Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

09.03.01 Информатика и вычислительная техника
Профиль: Программное обеспечение средств вычислительной техники и автоматизированных систем
(очная форма обучения)

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

в/на ООО «Мобильные Технологии»
(наименование профильной организации/структурного подразделения СибГУТИ)

Разработка Android приложения для просмотра GitHub репозиториев

Выполнил:
студент института ИВТ
гр. ИП-012
«27» мая 2023г. _____/ Николаев А.Д. /
(подпись)

Проверил:
Руководитель от профильной организации
«27» мая 2023г. _____/ Бобренев А.Ю.
(подпись)

Проверил:
Руководитель от СибГУТИ
«27» мая 2023г. _____/_____/
(подпись)

Новосибирск 2023

План-график проведения производственной **практики**

Вид практики

Николаев Алексей Дмитриевич

Фамилия Имя Отчество студента

института Информатика и вычислительная техника, 3 курса, гр. ИП-012

Направление: 09.03.01 – Информатика и вычислительная техника

Профиль: Программное обеспечение средств вычислительной техники и автоматизированных систем

Место прохождения практики ООО "Мобайл Девелопмент"

Объем практики: **360/10** часов/ЗЕ

Вид практики ***производственная***

Тип практики ***Технологическая (проектно-технологическая) практика***

Срок практики с "30" января 2023 г.
по "27" мая 2023 г.

Содержание практики*:

Наименование видов деятельности	Дата (начало – окончание)
1. Общее ознакомление со структурным подразделением предприятия, вводный инструктаж по технике безопасности	30.01.2023–01.02.2023
2. Выдача задания на практику, деление студентов на группы (если необходимо), определение конкретной индивидуальной темы, формирование плана работ	02.02.2023–04.02.2023
3. Работа с библиотечными фондами структурного подразделения или предприятия, сбор и анализ материалов по теме практики	06.02.2023–11.02.2023
4. Выполнение работ в соответствии с составленным планом: – Изучения основ языка разработки мобильных приложений Kotlin. – Ознакомление со средой разработки Android Studio и библиотек для работы с ним. – Изучение языка системы автоматической сборки Gradle	13.02.2023 – 20.05.2023
5. Анализ полученных результатов и произведенной работы Составление отчета по практике, защита отчета	22.05.2023–27.05.2023

*В соответствии с программой практики

Руководитель от профильной организации
«27» мая 2023г. _____ / Бородинов Д.Ю. (подпись)

Руководитель от СибГУТИ
«____» _____ 2023г. _____ / _____ / (подпись)

Задание на практику

Разработать Android приложение для просмотра GitHub репозиториев. Данное приложение должно удовлетворят следующим техническим требованиям:

1. Реализовать приложение на языке программирования Kotlin
2. Использовать XML Layouts для User Interface
3. Использовать систему сборки Kotlin Gradle DSL
4. Использовать Retrofit для работы с REST API
5. Использовать RecyclerView для отображения списка
6. Использовать ConstraintLayout для экрана детальной информации
7. Использовать Android Navigation Component для переходов между экранами
8. Использовать View Binding для связывания верстки с кодом
9. Создать экраны с помощью Fragment с подходом Single Activity
10. Использовать Coroutines для асинхронности и многопоточности
11. Использовать Kotlinx.Serialization для парсинга json информации с сайта GitHub
12. Использовать ViewModel для реализации логики экранов
13. Использовать LiveData / StateFlow для обновления данных на UI
14. Использовать Dagger Hilt для внедрения зависимостей
15. Сохранять токен авторизации в хранилище устройства - SharedPreferences
16. Корректно обрабатывать ситуации "загрузка данных", "ошибка загрузки", "пустой список"
17. Корректно обрабатывать смену конфигурации
18. При перезапуске приложения авторизация должна сохраняться
19. Использовать локализацию для всех строк, показываемых пользователю
20. Обеспечить поддержку Android API 21

Введение

Программист, разрабатывающий приложения для мобильных устройств, обязан хорошо знать целевую платформу своего приложения, а также каким функционалом обладает платформа в зависимости от версий.

