**ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ**

**Кафедра автоматизованих систем обробки інформації і управління**

«ЗАТВЕРДЖЕНО»

В.о. завідувача кафедри

\_\_\_\_\_\_\_\_\_\_ \_\_*І.П.Муха*\_\_

(підпис) (ініціали, прізвище)

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

Розробка інтегрованої системи для соціологічних опитувань на базі

технологічного стеку Metarhia». Розробка клієнтської частини (Android- застосунок

**Опис програми**

ІАЛЦ. 04030-03-13

|  |  |
| --- | --- |
| “ПОГОДЖЕНО”  Керівник проекту:  Білоконь А.А. |  |
| Нормоконтроль:  Ліщук К.І. | Виконавці:  Іванова Л.А. |

Київ – 2018 року

**Опис програми адміністративної панелі**

// AdminProvider.kt

package com.lidaamber.adminpanel.models

import android.os.Handler

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.network.NetworkManager

import com.metarhia.jstp.handlers.OkErrorHandler

/\*\*

\* Provider of admin API

\* @author lidaamber

\*/

object AdminProvider {

/\*\*

\* Android Handler used for handling events on main thread after network responses

\*/

private val handler = Handler()

/\*\*

\* Error codes and descriptions for admin interface

\*/

val adminErrors by lazy {

mapOf(

1025 to R.string.not\_authorized,

1026 to R.string.survey\_not\_found,

1027 to R.string.question\_not\_found

)

}

/\*\*

\* Error codes and descriptions for auth interface

\*/

private val authErrors by lazy {

mapOf(

1025 to R.string.invalid\_credentials,

1026 to R.string.must\_be\_authenticated,

1027 to R.string.invalid\_token,

1028 to R.string.already\_registered,

1029 to R.string.email\_in\_use

)

}

/\*\*

\* Initializes network connection

\*/

fun init() {

NetworkManager.init()

}

/\*\*

\* Imports data about users from .csv file

\*

\* @param data data about users from .csv file

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun importUsersData(data: String, callback: () -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.IMPORT\_USERS,

arrayListOf(data), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { args?.let { callback() } }

}

})

}

/\*\*

\* Gets surveys created by current administrator

\*

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun getSurveys(callback: (MutableList<Survey>) -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.GET\_SURVEYS,

arrayListOf(), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { args?.let { callback(JSTPConverter.getSurveys(it)) } }

}

})

}

/\*\*

\* Creates new survey

\*

\* @param title survey title

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun createSurvey(title: String, callback: (Survey) -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.CREATE\_SURVEY,

arrayListOf(title, null, null), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post {

callback(Survey(args!![0] as Int, title, ArrayList()))

}

}

})

}

/\*\*

\* Deletes survey

\*

\* @param survey survey to be deleted

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun deleteSurvey(survey: Survey, callback: () -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.DELETE\_SURVEY,

arrayListOf(survey.id), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Gets questions of survey with specified id

\*

\* @param id survey identifier

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun getSurveyQuestions(id: Int, callback: (MutableList<Question>?) -> Unit,

errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.GET\_QUESTIONS,

arrayListOf(id), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !adminErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post {

val questions = JSTPConverter.getQuestions(args)

callback(questions)

}

}

})

}

/\*\*

\* Deletes question by specified index from survey

\*

\* @param survey survey

\* @param questionIndex index of question to be deleted

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun deleteQuestion(survey: Survey, questionIndex: Int, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.DELETE\_QUESTION,

arrayListOf(survey.id, questionIndex), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Updates survey info

\*

\* @param id survey identifier

\* @param title survey title

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun updateSurvey(id: Int, title: String, groups: List<String>, callback: () -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.EDIT\_SURVEY,

arrayListOf(id, title, groups), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

fun getGroups(callback: (List<String>) -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.GET\_GROUPS,

arrayListOf(), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback(args!![0] as List<String>) }

}

})

}

/\*\*

\* Creates new question in survey

\*

\* @param surveyId survey identifier

\* @param question question

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun createQuestion(surveyId: Int, question: Question, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.CREATE\_QUESTION,

arrayListOf(surveyId, JSTPConverter.createQuestion(question)), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Updates question in survey

