обновление данных пользователем посредством свайпа сверху вниз.

В приложении 5 экранов:

- Экран с героями

- Экран героя

- Экран избранного

- Экран настроек

- Экран выбранного пользователем героя

**Теоретические сведения для решения задач**

В ходе написания приложения были изучены материалы и прослушаны лекции по следующим темам:

- Язык программирования Kotlin

- Основы программирования в среде Android Studio

- Основы сетевого взаимодействия, Retrofit

- Работа с локальной базой данных, Room

- Внедрение зависимостей c помощью Dagger-Hilt

- Архитектура мобильных приложения

- Написание пользовательского интерфейса с помощью Jetpack Compose

- Пагинация с помощью Paging 3

- Загрузка и кэширование картинок с помощью Coil

**Экран с героями**

На данном экране пользователю постранично (в виде бесконечной прокрутки) показываются все герои. Здесь можно добавить в избранное и выбрать героя для своего профиля, а также нажатием перейти на экран героя.

Ниже на рисунках 1 и 2 показывается загрузка данных и уже загруженные герои.

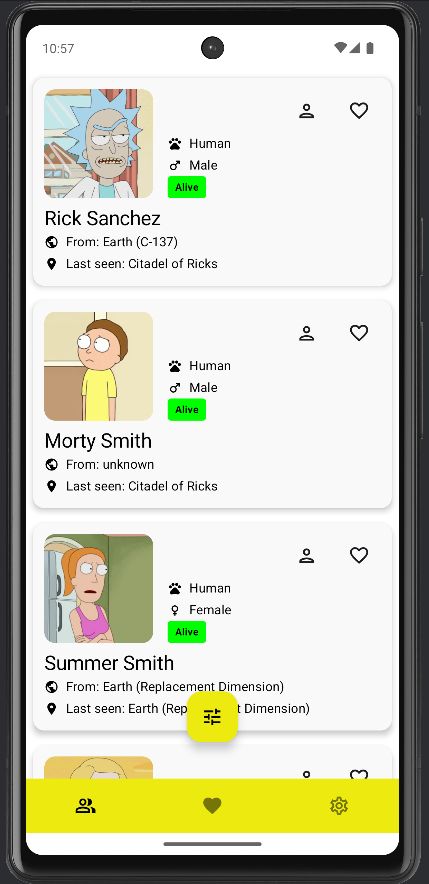


Рис 1. Загрузка данных Рис 2. Загруженные данные

Далее на рисунках 3 и 4 показывается, как меняется интерфейс при добавлении и избранное и выборе героя: закрашивается сердечко на этом герое, меняется иконка персонажа, а также появляется 4й пункт нижнего меню – профиль, в котором и можно увидеть выбранного персонажа.

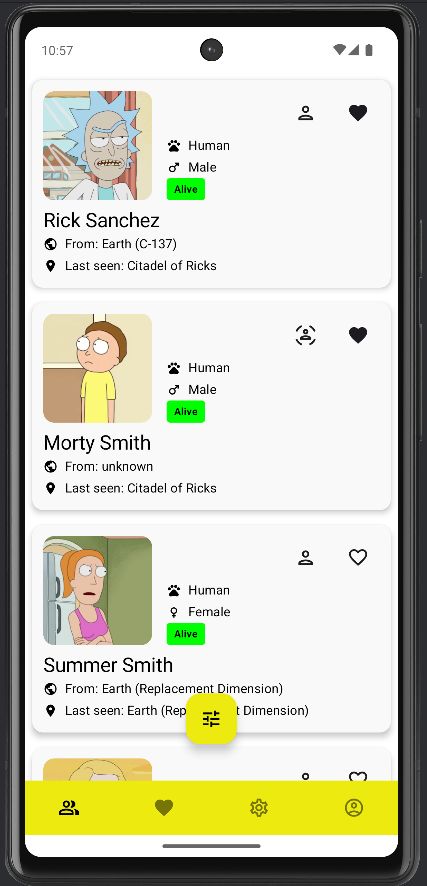
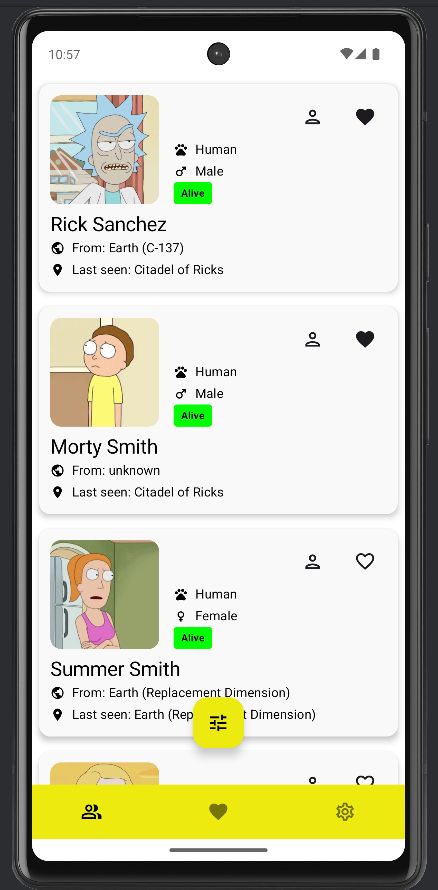