Android – операционная система для смартфонов, планшетов и других переносных электронных устройств, разработанная компанией Android, Inc., основанная на ядре операционной системы Linux и собственной реализации виртуальной машины Java компании Google. (см. рис. 1)

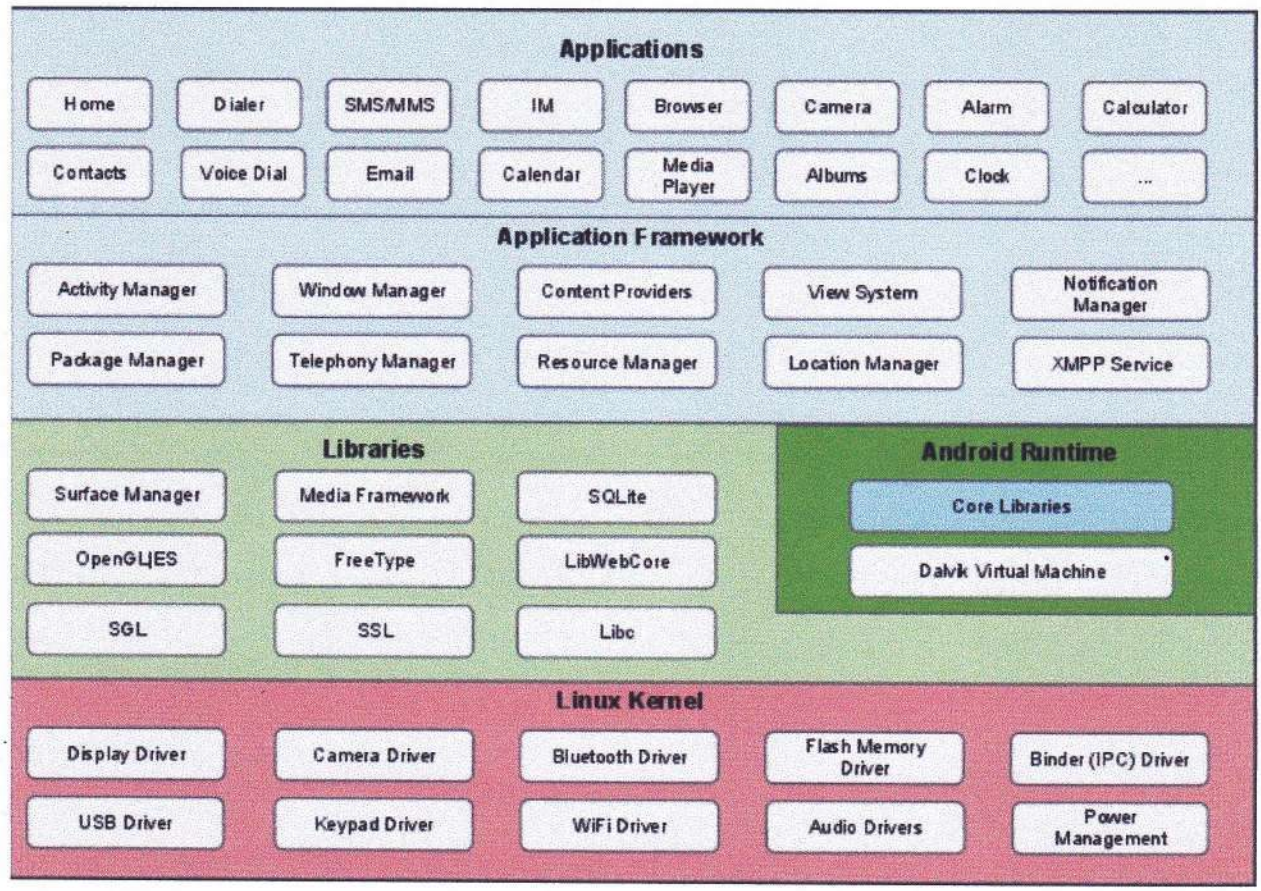

Рис. 1. Строение архитектуры Android приложений

Для разработки Android приложений на данный момент активно используется среда разработки Android Studio, которая поддерживает следующие языки программирования:

- Kotlin;
- Java;
- C++.

