МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт компьютерных технологий и информационной безопасности

Кафедра систем автоматизированного проектирования

ОТЧЕТ

О прохождении производственной практики

Место прохождения практики - ООО «АйвиАппс»

Вид практики: Производственная
Тип практики: Технологическая (проектно-технологическая) практика
Способ проведения практики: Стационарная
Форма проведения практики: Дискретная (по видам практик)

Обучающийся: Тоидзе Александр Сергеевич / Очная форма
Информационные системы и технологии 09.03.02

_____
(подпись)

Руководитель практики от профильной организации: Инженер-программист ООО «АйвиАппс» Спиркова А. В.

_____
(подпись)

Руководитель практики от ЮФУ: Старший преподаватель ИКТИБ Гладкова Н.В.

_____
(подпись)

Таганрог 2023 г.

Содержание

ВВЕДЕНИЕ

Менеджер паролей является важным инструментом для обеспечения безопасности в интернете. В современном мире, когда мы все больше и больше зависим от онлайн-сервисов и цифровых устройств, защита наших личных данных становится все более актуальной задачей.

Менеджеры паролей помогают нам создавать и хранить сложные и уникальные пароли для каждого аккаунта, обеспечивая таким образом надежную защиту от кибератак и хакерских атак.

Для разработки приложения был задействован язык программирования Kotlin, который был выпущен в открытый доступ в 2012 году и стал популярным среди разработчиков благодаря своей простоте, читаемости кода, безопасности и поддержке различных платформ. Он получил поддержку от Google как официальный язык программирования для разработки Android-приложений, что привлекло ещё больше внимания к этому языку.

В данном отчете будет представлена разработка менеджера пароля.

1 АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Задачей практики являлась разработка мобильного приложения «Менеджер паролей». Также, в задачи практики входило изучение языка программирования Kotlin, ознакомление с графическим редактором Figma, а также работа с системой управления версиями GitHub.

Задачи производственной практики:

– закрепление, углубление и расширение теоретических и практических знаний, умений и навыков, полученных студентами при изучении естественнонаучных и специальных дисциплин;

– овладение профессионально-практическими умениями, производственными навыками и передовыми методами работы;

– ознакомление с проблемами и направлениями развития технологий программирования;

– изучение автоматизации проектирования, производства, испытаний и оценки качества программного обеспечения;

– закрепление, углубление и расширение теоретических знаний, умений и навыков подготовки презентации и оформления научно-технических отчетов по результатам выполненной работы, а также методических материалов и пособий по применению программных систем;

– получение практического опыта по решению задач в различных областях.

2 ИЗУЧЕНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ KOTLIN

Kotlin - это статически типизированный язык программирования, который разрабатывается компанией JetBrains. Он был создан с целью быть универсальным и современным языком программирования, который может использоваться для разработки приложений на платформе Java.

Kotlin совместим с Java, что означает, что вы можете использовать его вместе с существующими Java-кодом, библиотеками и инструментами. Он также имеет свои уникальные особенности, такие как нулевая безопасность (null safety), расширения функций (extension functions), короткие функции (lambda expressions) и многое другое.

Kotlin поддерживает объектно-ориентированное и функциональное программирование, что делает его гибким и мощным инструментом для разработки различных типов приложений, включая мобильные приложения, веб-приложения, серверные приложения и многое другое.

Kotlin также широко используется в различных областях разработки программного обеспечения и имеет активное сообщество разработчиков, что делает его популярным выбором для многих проектов. В данном случае язык Kotlin будет широко задействован для создания менеджера пароля.

3 РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Необходимо реализовать следующее:

- Экран с сохраненными паролями (Рисунок 1);
- Экран с подробностями сохраненного пароля (Рисунок 2);
- Экран менеджера паролей пользователя, которые были обнаружены в утечках паролей (Рисунок 3);
- Экран добавления нового пароля в приложение (Рисунок 4);
- Экран генерации случайного пароля (Рисунок 5);

Для создания базы данных в приложении была использована библиотека Room. Она использует в себе базу данных SQLite. Разработчику необходимо задать библиотеке как должны выглядеть данные, их структуру в базе и способы взаимодействия с помощью специальных аннотаций. Создается хранилище, в неё помещается сущность (таблица данных), а также набор методов обращения к базе. Таблица в созданной базе данных имеет значения: иконка сервиса, название сервиса, его ссылка, имя аккаунта, пароль.

Реализован экран изменения сохраненного пароля, на случай когда пользователь поменял пароль на каком-то сервисе и его необходимо актуализировать в базе, чтобы не забыть его.

На главном экране пароли будут выводиться списком с помощью обьекта RecyclerView (Рисунок 6), он является мощным инструментом для вывода больших наборов данных. Также были задействованы различные текстовые, графические обьекты, поисковая строка.