\*

\* @param surveyId survey identifier

\* @param questionIndex index of question to be updated

\* @param question question

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun updateQuestion(surveyId: Int, questionIndex: Int, question: Question, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.EDIT\_QUESTION,

arrayListOf(surveyId, questionIndex, JSTPConverter.createQuestion(question)),

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Gets results for survey

\*

\* @param id survey id

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun getResultsForSurvey(id: Int, callback: (Map<String, List<Result>>) -> Unit,

errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.ADMIN\_INTERFACE, NetworkManager.GET\_RESULTS,

arrayListOf(id), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(adminErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post {

args?.let {

val results = JSTPConverter.getResults(it)

getSurveyQuestions(id, { questions ->

if (questions == null) {

callback(mapOf())

return@getSurveyQuestions

}

val surveyResults = results.mapIndexed { index, resultArray ->

return@mapIndexed Pair(questions[index].title, resultArray)

}.toMap()

callback(surveyResults)

}, errorCallback)

}

}

}

})

}

/\*\*

\* Signs in to administrators account

\*

\* @param email email

\* @param password password

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun signIn(email: String, password: String, callback: () -> Unit, errorCallback: (Int) -> Unit) {

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, NetworkManager.LOGIN,

JSTPConverter.createCredentials(email, password),

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

}

// JSTPConverter.kt

package com.lidaamber.adminpanel.models

/\*\*

\* Converter for JSTP arguments

\* @author lidaamber

\*/

object JSTPConverter {

/\*\*

\* Constants used to form JSTP messages' arguments

\*/

private const val ID = "id"

private const val TITLE = "title"

private const val TYPE = "type"

private const val ANSWERS = "answers"

private const val MAX = "max"

private const val MIN = "min"

private const val INFO = "info"

private const val EMAIL = "email"

private const val GROUPS = "groups"

/\*\*

\* Map of dependencies between question types and corresponding constants

\*/

private val typesValues by lazy {

mapOf(QuestionType.SELECT\_ONE to "chooseOne",

QuestionType.SELECT\_MANY to "chooseMany",

QuestionType.NUMBER\_INPUT to "number",

QuestionType.TEXT\_INPUT to "text")

}

private val valuesTypes by lazy {

mapOf("chooseOne" to QuestionType.SELECT\_ONE,

"chooseMany" to QuestionType.SELECT\_MANY,

"number" to QuestionType.NUMBER\_INPUT,

"text" to QuestionType.TEXT\_INPUT)

}

/\*\*

\* Gets surveys list from JSTP arguments

\*/

fun getSurveys(args: MutableList<\*>): MutableList<Survey> {

val surveysList = args[0] as? List<\*>

return surveysList?.map {

val surveyData = it as? Map<String, Any>

val survey = Survey(id = surveyData?.get(ID) as Int,

title = surveyData[TITLE] as String)

if (surveyData.containsKey(GROUPS)) {

survey.groups = surveyData[GROUPS] as MutableList<String>

}

return@map survey

}?.toMutableList() ?: arrayListOf()

}

/\*\*

\* Gets results from JSTP arguments

\*/

fun getResults(args: MutableList<\*>): List<List<Result>> {

val resultsList = args[0] as? List<\*>

return resultsList?.map { resultsQuestion ->

val resultsForQuestion = resultsQuestion as List<\*>

resultsForQuestion.map {

val resultData = it as? List<Any>

return@map Result(resultData!![0].toString(), resultData[1] as Int)

}

} ?: arrayListOf()

}

/\*\*

\* Creates question structure for JSTP arguments

\*/

fun createQuestion(question: Question): Map<String, Any> {

val questionMap = mutableMapOf<String, Any>(TITLE to question.title,

TYPE to typesValues[question.type]!!)

question.answers?.let {

questionMap[ANSWERS] = it

}

question.max?.let {

questionMap[MAX] = it

}

question.min?.let {

questionMap[MIN] = it

}

return questionMap

}

/\*\*

\* Gets questions from JSTP arguments

\*/

fun getQuestions(args: MutableList<\*>?): MutableList<Question>? {

if (args == null) return arrayListOf()

if (args[0] == null) return null

val questionsList = args[0] as? List<\*>

return questionsList?.map {

val questionData = it as? Map<\*, \*>

valuesTypes[questionData?.get(TYPE)]

return@map Question(title = questionData?.get(TITLE) as String,

type = valuesTypes[questionData.get(TYPE) as String]!!,

answers = questionData[ANSWERS] as? MutableList<String>?,

info = questionData[INFO] as? Map<String, String>?)