Кроме языка программирования, программист должен уметь работать со сторонними программными модулями, без которых приложение не будет обладать достаточным функционалом для реализации поставленных задач, и работать с системой сборки Gradle, в которой эти модули импортируются.

Базовые компоненты Android приложения

Для реализации Android приложения для просматривания GitHub репозиториев необходимо использовать множество базовых компонентов, присутствующих в Android Studio. В это множество входят следующие компоненты:

- XML Layouts;
- ViewModel;
- Activity;
- Fragments;
- RecyclerView;
- ConstraintLayout;
- Coroutines.

XML Layouts – файлы формата .xml, которые определяют структуру пользовательского интерфейса в приложении. В них хранятся все текстовые поля, кнопки, изображения и многое другое, называемое объектами представления (View), что позволяет расположить эти объекты в необходимом для пользователя виде и порядке.

Данные XML Layouts можно привязывать к программному коду, что позволяет изменять их содержимое во время работы программы. После привязывания создаётся класс привязки, которые содержит прямые ссылки на все View, имеющие ID в соответствующем Layout.

ViewModel – это класс, отвечающий за подготовку и управление данными для Activity или Fragment. Он также обрабатывает связь Activity / Fragment с остальной частью приложения (например, вызов классов бизнес-логики).

Activity – это фундаментальный компонент каждого Android-приложения. Через этот компонент происходит взаимодействие между пользователем и приложением. При помощи него пользователь телефона может «путешествовать» между окнами приложения или между разными приложениями.

Подход, при котором Activity, с которым работает пользователь, остаётся тем же самым, но содержимое меняется в зависимости от действий пользователя. Для реализации такого способа проектирования Android приложений активно используются объекты класса Fragments.

Fragments – компонент Android приложения, который по своему функционалу очень похож на Activity. Разница заключается лишь в том, что Fragment определяет собственный макет и управляет им, имеет собственный жизненный цикл и может обрабатывать свои собственные входные события.

Fragments не могут жить сами по себе. Они должны размещаться в Activity или другом Fragment. Иерархия Fragment's Views становится частью или присоединяется к иерархии Views хоста.

RecyclerView – компонент Android приложения, которые представляет собой View для эффективного отображения больших наборов данных в виде списка. Когда элемент прокручивается за пределы экрана, RecyclerView не уничтожает его View. Вместо этого RecyclerView повторно использует View для новых элементов, прокручиваемых на экране. RecyclerView повышает производительность и скорость отклика вашего приложения, а также снижает энергопотребление.

ConstraintLayout – компонент Android приложений, который позволяет определять положение и размер элементов внутри него, а также относительно других элементов: элементы могут располагаться друг под другом, растягиваться в зависимости от ширины Layout и тому подобное. Данный компонент работает в Android устройствах с уровнем API большим или равным 9.

Coroutines – это блоки кода, которые работают асинхронно, то есть по очереди. В нужный момент исполнение такого блока приостанавливается с сохранением всех его свойств, чтобы запустился другой код. Когда управление возвращается к первому блоку, он продолжает работу.

В Android Coroutines помогают управлять длительными задачами, которые в противном случае могут заблокировать основной поток и привести к тому, что Android приложение перестанет отвечать на запросы.