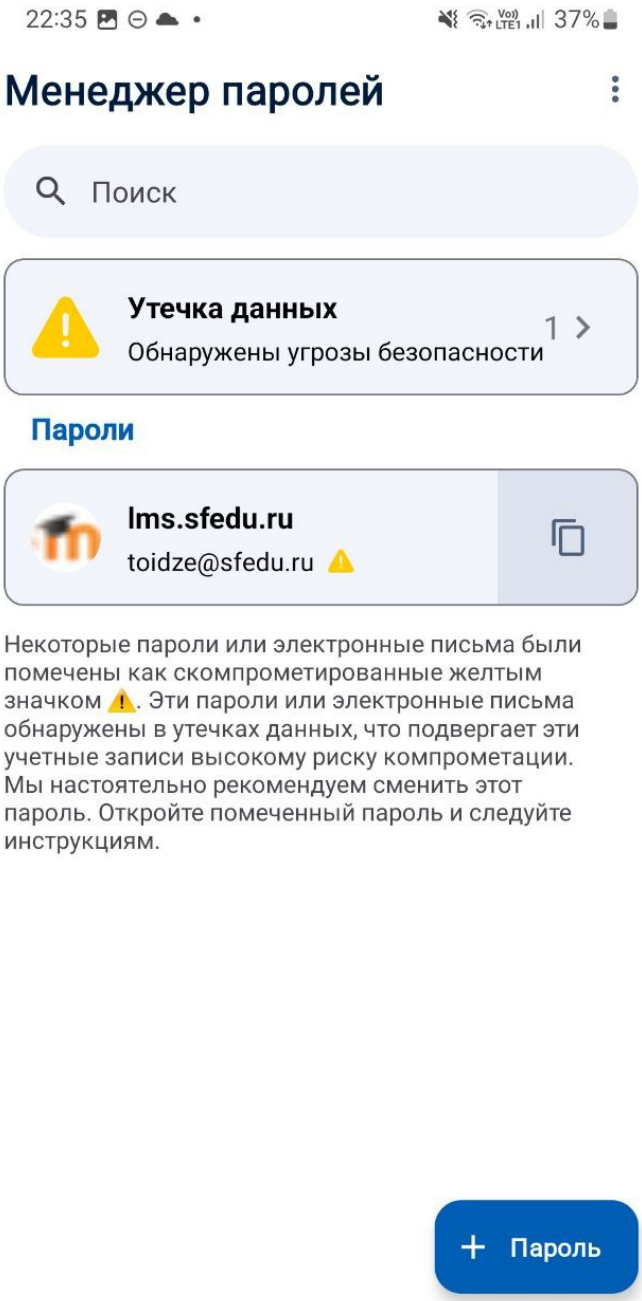

Рисунок 1 – Экран с сохраненными паролями

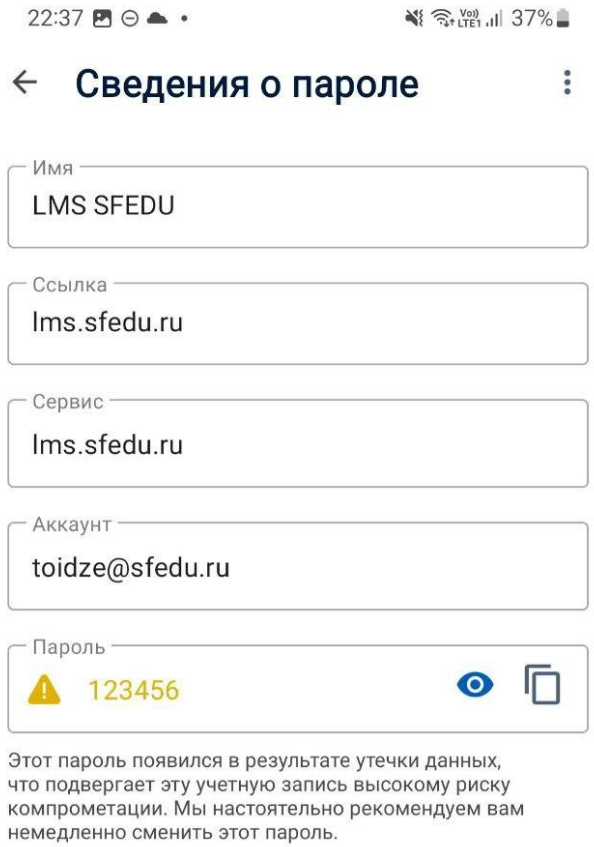

Рисунок 2 – Экран сведений о пароле

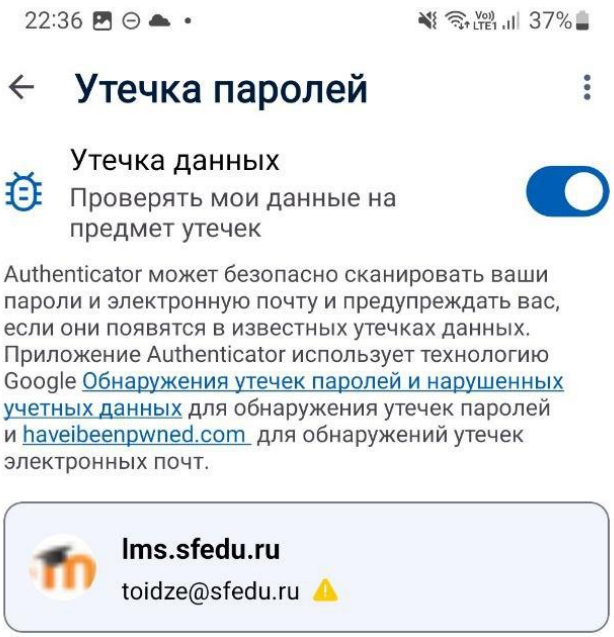

Рисунок 3 – Экран менеджера утекших сохраненных паролей

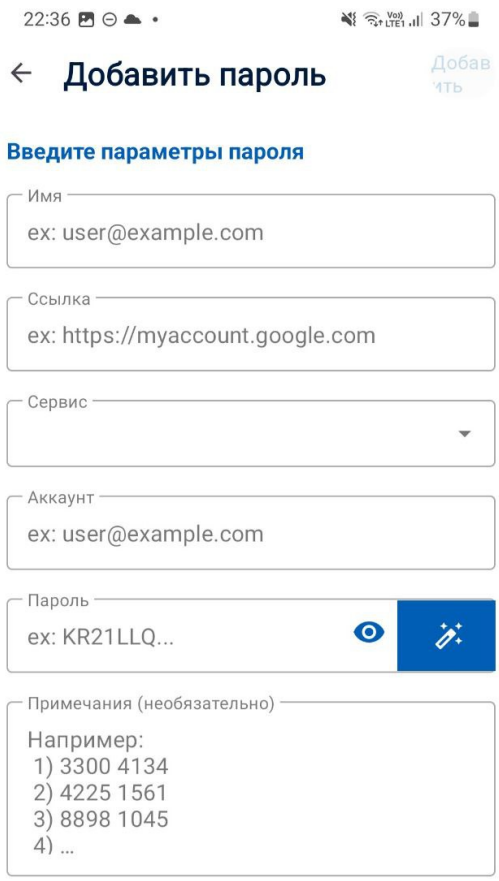

Рисунок 4 – Экран добавления нового пароля

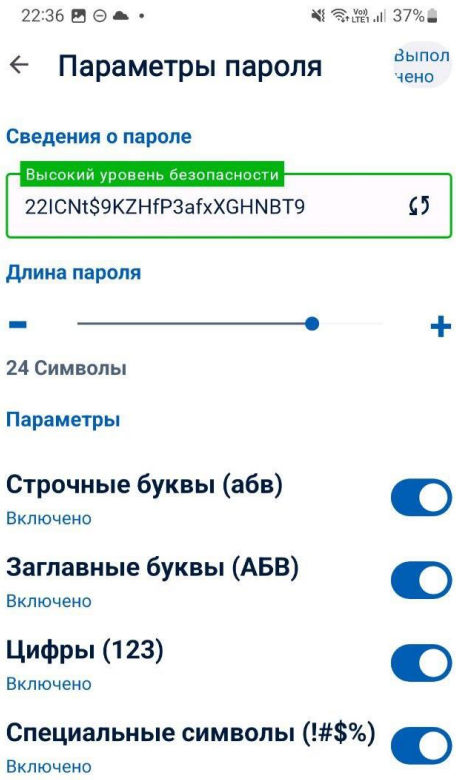

Рисунок 5 – Экран генерации нового случайного пароля

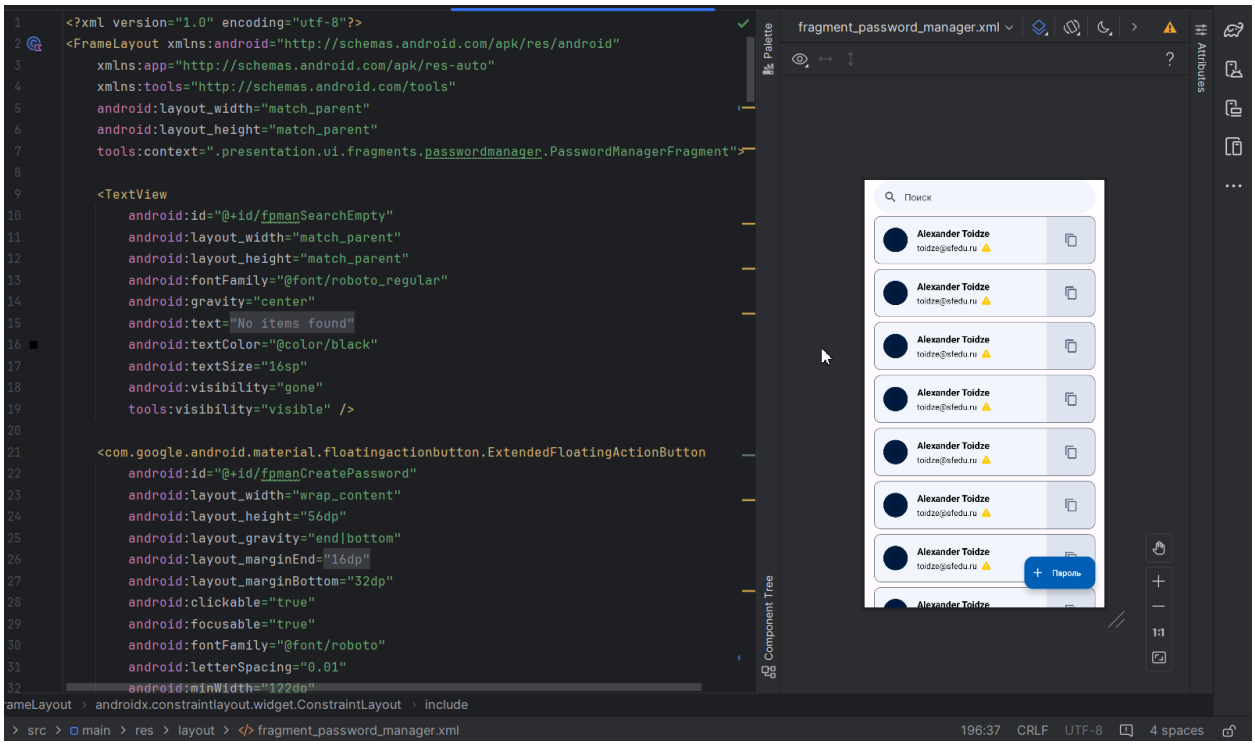

Рисунок 6 – Создание экрана вывода всех сохраненных паролей списком

Код программы приведен в Приложении.

4 ТЕСТИРОВАНИЕ И УСТРАНЕНИЕ НЕПОЛАДОК

База данных паролей была корректно реализована, запросы к ней исправно обрабатывались. Дизайн получился минималистичным, без лишних, отвлекающих элементов. При тестировании была допущена ошибка в логике поиска сохраненного пароля, она была успешно устранена. Менеджер работает исправно.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы в ходе производственной практики были выполнены следующие задачи:

- Изучение языка программирования Kotlin;
- Разработка и реализация менеджера паролей;
- Применение системы контроля версий Git;
- Применение графического редактора Figma;