}?.toMutableList() ?: arrayListOf()

}

/\*\*

\* Creates credentials structure for JSTP arguments

\*/

fun createCredentials(email: String, password: String): List<Any> {

return listOf(mapOf(EMAIL to email), password)

}

}

// Models.kt

package com.lidaamber.adminpanel.models

import java.io.Serializable

/\*\*

\* Survey model

\* @author lidaamber

\*/

data class Survey(var id: Int,

var title: String,

var groups: MutableList<String>? = null) : Serializable

/\*\*

\* Question type

\* @author lidaamber

\*/

enum class QuestionType(val id: Int) {

TEXT\_INPUT(0),

NUMBER\_INPUT(1),

SELECT\_ONE(2),

SELECT\_MANY(3)

}

/\*\*

\* Question model

\* @author lidaamber

\*/

data class Question(var title: String,

var type: QuestionType,

var min: Double? = null,

var max: Double? = null,

var answers: MutableList<String>? = null,

val info: Map<String, String>? = null) : Serializable

/\*\*

\* Result model

\* @author lidaamber

\*/

data class Result(var answer: String, var amount: Int)

// NetworkAvailabilityManager.kt

package com.lidaamber.adminpanel.network

import android.content.Context

import android.net.ConnectivityManager

/\*\*

\* Manager for tracking network state and sending corresponding network events

\* @author lidaamber

\*/

class NetworkAvailabilityManager(context: Context) {

/\*\*

\* Connectivity manager

\*/

private val manager: ConnectivityManager = context.getSystemService(Context.CONNECTIVITY\_SERVICE) as ConnectivityManager

/\*\*

\* Network events listener

\*/

private var listener: NetworkListener? = null

/\*\*

\* Network state

\*/

private var currentNetworkState: Boolean = false

/\*\*

\* Network availability state

\*/

val isNetworkAvailable: Boolean

get() {

val activeNetwork = manager.activeNetworkInfo

return activeNetwork != null && activeNetwork.isConnectedOrConnecting

}

/\*\*

\* Network state change event handler

\*/

fun onNetworkStateChanged() {

val lastState = currentNetworkState

currentNetworkState = isNetworkAvailable

if (listener == null) return

if (currentNetworkState && lastState != currentNetworkState) {

listener!!.onConnected()

} else if (!currentNetworkState && lastState != currentNetworkState) {

listener!!.onDisconnected()

}

}

/\*\*

\* Sets network state on resume

\*/

fun onResume() {

currentNetworkState = isNetworkAvailable

}

/\*\*

\* Sets network availability listener

\*/

fun setNetworkAvailabilityListener(listener: NetworkListener) {

this.listener = listener

}

/\*\*

\* Network events listener

\*/

interface NetworkListener {

/\*\*

\* Handler for connection network event

\*/

fun onConnected()

/\*\*

\* Handler for disconnection network event

\*/

fun onDisconnected()

}

}

// NetworkManager.kt

package com.lidaamber.adminpanel.network

import com.lidaamber.adminpanel.BuildConfig

import com.metarhia.jstp.connection.Connection

import com.metarhia.jstp.connection.SimpleConnectionListener

import com.metarhia.jstp.handlers.OkErrorHandler

import com.metarhia.jstp.transport.TCPTransport

/\*\*

\* Manager to work with network requests

\* @author lidaamber

\*/

object NetworkManager : SimpleConnectionListener() {

/\*\*

\* Server settings

\*/

private const val HOST = BuildConfig.HOST

private const val PORT = BuildConfig.PORT

/\*\*

\* JSTP application name

\*/

private const val APPLICATION\_NAME = "survey"