Плагины Android приложения

Ознакомившись в общих чертах с базовыми компонентами нашего Android приложения, рассмотрим подключаемые модули (плагины), которые необходимы для реализации функционала нашего приложения. Основными плагины в данном приложении будут являться:

- Retrofit
- Dagger Hilt

Большинство представленных на рынке приложений для Android подключаются к Интернету для выполнения некоторых сетевых операций. Например, получение электронных писем, сообщений или аналогичной информации с внутреннего сервера. Gmail, YouTube и Google Фото — это примеры приложений, которые подключаются к Интернету для отображения пользовательских данных.

Retrofit – библиотека, которая позволяет работать с REST (передачей состояния представления) - программным обеспечением, которое создаёт код для обмена между двумя приложениями данными с сервера по протоколу прикладного уровня HTTP. За счёт этой библиотеки возможно считывать информацию с сайтов (например с сайта github.com) в формате .json файлов для дальнейшей её обработки.

Hilt – это библиотека внедрения зависимостей для Android приложения, которая упрощает выполнение ручного внедрения зависимостей в проекте. Выполнение внедрения зависимостей вручную требует от вас создания каждого класса и его зависимостей вручную, а также использования контейнеров для повторного использования и управления зависимостями.

Hilt предоставляет стандартный способ использования внедрения зависимостей в вашем приложении, предоставляя контейнеры для каждого класса Android в вашем проекте и автоматически управляя их жизненными циклами. Hilt построен на основе популярной библиотеки DI Dagger, чтобы извлечь выгоду из правильности времени компиляции, производительности во время выполнения, масштабируемости и поддержки Android Studio, которую предоставляет Dagger.

Техническая реализация приложения

Для создания Android приложения для просмотра GitHub Репозиториев необходимо создать классы, соответствующие построенной заранее диаграмме классов (ULM) (см. рис. 2)