Рис 3. Персонаж в избранном Рис 4. Выбран герой для профиля

На рисунке 5 изображен выбор героя профиля и добавленные в избранное герои в темной теме.



Рис 5. Выбранный герой с избранным, темная тема

**Экран избранного**

На этом экране пользователю показываются герои, которых он добавил в избранное. Если оно пусто, показывается надпись «данных нет». Здесь также реализован постраничный вывод героев из базы данных, как на экране с героями.

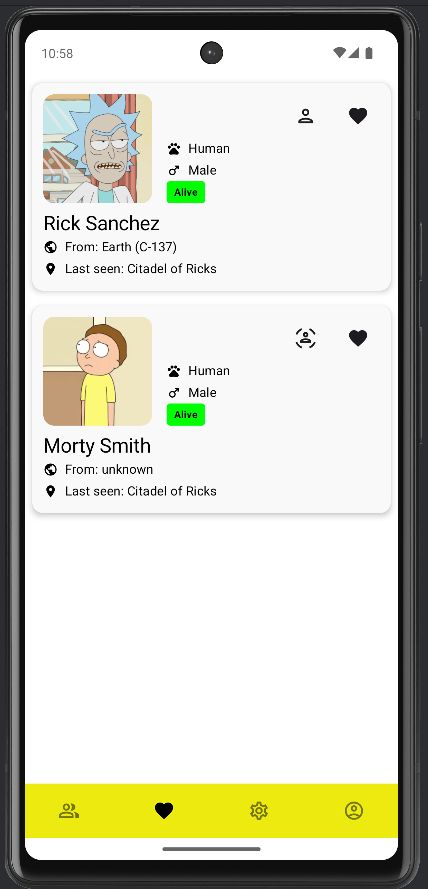


Рис 6. Пустое избранное Рис 7. Герои в избранном

**Экран настроек**

На этом экране пользователь может сделать следующее:

- изменить главный цвет приложения

- узнать версию приложения

- посмотреть исходный код приложения

- сообщить о баге

Ниже на рисунке 7 показан дизайн экрана настроек. На рисунке 8 показан процесс выбора нового главного цвета приложения. Главный цвет приложения выбирается отдельно для светлой и темной темы. Базовый главный цвет приложения – желтый. Новый цвет приложения сохраняется в памяти телефона и применяется, если выбран. Если не выбран, применяется базовый.