/\*\*

\* JSTP connection

\*/

private var connection: Connection? = null

/\*\*

\* Interfaces and methods constants

\*/

const val ADMIN\_INTERFACE = "admin"

const val CREATE\_SURVEY = "createSurvey"

const val GET\_SURVEYS = "getCreatedSurveys"

const val DELETE\_SURVEY = "deleteSurvey"

const val EDIT\_SURVEY = "editSurvey"

const val GET\_RESULTS = "getResults"

const val CREATE\_QUESTION = "createQuestion"

const val EDIT\_QUESTION = "editQuestion"

const val DELETE\_QUESTION = "deleteQuestion"

const val GET\_QUESTIONS = "getQuestions"

const val AUTH\_INTERFACE = "auth"

const val LOGIN = "login"

const val IMPORT\_USERS = "importUsers"

const val GET\_GROUPS = "getGroups"

/\*\*

\* Initializes network connection

\*/

fun init() {

if (connection == null) {

connection = Connection(TCPTransport(HOST, PORT, true))

connection?.addListener(this)

connection?.setReconnectCallback { \_, transportConnector ->

transportConnector.connect(null)

}

connection?.connect(APPLICATION\_NAME)

} else {

connection?.let {

if (it.isConnected) return@let

it.connect(connection?.appData)

}

}

}

/\*\*

\* Makes resendable call

\*/

fun call(interfaceName: String, method: String, arguments: List<Any?>,

okErrorHandler: OkErrorHandler) {

connection?.callResendable(interfaceName, method, arguments, okErrorHandler)

}

}

// BasePresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.network.NetworkAvailabilityManager

import com.lidaamber.adminpanel.views.NetworkAvailabilityView

/\*\*

\* Base presenter for views

\* @author lidaamber

\*/

interface BasePresenter : NetworkAvailabilityPresenter

/\*\*

\* BasePresenter implementation

\* @author lidaamber

\*/

open class BasePresenterImpl(val context: Context,

private val networkView: NetworkAvailabilityView) : BasePresenter,

NetworkAvailabilityManager.NetworkListener {

/\*\*

\* Network availability manager

\*/

protected val networkAvailabilityManager = NetworkAvailabilityManager(context)

override fun onResume() {

networkAvailabilityManager.onResume()

}

override fun init() {

networkAvailabilityManager.setNetworkAvailabilityListener(this)

}

override fun onNetworkStateChanged() {

networkAvailabilityManager.onNetworkStateChanged()

}

override fun onConnected() {

AdminProvider.init()

}

override fun onDisconnected() {

networkView.showNotConnectedMessage()

}

}

// EditQuestionPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.models.Question

import com.lidaamber.adminpanel.models.QuestionType

import com.lidaamber.adminpanel.views.EditQuestionView

/\*\*

\* Presenter for creating and editing questions functionality

\* @author lidaamber

\*/

interface EditQuestionPresenter : BasePresenter {

/\*\*

\* Sets survey id

\*/

fun setSurveyId(surveyId: Int)

/\*\*

\* Sets mode (new or editing)

\*/

fun setMode(modeNew: Boolean)

/\*\*

\* Sets survey question to be edited (for corresponding mode)

\*/

fun setSurveyQuestion(question: Question, index: Int)

/\*\*

\* Sets minimum range for input questions

\*/

fun setMinRange(min: Double?)

/\*\*

\* Sets maximum range for input questions

\*/

fun setMaxRange(max: Double?)

/\*\*

\* Event handler for text input question type check

\*/

fun onTextInputQuestionClicked()

/\*\*

\* Event handler for number input question type check

\*/

fun onNumberInputQuestionClicked()

/\*\*

\* Event handler for choose one question type check

\*/

fun onChooseOneQuestionClicked()

/\*\*

\* Event handler for choose many question type check

\*/

fun onChooseManyQuestionClicked()

/\*\*

\* Event handler for question confirmation click

\*/

fun onConfirmQuestionClicked(title: String)

/\*\*

\* Event handler for add answer click

\*/

fun onAddAnswerClicked()

/\*\*

\* Event handler for confirmation of creating answer

\*/

fun onCreateAnswerConfirmed(answer: String)