В результате практики были освоены следующие компетенции: УК-2, УК-3, УК-4, ПК-1, ПК-2, ПК-3.

Цель практики, заключающаяся в разработке менеджера паролей с помощью языка программирования Kotlin, была успешно достигнута.

Таким образом, произошло знакомство с рабочим процессом разработки мобильного приложения для операционной системы Android. Цели и задачи, ставящиеся при прохождении практики, были полностью выполнены.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Kotlin — язык программирования: для чего используется, плюсы и минусы / Хабр [Электронный ресурс]. – URL: https://habr.com/ru/articles/783456/ Дата обращения: 20.03.2023.

ПРИЛОЖЕНИЕ

Экран менеджера паролей:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    viewModel.obtainEvent(viewEvent =
PasswordManagerEvents.CleanSearchPredicate)
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    searchBarHelper = SearchBarHelper(binding = binding.fpmanSearchBar)
    ToolbarEventsManager.sendEvent(ToolbarEvents.OnEditButtonState(isVisible
= false))
    searchBarHelper?.doOnSearchTextChanged(debounceInMills = 300) { string ->
        viewModel.obtainEvent(
            PasswordManagerEvents.SearchQuery(
                searchByPredicate = string
            )
        )
    }
    searchBarHelper?.doOnKeyboardEnterClicked { text ->
        viewModel.obtainEvent(
            PasswordManagerEvents.SearchQuery(
                searchByPredicate = text
            )
        )
    }
    searchBarHelper?.doOnStateChangedListener { state ->
```

```
        if (state == SearchBarState.COLLAPSED) {
            viewModel.obtainEvent(
                PasswordManagerEvents.ResetQueryParamIfNeeded
            )
        }
    }
    setupRecyclerView()
    setupActions()
    viewModel.obtainEvent(viewEvent =
PasswordManagerEvents.GetAllPassword)
    binding.fpmanCreatePassword.setOnClickListener {
        if (!preferencesManager.hasPremium &&
preferencesManager.settingsPasswordsDone >=
Config.PASSWORDS_FREE_LIMIT) {
            showPremiumFragment()
        } else {
            openAddPasswordScreen()
        }
    }

binding.fpmanSuccessState.layoutTransition.enableTransitionType(LayoutTransiti
on.CHANGING)
}

private fun setupActions() {
    repeatOnStarted {
        viewModel.viewActions().collect { action ->
            when (action) {
                is PasswordManagerActions.ShowRestoreErrorDialog -> {
```