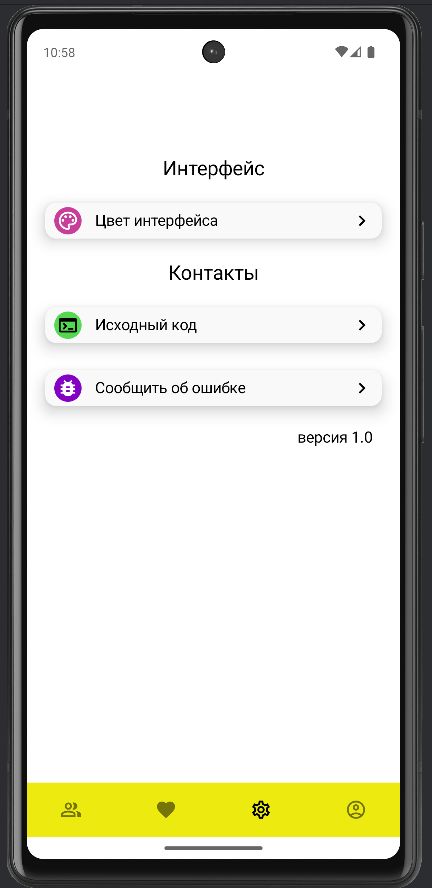
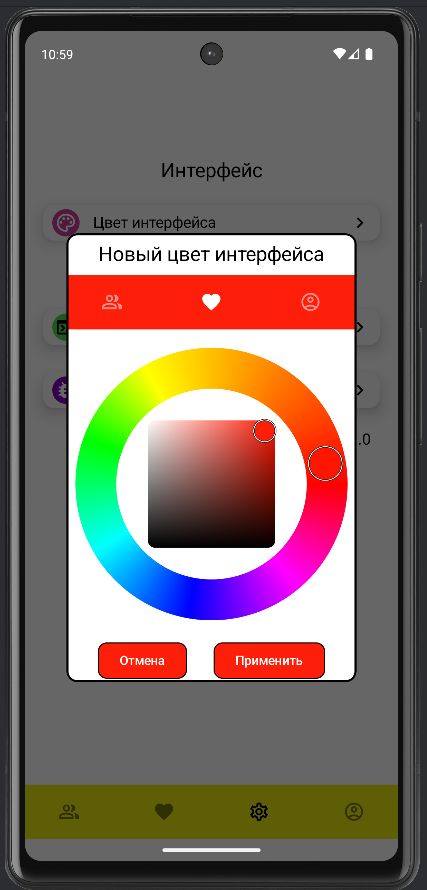


Рис 7. Экран настроек Рис 8. Изменение цвета

На рисунке 9 показан экран настроек после применения выбранного цвета.

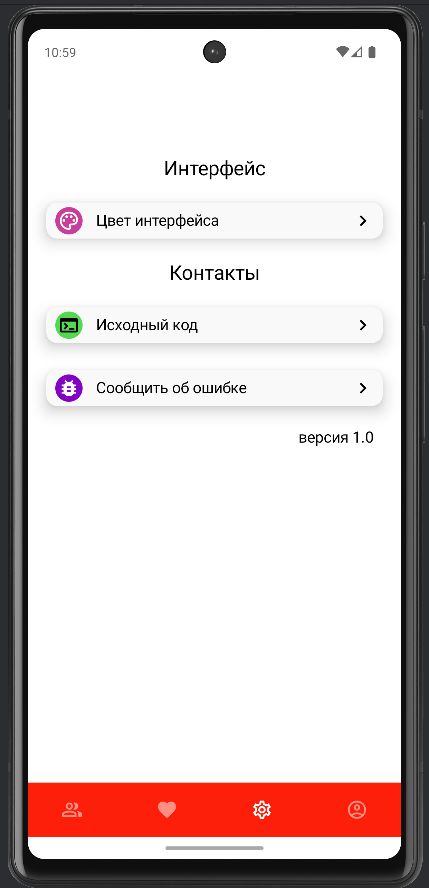


Рис 9. Примененный новый цвет.

Цвет иконок на панели навигации, которая показана в диалоге, выбирается автоматически в зависимости от люминанса (составляющей цвета, которая оценивает его яркость: если цвет относится к темным, имеющим люминанс меньше 0.4, то иконки становятся белыми, если к светлым, имеющим люминанс 0.4 и больше, черными).

**Экран выбранного пользователем героя**

На этом экране показывается вся информация о герое, которая приходит с API, и загружается его картинка.

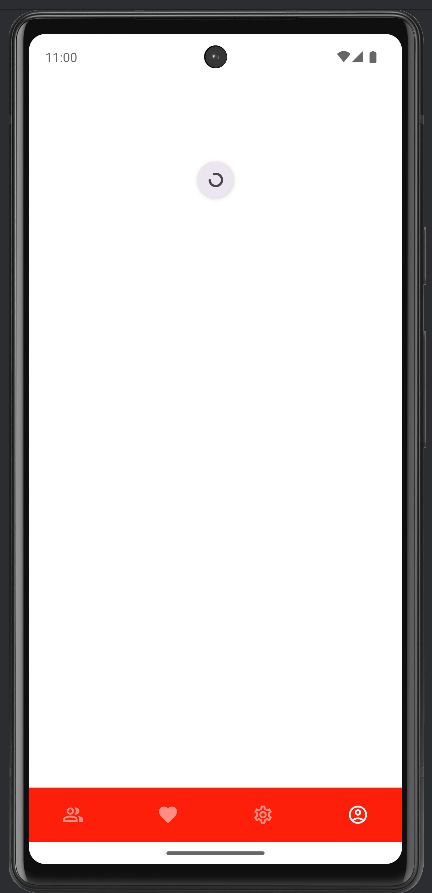
На рисунке 10 ниже показывается процесс загрузки данных выбранного героя.