/\*\*

\* Event handler for deleting answer

\*/

fun onDeleteAnswerClicked(answer: String)

}

/\*\*

\* EditQuestionPresenter implementation

\*/

class EditQuestionPresenterImpl(context: Context, val view: EditQuestionView) : EditQuestionPresenter,

BasePresenterImpl(context, view) {

/\*\*

\* Survey identifier

\*/

var id: Int = 0

/\*\*

\* Question index

\*/

private var index: Int? = null

/\*\*

\* New question status

\*/

private var modeNew: Boolean = true

/\*\*

\* Question to be edited

\*/

lateinit var question: Question

override fun setMode(modeNew: Boolean) {

this.modeNew = modeNew

if (modeNew) {

question = Question("", QuestionType.TEXT\_INPUT)

view.setQuestionType(question.type)

}

}

override fun setSurveyQuestion(question: Question, index: Int) {

this.question = question

this.index = index

view.setTitle(question.title)

view.setQuestionType(question.type)

question.answers?.let {

view.displayAnswers(it)

}

question.min?.let {

view.setMinRange(it)

}

question.max?.let {

view.setMaxRange(it)

}

}

override fun setMinRange(min: Double?) {

question.min = min

}

override fun setMaxRange(max: Double?) {

question.max = max

}

override fun onTextInputQuestionClicked() {

question.type = QuestionType.TEXT\_INPUT

question.answers = null

view.displayAnswers(ArrayList())

view.setInputQuestionOptionsEnabled(true)

view.setChooseQuestionOptionsEnabled(false)

}

override fun onNumberInputQuestionClicked() {

question.type = QuestionType.NUMBER\_INPUT

question.answers = null

view.displayAnswers(ArrayList())

view.setInputQuestionOptionsEnabled(true)

view.setChooseQuestionOptionsEnabled(false)

}

override fun onChooseOneQuestionClicked() {

question.type = QuestionType.SELECT\_ONE

question.min = null

question.max = null

if (question.answers == null) question.answers = ArrayList()

view.displayAnswers(question.answers!!)

view.setInputQuestionOptionsEnabled(false)

view.setChooseQuestionOptionsEnabled(true)

}

override fun onChooseManyQuestionClicked() {

question.type = QuestionType.SELECT\_MANY

question.min = null

question.max = null

if (question.answers == null) question.answers = ArrayList()

view.displayAnswers(question.answers!!)

view.setInputQuestionOptionsEnabled(false)

view.setChooseQuestionOptionsEnabled(true)

}

override fun onConfirmQuestionClicked(title: String) {

question.title = title

if (modeNew) AdminProvider.createQuestion(id, question, {

view.close()

}, { errorId ->

view.displayMessage(context.getString(errorId))

}) else AdminProvider.updateQuestion(id, index!!, question, {

view.close()

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun setSurveyId(surveyId: Int) {

this.id = surveyId

}

override fun onAddAnswerClicked() {

view.openAddAnswerDialog()

}

override fun onCreateAnswerConfirmed(answer: String) {

question.answers?.add(answer)

view.updateAnswersList()

}

override fun onDeleteAnswerClicked(answer: String) {

question.answers?.remove(answer)

view.updateAnswersList()

}

}

// EditSurveyPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.models.Question

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.views.EditSurveyView

/\*\*

\* Presenter for creating and editing survey functionality

\* @author lidaamber

\*/

interface EditSurveyPresenter : BasePresenter {

/\*\*

\* Sets survey info and mode

\*/

fun setSurveyMode(survey: Survey, modeNew: Boolean)

/\*\*

\* Event handler for add question click

\*/

fun onAddQuestionClicked()

/\*\*

\* Event handler for survey confirmation click

\*/

fun onConfirmButtonClicked(title: String)

/\*\*

\* Event handler for question edit click

\*/

fun onEditQuestionClicked(question: Question)

/\*\*

\* Event handler for delete question click

\*/

fun onRemoveQuestionClicked(question: Question)

/\*\*

\* Event handler for delete group click

\*/

fun onGroupDeleteClicked(group: String)