```
                restoreBackupHelper.showNoRestoreDataDialog()
            }

            is PasswordManagerActions.ShowRestoreSuccessDialog -> {
                restoreBackupHelper.showSuccessRestoreDataDialog()
            }

            is PasswordManagerActions.UpdateAddFABState -> {
                binding.fpmanCreatePassword.beVisibleIf(action.isVisible)
            }

            else -> Unit
        }
    }
  }
}

override fun onResume() {
  searchBarHelper?.collapse {
    viewModel.obtainEvent(viewEvent =
PasswordManagerEvents.ResetQueryParamIfNeeded)
  }
  super.onResume()
  ToolbarEventsManager.addListener(listener = toolbarListener)
}

override fun onPause() {
  super.onPause()
  ToolbarEventsManager.removeListener(listener = toolbarListener)
```

```
}

private fun setupRecyclerView() = binding.apply {
    fpmanRecyclerView.addOnScrollListener(object :
RecyclerView.OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
            super.onScrolled(recyclerView, dx, dy)
            val offset = recyclerView.computeVerticalScrollOffset()
            ToolbarEventsManager.sendEvent(
                toolbarEvent = ToolbarEvents.OnScrollState(
                    isOnTop = offset <= 0
                )
            )
        }
    })
    repeatOnStarted {
        val passwordItemsAdapter =
PasswordItemAdapter(preferencesManager.isLeakedDataManagerActive)
        passwordItemsAdapter.openPasswordDetailsCallback = {
passwordDetailsCallback ->

navigate(R.id.action_passwordManagerFragment_to_viewPasswordDetailsFragm
ent)
            detailsViewModel.obtainEvent(
                viewEvent = PasswordDetailsEvents.CreatePasswordDetailsList(model
= passwordDetailsCallback)
            )
        }
        fpmanRecyclerView.apply {
```

```
        adapter = passwordItemsAdapter
        itemAnimator = null
        layoutManager = LinearLayoutManager(context)
    }

    viewModel.viewStates().collect { state ->
        fpmanSearchEmpty.beGone()
        fpmanLoadingState.beVisibleIfSmooth(state is
PasswordManagerStates.Loading)
        fpmanFirstStartState.beVisibleIfSmooth(state is
PasswordManagerStates.Empty)
        when (state) {
            is PasswordManagerStates.Success -> {

ToolbarEventsManager.sendEvent(ToolbarEvents.OnEditButtonState(isVisible =
state.result.isNotEmpty()))
                fpmanLeakedDataManager.beVisibleSmooth()
                ToolbarEventsManager.sendEvent(
                    toolbarEvent = ToolbarEvents.OnEditPasswordListState(
                        isVisibility = state.result.isNotEmpty()
                    )
                )
                if (state.result.isEmpty()) {
                    fpmanRecyclerView.beGone()
                    fpmanSearchEmpty.beVisibleSmooth()
                    return@collect
                }
                val itemsList = state.result.toMutableList()
                itemsList.add(0, SectionModel(sectionName = "Пароли"))
```

```
passwordItemsAdapter.submitList(setupLeakedDataManagerItem(itemsList))
                fpmanRecyclerView.beVisibleSmooth()
            }

            is PasswordManagerStates.Empty -> {
                ToolbarEventsManager.sendEvent(
                    toolbarEvent = ToolbarEvents.OnEditPasswordListState(
                        isVisibility = false
                    )
                )
            }

            else -> Unit
        }
    }
  }
}

private fun setupLeakedDataManagerItem(list: MutableList<Any>):
MutableList<Any> {
   binding.apply {
      if (preferencesManager.isLeakedDataManagerActive) {
         var counter = 0
         list.forEach {
            if (it is PasswordItemModel && (it.isLeakedPassword ||
it.isLeakedAccount)) counter++
         }
         fpmanLeakedManagerSubtitle.text =
```

```
getString(R.string.SecurityRisksDetected)
            fpmanLeakedManagerCounter.text = counter.toString()
            if (counter > 0) {
                list.add(list.size, TextModel(text =
getString(R.string.LeakedDataDescription)))
            }
        } else {
            fpmanLeakedManagerSubtitle.text =
getString(R.string.SecurityMonitorNotActivated)
            fpmanLeakedManagerCounter.text = ""
        }
        fpmanLeakedDataManager.setOnClickListener {