Рис 10. Загрузка данных выбранного героя

На рисунках 11 и 12 показан профиль выбранного героя с информацией о нем.

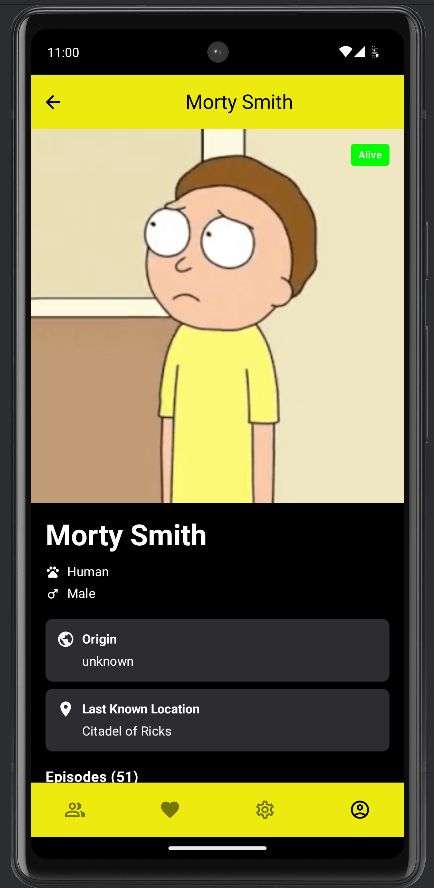
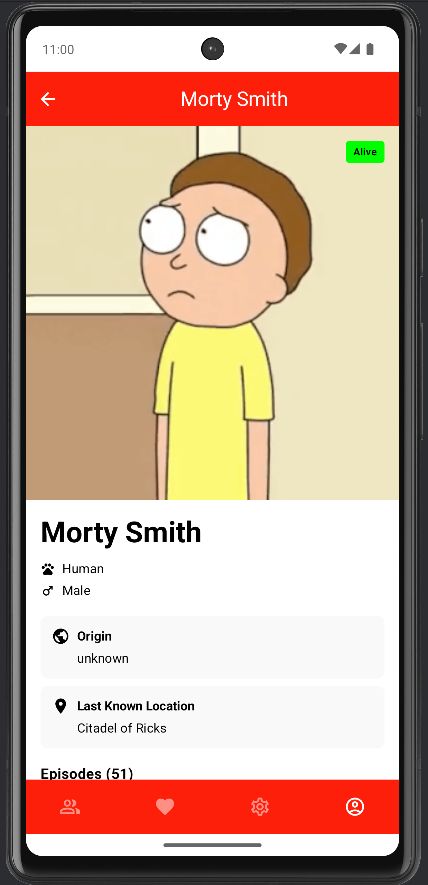


Рис 11. Выбранный герой Рис 12. Выбранный герой, темная тема

**Экран героя**

На этом экране показывается детальная информация о герое, как и на экране профиля. Этот экран открывается при нажатии на героев на экране героев.

Ниже на рисунках 13 и 14 показан данный экран в светлой и темной теме.

Здесь, как и на экране выбранного пользователем героя, показываются данные, которые берутся с API.



Рис 13. Экран героя Рис 14. Экран героя, темная тема

**Заключение**

В данной работе изучались теоретические основы языка программирования Kotlin и их практическая реализация.

В ходе выполнения работы были решены следующие задачи:

- Изучены материалы по следующим темам:

- Язык программирования Kotlin

- Основы программирования в среде Android Studio

- Основы сетевого взаимодействия, Retrofit

- Работа с локальной базой данных, Room

- Внедрение зависимостей c помощью Dagger-Hilt

- Архитектура мобильных приложения

- Написание пользовательского интерфейса с помощью Jetpack Compose

- Пагинация с помощью Paging 3

- Загрузка и кэширование картинок с помощью Coil

- Написано мобильное Android-приложение «Герои Рик и Морти»

Таким образом, все поставленные задачи были решены. Цель обучения можно считать достигнутой.