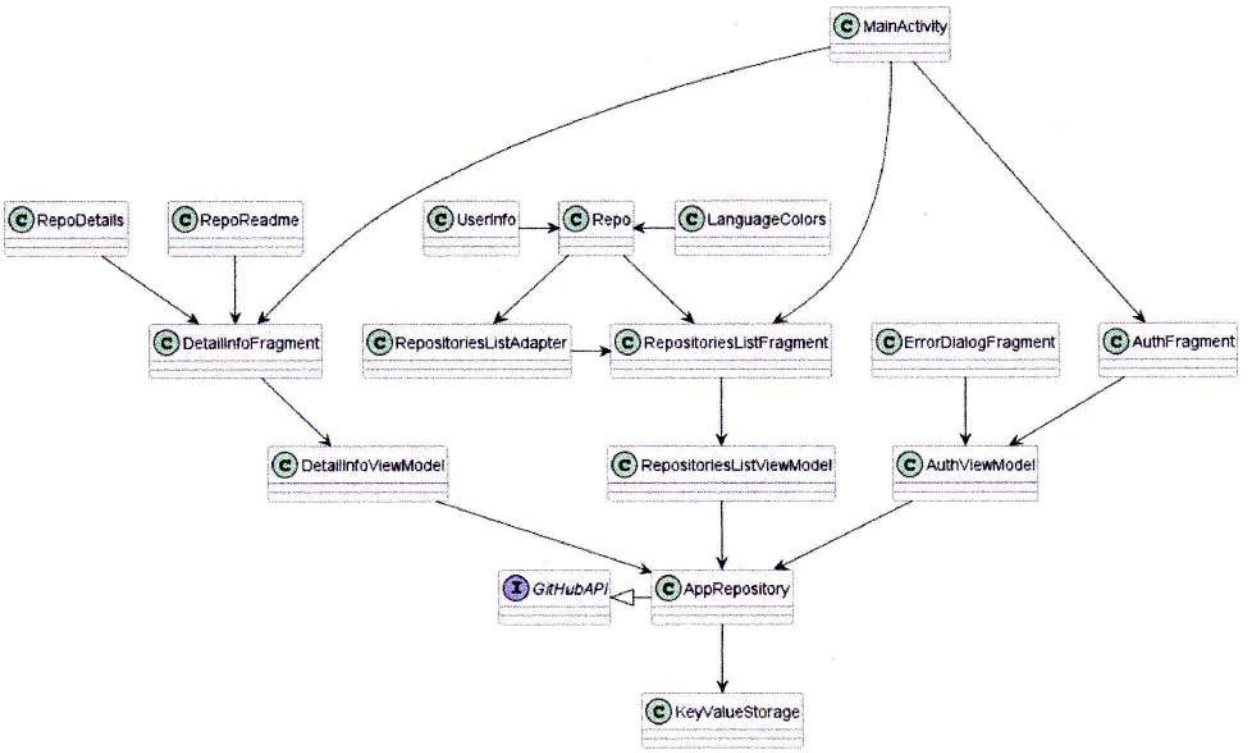

Рис. 2. ULM диаграмма классов Android приложения

При запуске Android приложения запускается перегруженная функция onCreate() класса MainActivity, которая создаёт View, на котором будут отображаться все фрагменты приложения, в соответствии с подходом SingleActivity.

После создания MainActivity создаётся фрагменты классов AuthFragment, RepositoriesListFragment и DetailInfoFragment, т.к. в программном коде эти классы указаны с аннотацией @AndroidEntryPoint библиотеки Hilt, которая связывает их инициализацию с открытием приложения. Переключение между модулями, в которых хранятся фрагментов обеспечивается за счёт объекта класса NavController, которому при инициализации фрагментов передаются их модули в соответствии с их названиями.

AuthFragment отвечает за то, чтобы пользователь мог ввести токен профиля GitHub'а для получения списка репозиториев в RepositoriesListFragment и переключения на этот фрагмент. (см. рис. 3)

Рис. 3. Пример ввода токена в AuthFragment для авторизации пользователя

В случае ввода неправильного токена, пользователю подсветится красным цветов надпись о том, что введённый токен не корректен. В случае если у пользователя будет отсутствовать подключение к интернету, появится всплывающее окно с сообщением (ErrorDialogFragment) о том, что произошла ошибка и необходимо перепроверить подключение к Интернету. (см. рис. 4)

Рис. 4. Попытка авторизации пользователя без подключения к интернету

После того, как авторизация прошла успешна, в объект класса KeyValueStorage переменная isAuthorized становится равна true, а переменные присваивают значения токена авторизации (authToken) и имени пользователя (userName). После внесения изменения эти данные сохраняются в переменной типа SharedPreferences в результате чего при выходе из программы информация о пользователе сохраняется и ему не надо авторизироваться повторно.

RepositoriesListFragment отвечает за то, чтобы считать полученную объектом класса AppRepository с помощью библиотеки Retrofit информацию о репозиториях и сформировать из неё объект класса RecyclerView. Данные о каждом репозитории сохраняются в список типа Repo, в котором хранится краткая информация о репозитории, включая объект класса UserInfo, а также объект класса LanguageColors для определения цвета текста главного языка программирования репозитория.

Т.к. элементами RecyclerView являются Views, то необходимо создать отдельный .xml файл (repositories_list_item.xml), для структуризации полученной информации. Также для работы RecyclerView нужен адаптер RepositoriesListAdapter, который предназначен для создания элементов списка по заданному .xml файлу. (см. рис. 5)