navigate(R.id.action_passwordManagerFragment_to_leakedPasswordsFragment)
        }
    }
    return list
}
```

Экран сведений о пароле:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.apply {
        setupActionsWithPasswordField()
        setupState()
        setupSecurityRecommendations(
            isLeakedPassword = if (passwordDetails.isNotEmpty())
                passwordDetails[0].isLeakedPassword else false
        )
```

```
        setupLeakReportAction()
        fvpdGetFullLeakReport.setOnClickListener {

navigate(R.id.action_viewPasswordDetailsFragment_to_leakReportFragment)

        }
    }
}

private fun setupState() {
    repeatOnCreated {
        detailsViewModel.viewStates().collect { state ->
            when (state) {
                is PasswordDetailsStates.Success -> {
                    setupTextViewsFields(passwordDetails = state.passwordDetails)
                    passwordDetails = listOf(state.passwordDetails)
                    reportViewModel.obtainEvent(
                        viewEvent = LeakReportEvents.SetupLeakReportPage(
                            passwordItem = state.passwordDetails
                        )
                    )
                    setupSecurityRecommendations(
                        isLeakedPassword = state.passwordDetails.isLeakedPassword
                    )
                    Log.d("LeakReportFragment", passwordDetails.toString())
                }
                else -> {}
            }
        }
```

```
    }
}

private fun setupLeakReportAction() {
    repeatOnCreated {
        reportViewModel.viewActions().collect { action ->
            when (action) {
                is LeakReportActions.OpenFullLeakReport -> {
                    binding.fvpdAccountLeakedInfoProgressBar.beGoneSmooth()
                    setupSecurityRecommendations(
                        isLeakedAccount = action.isOpenable,
                        isLeakedPassword = passwordDetails[0].isLeakedPassword
                    )
                    Log.d("OpenLeakReport", "true,
${passwordDetails[0].isLeakedPassword}")
                }
                else -> Unit
            }
        }
    }
}

private fun setupTextViewsFields(passwordDetails: PasswordItemModel) =
binding.apply {
    fvpdNameString.setText(passwordDetails.name)
    fvpdLinkString.setText(passwordDetails.link)
    fvpdServiceString.setText(passwordDetails.service)
    fvpdAccountString.setText(passwordDetails.account)
```

```
    fvpdPasswordString.setText(passwordDetails.password)
}
```

Экран менеджера утекших сохраненных паролей:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.apply {
        flpCheckDataForLeaksDescription.movementMethod =
LinkMovementMethod.getInstance()
        flpCheckDataForLeaksSwitch.setOnCheckedChangeListener { _, isChecked -
>
            viewModel.obtainEvent(
                viewEvent = LeakedPasswordsEvents.SetupCheckDataForLeaks(
                    isChecked = isChecked
                )
            )
        }
    }
    setupState()
}

private fun setupState() {
    repeatOnStarted {
        val leakedPasswordsAdapter = LeakedPasswordsAdapter()
        leakedPasswordsAdapter.openLeakedPasswordCallback = {
leakedPasswordCallback ->

navigate(R.id.action_leakedPasswordsFragment_to_viewPasswordDetailsFragme
nt)
```

```
            detailsViewModel.obtainEvent(
                viewEvent = PasswordDetailsEvents.CreatePasswordDetailsList(model
= leakedPasswordCallback)
            )
        }
        binding.flpLeakedPasswordsRecycler.apply {
            adapter = leakedPasswordsAdapter
            itemAnimator = null
            layoutManager = LinearLayoutManager(context)
        }

        viewModel.viewStates().collect { state ->
            when (state) {
                is LeakedPasswordsStates.Success -> {
                    val leakedPasswords = state.leakedPasswordsList
                    leakedPasswordsAdapter.submitList(leakedPasswords)
                    binding.flpCheckDataForLeaksSwitch.isChecked =
state.isCheckDataForLeaks
                    binding.flpLeakedPasswordsRecycler.beVisibleIf(condition =
state.isCheckDataForLeaks)
                    Log.d("Leaked", state.isCheckDataForLeaks.toString())
                }
                is LeakedPasswordsStates.Empty -> {
                    binding.flpCheckDataForLeaksSwitch.isChecked =
state.isCheckDataForLeaks
                }
                else -> Unit
            }
        }
```