**Список литературы**

1. Android Developers. Официальная документация –URL: [https://developer.android.com](https://developer.android.com/)
2. Google Developers. Руководства по Android – URL: [https://developer.google.com](https://developer.google.com/)
3. Habr. Разработка под Android – URL: <https://habr.com/ru/hub/android_dev/>
4. Medium. Статьи по Android-разработке – URL: <https://medium.com/tag/android>
5. Android Jetpack. Библиотеки для разработки – URL: <https://developer.android.com/jetpack>

**Приложения**

Приложение 1: Экран героя

@Composable

internal fun CharacterScreen(

character: Screen.Character,

sendRootIntent: (NavRootIntent) -> Unit

) { val viewModel = hiltViewModel<CharacterViewModel, CharacterViewModel.Factory> { factory -> factory.create(character.id) }

val state by viewModel.state.collectAsState()

val mainColor by rememberUpdatedState(

state.settings.getMainColorForThisTheme(isSystemInDarkTheme()) ?: AppTheme.colors.mainColor)

BackHandler { sendRootIntent(NavRootIntent.Back) }

Scaffold(

topBar = { if (!state.isLoading && !state.isError && state.characterData != null) {

state.characterData?.let { CharacterTopAppBar(characterName = it.name,

onBackClick = { sendRootIntent(NavRootIntent.Back) },

containerColor = mainColor) } } },

containerColor = AppTheme.colors.background

) { paddingValues -> PullToRefreshBox(

modifier = Modifier.fillMaxSize().padding(top = paddingValues.calculateTopPadding()),

contentAlignment = Alignment.Center, isRefreshing = state.isLoading,

onRefresh = { viewModel.handleIntent(CharacterIntent.FetchData) }

) { state.let { state ->

if (state.isError) { ErrorItem(message = "error",modifier = Modifier.fillMaxSize()) { viewModel.handleIntent(CharacterIntent.FetchData) } }

if (!state.isLoading && !state.isError && state.characterData != null) {

CharacterContent(character = state.characterData) } } } } }

Приложение 2. Экран избранного.

@OptIn(ExperimentalMaterial3Api::class)

@Composable

internal fun FavouritesScreen(sendRootIntent: (NavRootIntent) -> Unit) { val viewModel: FavouritesViewModel =

hiltViewModel<FavouritesViewModel, FavouritesViewModel.Factory> { it.create(sendRootIntent) }

val characters = viewModel.favouriteCharactersFlow.collectAsLazyPagingItems()

val state by viewModel.state.collectAsState()

LaunchedEffect(Unit) { viewModel.effects.collect { effect -> when (effect) {

CharactersEffect.Refresh -> characters.refresh() } } }

LaunchedEffect(characters.itemCount) { viewModel.handleIntent(

FavouritesIntent.IsDataEmpty(characters.itemCount == 0)) }

Scaffold(containerColor = AppTheme.colors.background) { innerPadding ->

PullToRefreshBox(modifier = Modifier.fillMaxSize(),

contentAlignment = Alignment.Center,

isRefreshing = characters.loadState.refresh is LoadState.Loading,

onRefresh = { viewModel.handleIntent(FavouritesIntent.Refresh) }

) { if (state.dataIsEmpty) {

ThisThemeText(stringResource(R.string.data\_is\_empty)) }

when (characters.loadState.refresh) { is LoadState.Error -> {

val error = characters.loadState.refresh as LoadState.Error

ErrorItem(message = error.error.localizedMessage ?: "Unknown error",

modifier = Modifier.align(Alignment.Center),

onClickRetry = { characters.retry() }) }

else -> LazyColumn(modifier = Modifier.fillMaxSize()) {

items(count = characters.itemCount) { index -> val entity = characters[index]

if (entity != null) { CharacterItem(character = entity,

placeholder = painterResource(R.drawable.placeholder), isFavourite = true, changeIsFavourite = { viewModel.handleIntent(

FavouritesIntent.ChangeIsFavourite(entity.id)) },

onClick = { viewModel.handleIntent(

FavouritesIntent.ClickToCharacter(entity.id)) },

isChosen = state.settings.yourCharacterId == entity.id,

changeIsChosen = { viewModel.handleIntent(

FavouritesIntent.SetIsMyCharacter(entity.id)) }) }

else LoadingItem() }

if (characters.loadState.append is LoadState.Error) { item {

val error = characters.loadState.append as LoadState.Error

ErrorItem(message = error.error.localizedMessage ?: "Load more error",modifier = Modifier.fillMaxWidth(),onClickRetry = { characters.retry() }) } }if (characters.loadState.append is LoadState.Loading) { item {

Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement = Arrangement.Center) {CircularProgressIndicator(color = Color.Red,

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

Приложение 3. Экран героев.