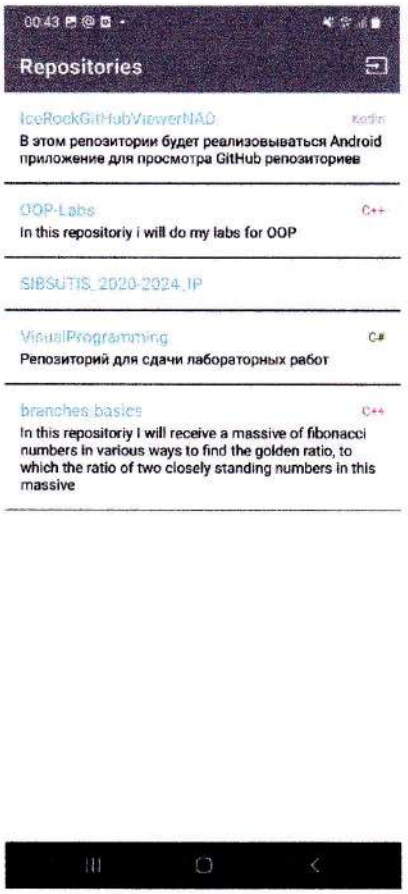

Рис. 5. Список репозиториев пользователя

При выборе интересующего пользователя репозитория, фрагмент запрашивает объект класс AppRepository получить с помощью библиотеки Retrofit более детальную информацию о репозитории, которая считывается фрагментом класса DetailsInfoFragment.

DetailsInfoFragment отвечает за то, чтобы отобразить пользователю более подробную информацию о репозитории (количестве звёзд, лицензии, количестве веток и т.п.), а также содержание ReadMe.md файла, хранящегося в репозитории. Информация о текущем репозитории сохраняется в объекте класса RepoDetails, в то время как содержание файла ReadMe.md – в объекте класса RepoReadMe. После этого информация передаётся в xml файл фрагмента и информация становится видна пользователю. (см. рис. 6)

Рис. 6. Детальная информация об открытом репозитории

Как уже говорилось ранее, класс AppRepository отвечает за то, чтобы делать запросы на HTTP сервер сайта github.com с помощью библиотеки Retrofit, после чего передавать полученную из .json файлов информацию фрагментам пользовательского интерфейса.

Класс KeyValueStorage отвечает за то, чтобы сохранять текущее состояние приложение в случае разрушения ViewModel объекта MainActivity в результате поворота экрана или закрытия, в результате чего пользователю не нужно повторно авторизироваться.

Заключение

В процессе прохождения производственной практики были изучены основы программирования мобильных приложений в крупных предприятиях, а также получен опыт работы в условиях производственной деятельности. Поставленные задачи были сделаны и сданы в срок.

Приложение было создано согласно всем техническим требованиям и были использованы все теоретические знания, накопленные во время производственной практики.

Список используемой литературы

1. Соколова В.В. Вычислительная техника и информационные технологии. Разработка мобильных приложений: Учебное пособие / Национальный Исследовательский Томский Политехнический университет. – Москва: Юрайт, 2020. – 175 с.
2. Android Developers: [Электронный ресурс]. URL: https://developer.android.com (Дата обращения: 23.05.2023).
3. IceRock KMM University: [Электронный ресурс]. URL: https://kmm.icerock.dev/university/intro (Дата обращения: 23.05.2023).
4. Kotlin v1.8.21: [Электронный ресурс]. URL: https://kotlinlang.org/docs/home.html (Дата обращения: 15.05.2023).
5. RecyclerView: [Электронный ресурс] // Освой Android играючи. URL: https://developer.alexanderklimov.ru/android/views/recyclerview-kot.php (Дата обращения: 16.05.2023).

Приложения

AppRepository

```
@Singleton
class AppRepository @Inject constructor(
    private val githubApi: GithubApi,
    private val keyValueStorage: KeyValueStorage,
) {

    suspend fun getRepositories(): Response<List<Repo>> = githubApi.getRepos-
itoriesList(
        authHeader = "Bearer ${keyValueStorage.authToken}",
        userName = keyValueStorage.userName!!
    )

    suspend fun getRepository(repoId: String): {…}

    suspend fun getRepositoryReadme(…) {…}

    suspend fun signIn(token: String): Response<UserInfo> {
        val res = githubApi.getUserInfo(
            authHeader = "Bearer $token"
        )
        if (res.isSuccessful) {
            keyValueStorage.authToken = token
            keyValueStorage.userName = res.body()?.login
            keyValueStorage.isAuthorized = true
        }
        return res
    }
}
```