```
    }
}
```

Экран добавления нового пароля:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    setupActions()
    setupGeneratorActions()
    binding.apply {
        KeyboardVisibilityEvent.addKeyboardEventCallback(callback =
keyboardEventCallback)
        ToolbarEventsManager.addListener(listener = listener)
        setupToolbarState(isActive = false)
        viewModel.obtainEvent(viewEvent =
PasswordManagerEvents.GetAllStrings)
        fapPasswordStringLayout.setEndIconOnClickListener {
            showPasswordGeneratorDialog()
        }
        fpgbsButtonShowPassword.setOnClickListener {
            if (fapPasswordString.inputType ==
InputType.TYPE_TEXT_VARIATION_PASSWORD or
InputType.TYPE_CLASS_TEXT) {
                fapPasswordString.inputType = InputType.TYPE_CLASS_TEXT or
InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD
                fpgbsButtonShowPassword.setImageResource(R.drawable.ic_eye_off)
            } else {
                fapPasswordString.inputType =
InputType.TYPE_TEXT_VARIATION_PASSWORD or
InputType.TYPE_CLASS_TEXT
```

```
            fpgbsButtonShowPassword.setImageResource(R.drawable.ic_eye)
        }
    }
    fapGenerateSecurePassword.setOnClickListener {
        showPasswordGeneratorDialog()
    }
    fapPasswordString.setOnFocusChangeListener { _, hasFocus ->
        if (KeyboardVisibilityEvent.keyboardIsOpened) {
            setupGenerateSecurePasswordMessageState(
                isKeyboardOpened = hasFocus,
                keyboardHeight = KeyboardVisibilityEvent.keyboardHeight
            )
        }
    }
    fapPasswordString.setText(viewModel.generatedPassword)
    fapServiceString.setOnClickListener {

AddManuallyBottomSheetDialogFragment().show(parentFragmentManager,
"Tag")
    }
    fapLinkStringLayout.editText?.doOnTextChanged { text, _, _, _ ->
        if (text?.length!! > 3) authViewModel.obtainEvent(
            viewEvent = AuthenticatorEvents.CheckIsServiceAvailable(
                url = text.toString()
            )
        )
    }
    setupLinkString()
```

```
        listOf(
            fapLinkStringLayout,
            fapNameStringLayout,
            fapAccountStringLayout,
            fapPasswordStringLayout
        ).inputsIsNotEmpty { isNotEmpty ->
            setupToolbarState(isActive = isNotEmpty)
        }
        // Hide keyboard when tapping outside of any edi text
        listOf(
            fapNameString,
            fapLinkString,
            fapAccountString,
            fapPasswordString,
            fapNotesString
        ).hideKeyboardOnNoFocus(root)

    }
}

private fun isAddButtonActive(): Boolean {
    return !binding.fapLinkStringLayout.editText?.text.isNullOrEmpty()
            && !binding.fapNameStringLayout.editText?.text.isNullOrEmpty()
            && !binding.fapAccountStringLayout.editText?.text.isNullOrEmpty()
            && !binding.fapPasswordStringLayout.editText?.text.isNullOrEmpty()
}

private fun setupActions() {
    repeatOnStarted {
```

```
        viewModel.viewActions().collect { action ->
            when (action) {
                is PasswordManagerActions.OnGetAllStrings -> {
                    binding.apply {
                        fapNameString.setText(action.allStrings.getOrNull(0))
                        fapLinkString.setText(action.allStrings.getOrNull(1))
                        fapServiceString.setText(action.allStrings.getOrNull(2))
                        fapAccountString.setText(action.allStrings.getOrNull(3))
                    }
                }
                is PasswordManagerActions.OnAddPasswordResult -> {
                    if (preferencesManager.settingsPasswordsDone <=
PASSWORDS_FREE_LIMIT-1) preferencesManager.settingsPasswordsDone += 1
                    navigateUp()
                    reviewAppManager.showRatingDialogIfNeeded()
                }
                else -> Unit
            }
        }
    }
}

private fun setupGeneratorActions() {
    repeatOnStarted {
        generatorViewModel.viewActions().collect { action ->
            when (action) {
                is PasswordGeneratorActions.SendGeneratedPassword ->
binding.fapPasswordString.setText(action.password)
                else -> Unit
```

```
        }
    }
  }
}

@SuppressLint("SetTextI18n")
private fun setupLinkString() {
  repeatOnStarted {
    authViewModel.viewActions().collect { action ->
      when (action) {
        is AuthenticatorActions.ServiceCheckingResult -> {
          linkString = action.service
          if (action.isAutoVerify) {

binding.fapServiceString.setText("${getString(R.string.auto_service)} •
$linkString")
          } else {

binding.fapServiceString.setText("${getString(R.string.manually_service)} •
$linkString")
          }
        }
        else -> Unit
      }
    }
  }
}
```

Экран генерации нового случайного пароля:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        ToolbarEventsManager.addListener(listener = listener)
        ToolbarEventsManager.sendEvent(toolbarEvent =
ToolbarEvents.OnDoneGeneratingPasswordState(true))
        setupStates()
        setupClicks()
        setupSwitcherListeners()
    }

    private fun setupSwitcherListeners() = binding.apply {
        fppLowercaseLettersSwitch.setOnCheckedChangeListener { _, isChecked ->
            viewModel.obtainEvent(
                viewEvent = PasswordGeneratorEvents.UpdateLowerCaseLettersState(
                    isIncluded = isChecked
                )
            )
            fppLowercaseLettersState.isEnabledState(isEnabled = isChecked)
        }
        fppUppercaseLettersSwitch.setOnCheckedChangeListener { _, isChecked ->
            viewModel.obtainEvent(
                viewEvent = PasswordGeneratorEvents.UpdateUpperCaseLettersState(
                    isIncluded = isChecked
                )
            )
            fppUppercaseLettersState.isEnabledState(isEnabled = isChecked)
        }
        fppNumbersSwitch.setOnCheckedChangeListener { _, isChecked ->
```