@Composable

internal fun CharactersScreen(sendRootIntent: (NavRootIntent) -> Unit) { val viewModel: CharactersViewModel =

hiltViewModel<CharactersViewModel, CharactersViewModel.Factory> { it.create(sendRootIntent) }

val characters = viewModel.charactersFlow.collectAsLazyPagingItems()

val state by viewModel.state.collectAsState()

val mainColor by rememberUpdatedState(state.settings.getMainColorForThisTheme(isSystemInDarkTheme()) ?: AppTheme.colors.mainColor) LaunchedEffect(characters.itemCount) { viewModel.handleIntent(CharactersIntent.IsDataEmpty(characters.itemCount == 0)) } LaunchedEffect(Unit) { viewModel.effects.collect { effect -> when (effect) { CharactersEffect.Refresh -> characters.refresh() } } }

Scaffold(containerColor = AppTheme.colors.background, floatingActionButtonPosition = FabPosition.Center,

floatingActionButton = { FloatingActionButton( onClick = { viewModel.handleIntent(CharactersIntent.ChangeFilterVisibility(true)) },

containerColor = mainColor, contentColor = mainColor.suitableColor()

) { Icon(painter = painterResource(id = R.drawable.filters), contentDescription = "Filters") } }

) { innerPadding -> if (state.isFilterBottomSheetVisible) { FiltersBottomSheet(

initialFilters = state.filters,

onFiltersApplied = { filters -> viewModel.handleIntent(CharactersIntent.ApplyFilters(filters)) },

onDismissRequest = { viewModel.handleIntent(CharactersIntent.ChangeFilterVisibility(false)) }) }

PullToRefreshBox(modifier = Modifier.fillMaxSize(),

contentAlignment = if (state.dataIsEmpty) Alignment.Center else Alignment.TopCenter,

isRefreshing = characters.loadState.refresh is LoadState.Loading,

onRefresh = { viewModel.handleIntent(CharactersIntent.Refresh) }

) { if (state.dataIsEmpty) { ThisThemeText(stringResource(R.string.data\_is\_empty)) }

when (characters.loadState.refresh) { is LoadState.Error -> {

val error = characters.loadState.refresh as LoadState.Error

ErrorItem(message = error.error.localizedMessage ?: "Unknown error",

modifier = Modifier.align(Alignment.Center),

onClickRetry = { characters.retry() }) }

else -> LazyColumn(modifier = Modifier.fillMaxSize()) {

items(count = characters.itemCount) { index -> val entity = characters[index] if (entity != null) { CharacterItem( character = entity, placeholder = painterResource(R.drawable.placeholder),

isFavourite = entity.id in state.favourites, changeIsFavourite = { viewModel.handleIntent(CharactersIntent.ChangeIsFavourite(entity.id)) },

onClick = { viewModel.handleIntent(CharactersIntent.ClickToCharacter(entity.id)) },

isChosen = state.settings.yourCharacterId == entity.id,

changeIsChosen = { viewModel.handleIntent(CharactersIntent.SetIsMyCharacter(entity.id)) }) }

else LoadingItem() }

if (characters.loadState.append is LoadState.Error) { item {

val error = characters.loadState.append as LoadState.Error

ErrorItem(message = error.error.localizedMessage ?: "Load more error",

modifier = Modifier.fillMaxWidth(),

onClickRetry = { characters.retry() }) } }

if (characters.loadState.append is LoadState.Loading) { item {

Row(modifier = Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.Center) {

CircularProgressIndicator(color = Color.Red,

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

Приложение 4. Экран профиля выбранного героя

@OptIn(ExperimentalMaterial3Api::class)

@Composable

internal fun ProfileScreen(sendRootIntent: (NavRootIntent) -> Unit) {

val viewModel = hiltViewModel<ProfileScreenViewModel>()

val state by viewModel.state.collectAsState()

val mainColor by rememberUpdatedState(state.settings.getMainColorForThisTheme(isSystemInDarkTheme()) ?: AppTheme.colors.mainColor)