AuthFragment

```
@AndroidEntryPoint
class AuthFragment : Fragment() {
    private fun handleAction(action: Action) {
        when (action) {
            is Action.RouteToMain -> {
 navController.navigate(R.id.action_authFragment_to_repositoriesListFragment)
            }

            is Action.ShowError -> {
                val errorDialogFragment = ErrorDialogFragment.create(
                    error = action.error,
                    code = action.httpCode
                )
                activity?.supportFragmentManager?.let {
                    errorDialogFragment.show(it, getString(R.string.error))
                }
                binding.signInButton.isEnabled = true
            }
        }
    }
}
```

AuthViewModel

```
@HiltViewModel
class AuthViewModel @Inject constructor(
    private val repository: AppRepository,
) : ViewModel() {

    private val tokenValidationPattern = "^[a-z_0-9]+$".toRegex(RegexOption.IGNORE_CASE)

    var token = MutableLiveData<String>()
        private set

    var state = MutableLiveData<State>()
        private set


    private val _actions: Channel<Action> = Channel(Channel.BUFFERED)
    val actions: Flow<Action> = _actions.receiveAsFlow()
. . .

    private val exceptionHandler = CoroutineExceptionHandler { _, throwable ->
        viewModelScope.launch {
            _actions.send(
                Action.ShowError(
                    error = ApiError.NetworkError(ErrorType.NetworkError),
                    message = throwable.localizedMessage
                )
            )
        }
    }

    fun onSignInButtonPressed() {
        signInJob = CoroutineScope(Dispatchers.IO + exceptionHandler).launch {
            state.postValue(State.Loading)

            val response = repository.signIn(token.value.toString())
            withContext(Dispatchers.Main) {

                if (response.code() == 401) {
                    state.postValue(State.InvalidInput)
                    return@withContext
                }
                _actions.send(
                    Action.ShowError(
                        error = ApiError.NetworkError(ErrorType.HttpError(response.code())),
                        httpCode = response.code()
                    ))
                state.postValue(State.Idle)
            }
        }
    }
}
```

DetailInfoFragment

```
@AndroidEntryPoint
class DetailInfoFragment : Fragment() {
    . . .
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.menu_action_logout -> {
                viewModel.logOut()
                navController.navigate(R.id.action_detailFrag-
ment_to_authFragment)
                true
            }
            android.R.id.home -> {
         navController.navigate(R.id.action_detailFragment_to_repositories-
ListFragment)
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }

    private fun FragmentDetailInfoBinding.bindReadme(
        state: ReadmeState,
    ) {
        readmeText.visibility =
            if (state !is ReadmeState.Loading) View.VISIBLE else View.INVISI-
BLE

        noReadmeLabel.visibility =
            if (state is ReadmeState.Empty) View.VISIBLE else View.INVISIBLE

        if (state is ReadmeState.Loaded) {
            readmeText.removeAllViews()
            val paragraphs = markDownRenderer.parseMarkdown(state.markdownTo-
String())
            for (paragraph in paragraphs) readmeText.addView(paragraph)
        }
        readmeLoadingPb.visibility = if (state is ReadmeState.Loading)
View.VISIBLE else
            View.INVISIBLE
    }
    private fun FragmentDetailInfoBinding.bindDetailInfo(… ) {… }
    private fun openUrl(url: String) {
        val intent = Intent(Intent.ACTION_VIEW, Uri.parse(url))
        startActivity(intent)
    }
}
```