```
            viewModel.obtainEvent(
                viewEvent = PasswordGeneratorEvents.UpdateNumbersState(
                    isIncluded = isChecked
                )
            )
            fppNumbersState.isEnabledState(isEnabled = isChecked)
        }
        fppSpecialSymbolsSwitch.setOnCheckedChangeListener { _, isChecked ->
            viewModel.obtainEvent(
                viewEvent = PasswordGeneratorEvents.UpdateSpecialSymbolsState(
                    isIncluded = isChecked
                )
            )
            fppSpecialSymbolsState.isEnabledState(isEnabled = isChecked)
        }
    }

    private fun setupClicks() = binding.apply {
        fppButtonRegeneratePassword.setOnClickListener {
            viewModel.obtainEvent(viewEvent =
PasswordGeneratorEvents.GeneratePassword)
        }
    }

    private fun setupStates() {
        repeatOnStarted {
            viewModel.viewStates().collect { state ->
                when (state) {
```

```
                is PasswordGeneratorStates.Success -> setupUI(state =
state.generatorStateModel)
                else -> Unit
            }
        }
    }
}

@SuppressLint("SetTextI18n")
private fun setupUI(state: PasswordGeneratorStateModel) = binding.apply {
    includedCheckboxesCounter = state.isHowMuchCheckBoxesIncluded
    fppTextViewCode.text = state.passwordContainerState.generatedPassword
    fppPasswordCardView.strokeColor =
        getColorCompat(colorId = state.passwordContainerState.containerColorId)
    fppPasswordHint.text =
getString(state.passwordContainerState.containerHintMessageId)
    fppPasswordHint.setBackgroundColor(getColorCompat(colorId =
state.passwordContainerState.containerColorId))
    fppPasswordLengthSeekBar.let {
        it.max = state.passwordSeekbarStateModel.passwordMaxLength
        it.min = state.passwordSeekbarStateModel.passwordMinLength
        it.progress = state.passwordSeekbarStateModel.passwordLength
    }
    fppPasswordLengthHint.let {
        it.text =
            "${state.passwordSeekbarStateModel.passwordLength}
${getString(R.string.password_manager_password_generator_text_input_hint_cha
racters)}"
```

```
it.setTextColor(getColorCompat(state.passwordSeekbarStateModel.passwordLengthColorId))
        }
        fppLowercaseLettersSwitch.isChecked = state.isLowerCaseLettersIncluded
        fppUppercaseLettersSwitch.isChecked = state.isUpperCaseLettersIncluded
        fppNumbersSwitch.isChecked = state.isNumbersIncluded
        fppSpecialSymbolsSwitch.isChecked = state.isSpecialSymbolsIncluded

        fppPasswordLengthDecrease.setOnClickListener {
            fppPasswordLengthSeekBar.let {
                if (it.progress > it.min) {
                    it.progress--
                    state.passwordSeekbarStateModel.passwordLength = it.progress
                }
            }
        }

        fppPasswordLengthIncrease.setOnClickListener {
            fppPasswordLengthSeekBar.let {
                if (it.progress < it.max) {
                    it.progress++
                    state.passwordSeekbarStateModel.passwordLength = it.progress
                }
            }
        }

        fppPasswordLengthSeekBar.setOnSeekBarChangeListener(object:
SeekBar.OnSeekBarChangeListener {
```

```
            override fun onProgressChanged(seekBar: SeekBar?, progress: Int,
fromUser: Boolean) {
                state.passwordSeekbarStateModel.passwordLength =
binding.fppPasswordLengthSeekBar.progress
                viewModel.obtainEvent(
                    viewEvent =
PasswordGeneratorEvents.UpdatePasswordLengthState(length =
state.passwordSeekbarStateModel.passwordLength)
                )
                binding.fppButtonRegeneratePassword.performClick()
            }

            override fun onStartTrackingTouch(seekBar: SeekBar?) {
                Log.d("SeekBar", "onStartTrackingTouch")
            }

            override fun onStopTrackingTouch(seekBar: SeekBar?) {
                Log.d("SeekBar", "onStopTrackingTouch")
            }
        })
    }

    private fun showWeakSecurityAlertDialog() {
        context?.showCustomAlertDialog(
            title =
getString(R.string.password_manager_password_generator_weak_security_alert_ti
tle),
```

```
            subtitle =
getString(R.string.password_manager_password_generator_weak_security_alert_subtitle),
            negativeButtonText =
getString(R.string.password_manager_password_generator_weak_security_alert_use_anyway),
            positiveButtonText =
getString(R.string.password_manager_password_generator_weak_security_alert_change_password),
            positiveCallback = {},
            negativeCallback = {
                viewModel.obtainEvent(viewEvent =
PasswordGeneratorEvents.SaveGeneratedPassword(password =
binding.fppTextViewCode.text.toString()))
                navigateUp()
            }
        )
    }
```