BackHandler { sendRootIntent(NavRootIntent.Back) }

Scaffold(

topBar = { if (!state.isLoading && !state.isError && state.characterData != null) {

state.characterData?.let { CharacterTopAppBar(

characterName = it.name,

onBackClick = { sendRootIntent(NavRootIntent.Back) },

containerColor = mainColor) } } },

containerColor = AppTheme.colors.background

) { paddingValues -> PullToRefreshBox(

modifier = Modifier.fillMaxSize().padding(top = paddingValues.calculateTopPadding()),

contentAlignment = Alignment.Center,

isRefreshing = state.isLoading,

onRefresh = { viewModel.handleIntent(CharacterIntent.FetchData) }

) { state.let { state ->

if (state.isError) { ErrorItem(

message = "error",

modifier = Modifier.fillMaxSize()

) { viewModel.handleIntent(CharacterIntent.FetchData) } }

if (!state.isLoading && !state.isError && state.characterData != null) {

CharacterContent(character = state.characterData) } } } } }

Приложение 5. Экран настроек

@Composable

internal fun SettingsScreen(sendRootIntent: (NavRootIntent) -> Unit) {

val viewModel: SettingsScreenViewModel = hiltViewModel()

val state by viewModel.state.collectAsState()

val context = LocalContext.current

val dark = isSystemInDarkTheme()

val mainDefaultColor = AppTheme.colors.mainColor

val mainColor by remember { derivedStateOf { state.settings.getMainColorForThisTheme(isDark = dark) ?: mainDefaultColor } }

BackHandler { sendRootIntent(NavRootIntent.Back) }

LaunchedEffect(null) { viewModel.effects.collect { effect -> when (effect) {

is SettingsEffect.OpenLink -> context.openLink(effect.link)

is SettingsEffect.SendEmail -> context.sendEmail(effect.email) } } }

Scaffold(containerColor = AppTheme.colors.background, modifier = Modifier.fillMaxSize()) { innerPadding ->

Column(modifier = Modifier.fillMaxSize().padding(top = innerPadding.calculateTopPadding())) {

if (state.colorPickerDialogIsEditable) { ColorPickerDialog(

firstColor = mainColor,

onDismissRequest = { viewModel.handleIntent(SettingsIntent.CloseColorEditDialog) },

onChangeColorCallback = { selectedColor -> viewModel.handleIntent(

SettingsIntent.SaveSettings { s -> if (dark) s.copy(darkThemeColor = selectedColor) else s.copy(lightThemeColor = selectedColor) }) }) }

Spacer(modifier = Modifier.height(30.dp))

Column(modifier = Modifier.weight(1f).padding(horizontal = 20.dp).verticalScroll(rememberScrollState())) {

Box(modifier = Modifier.fillMaxWidth()) { ThisThemeText(

modifier = Modifier.padding(10.dp).align(Alignment.Center),

fontSize = FontSize.big22,

text = stringResource(R.string.interface\_str)) }

CardOfSettings(text = stringResource(R.string.interface\_color), icon = { Icon(painter = painterResource(id = R.drawable.palette),

contentDescription = stringResource(R.string.interface\_color),

tint = it.suitableColor()) }, onClick = { viewModel.handleIntent(SettingsIntent.OpenColorEditDialog) })

ThisThemeText(modifier = Modifier.padding(10.dp).align(Alignment.CenterHorizontally), fontSize = FontSize.big22, text = stringResource(R.string.contacts)) CardOfSettings(text = stringResource(R.string.code), icon = { Icon(

painter = painterResource(id = R.drawable.terminal),contentDescription = stringResource(R.string.code),

tint = it.suitableColor()) },onClick = { viewModel.handleIntent(SettingsIntent.OpenLink(Link.CODE)) })

CardOfSettings(text = stringResource(R.string.report\_a\_bug), icon = { Icon(

painter = painterResource(id = R.drawable.bug\_report),

contentDescription = stringResource(R.string.report\_a\_bug),

tint = it.suitableColor()) },

onClick = { viewModel.handleIntent(SettingsIntent.SendEmail(Link.MAIL)) })

ThisThemeText(modifier = Modifier.padding(10.dp).padding(bottom = 20.dp).align(Alignment.End),

fontSize = FontSize.small17,

text = "${stringResource(R.string.version)} ${context.getVersionName()}") } } } }