DetailInfoViewModel

```
@HiltViewModel
class DetailInfoViewModel @Inject constructor(
    private val repository: AppRepository,
) : ViewModel() {

    private var fetchRepositoryDetailsJob: Job? = null
    private var fetchRepositoryReadmeJob: Job? = null

    var state = MutableLiveData<State>()
        private set
```

```
    var readmeState = MutableLiveData<ReadmeState>()
        private set

    fun fetchRepository(repositoryName: String) {
        fetchRepositoryDetailsJob =
            CoroutineScope(Dispatchers.IO + fetchRepositoryExceptionHan-
dler).launch {
                state.postValue(State.Loading)
                val response = repository.getRepository(repositoryName)
                withContext(Dispatchers.Main) {
                    if (response.isSuccessful) {
                        state.postValue(State.Loaded(response.body()!!))
                        if (response.body() is RepoDetails) re-
sponse.body()?.let { repo ->
                            fetchReadme(
                                repositoryName = repo.name,
                                defaultBranch = repo.defaultBranch,
                                owner = repo.owner.login
                            )
                        }
                        return@withContext
                    }
                    state.postValue(
                        State.Error(
                            ApiError.Error(response.code().toString())
                        )
                    )
                }
            }
    }

    fun fetchReadme(...) {...}
    ...
}
```

RepositoriesListFragment

```
@AndroidEntryPoint
class RepositoriesListFragment : Fragment() {
        . . .
    private fun FragmentRepositoriesListBinding.bindRepositoriesList(
        state: RepositoriesListViewModel.State,
    ) {
        repositoriesListLoadingPb.visibility =
            if (state is Loading) View.VISIBLE else View.INVISIBLE
        repositoriesListRecyclerView.visibility = if (state is Loaded)
View.VISIBLE else View.INVISIBLE
        if (state is Loaded) {
            val repositoriesListAdapter = RepositoriesListAdapter { position
-> onListItemClick(state.repos[position])
            }
            repositoriesListAdapter.items = state.repos

            repositoriesListRecyclerView.adapter = repositoriesListAdapter
        } else {
            repositoriesListRecyclerView.adapter = null
        }
    }
}
```

RepositoriesListViewModel

```
@HiltViewModel
class RepositoriesListViewModel @Inject constructor(
    private val repository: AppRepository,
) : ViewModel() {
. . .
    fun fetchRepositories() {
        fetchRepositoriesJob = CoroutineScope(Dispatchers.IO + exceptionHan-
dler).launch {
            state.postValue(State.Loading)
            val response = repository.getRepositories()
            withContext(Dispatchers.Main) {
                if (response.isSuccessful) {
                    if (response.body().isNullOrEmpty()) {
                        state.postValue(State.Empty)
                        return@withContext
                    }
                    state.postValue(
                        State.Loaded(response.body()?.take(10) ?: emp-
tyList())
                    )
                    return@withContext
                }
                state.postValue(
                State.Error(ApiError.NetworkError(ErrorType.HttpError(re-
sponse.code())))
                )
            }
        }
    }
}
```

GithubApi

```
interface GithubApi {
@GET("user")
suspend fun getUserInfo(
    @Header("Authorization") authHeader: String,
): Response<UserInfo>
@GET("users/{userName}/repos")
. . .
@GET("repos/{owner}/{repoName}")
. . .
@GET("repos/{owner}/{repoName}/readme")
. . .
}
```

Отзыв о работе студента

Николаев Алексей Дмитриевич

(ФИО студента)

Способен выполнять задачи любого уровня сложности. К выполнению поставленных задач подходит с большой ответственностью и трудолюбием. Склонен к отслеживанию выполнения работы с целью получения замечаний. Предпочитает работать в одиночку. Если каждый участвующий проявлял бы проявлял инициативу и старался самостоятельно решить их вопрос.

Уровень освоения компетенций

_____

(ФИО студента)

Компетенции	Уровень сформированности компетенций
ПК-1 - Способен разрабатывать требования и проектировать программное обеспечение	

отметка о зачете _____

Руководитель практики от СибГУТИ:

_____ _____ _____

Должность руководителя подпись ФИО руководителя

"___" _____ 20____ г.