/\*\*

\* Event handler for add group button click

\*/

fun onAddGroupClicked()

/\*\*

\* Event handler for choosing group

\*/

fun onChooseGroupConfirmed(group: String)

}

/\*\*

\* Presenter for creating and editing survey functionality

\* @author lidaamber

\*/

class EditSurveyPresenterImpl(context: Context, val view: EditSurveyView) : EditSurveyPresenter,

BasePresenterImpl(context, view) {

/\*\*

\* Survey to be edited

\*/

lateinit var survey: Survey

/\*\*

\* New survey status

\*/

private var modeNew: Boolean = false

/\*\*

\* Survey questions

\*/

private lateinit var questions: MutableList<Question>

private lateinit var groups: MutableList<String>

override fun setSurveyMode(survey: Survey, modeNew: Boolean) {

this.survey = survey

this.modeNew = modeNew

view.displayTitle(survey.title)

if (modeNew) {

questions = ArrayList()

}

AdminProvider.getSurveyQuestions(survey.id, {

questions = it ?: arrayListOf()

view.displaySurveyQuestions(questions)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

this.groups = survey.groups ?: ArrayList()

view.displayGroups(groups)

}

override fun onAddQuestionClicked() {

view.showNewQuestionScreen(survey.id)

}

override fun onConfirmButtonClicked(title: String) {

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.updateSurvey(survey.id, title, this.groups, {

view.close()

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onEditQuestionClicked(question: Question) {

view.showEditQuestionScreen(survey.id, question, questions.indexOf(question))

}

override fun onGroupDeleteClicked(group: String) {

groups.remove(group)

view.updateGroupsList()

}

override fun onRemoveQuestionClicked(question: Question) {

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.deleteQuestion(survey, questions.indexOf(question), {

questions.remove(question)

view.updateSurveyQuestions()

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onAddGroupClicked() {

AdminProvider.getGroups({

view.openGroupsList(it)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onChooseGroupConfirmed(group: String) {

groups.add(group)

view.updateGroupsList()

}

}

// ImportUsersPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.app.Activity

import android.content.Context

import android.content.Intent

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.ui.extensions.getFileContent

import com.lidaamber.adminpanel.views.ImportUsersView

/\*\*

\* Presenter for importing users functionality

\* @author lidaamber

\*/

interface ImportUsersPresenter : BasePresenter {

/\*\*

\* Event handler for import click

\*/

fun onImportClicked()

/\*\*

\* Processes activity result to get chosen file content

\*/

fun processActivityResult(requestCode: Int, resultCode: Int, data: Intent?)

}

class ImportUsersPresenterImpl(context: Context, val view: ImportUsersView) :

ImportUsersPresenter, BasePresenterImpl(context, view) {

companion object {

/\*\*

\* Request code for getting .csv file

\*/

const val REQUEST\_CSV = 0

}

override fun onImportClicked() {

view.openFileChooser(REQUEST\_CSV)

}

override fun processActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

if (requestCode != REQUEST\_CSV || resultCode != Activity.RESULT\_OK) return

data?.let {

AdminProvider.importUsersData(it.data.getFileContent(context), {

view.displayMessage(context.getString(R.string.upload\_successful))

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

}

}

// LoginPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.views.LoginView

/\*\*

\* @author lidaamber

\*/

interface LoginPresenter : BasePresenter {

fun onSignInClicked(email: String, password: String)

}

class LoginPresenterImpl(context: Context, val view: LoginView) : LoginPresenter,

BasePresenterImpl(context, view) {

override fun init() {

super.init()

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.init()

}

override fun onSignInClicked(email: String, password: String) {

if (!validateFields(email, password)) return

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.signIn(email, password, {

view.showMainScreen()

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

private fun validateFields(email: String, password: String): Boolean {

var valid = true

if (email.isEmpty()) valid = false

view.setEmailErrorVisible(email.isEmpty())

if (password.isEmpty()) valid = false

view.setPasswordErrorVisible(password.isEmpty())

return valid

}

}

// NetworkAvailabilityPresenter.kt

package com.lidaamber.adminpanel.presenters

/\*\*

\* Presenter to track network availability

\* @author lidaamber

\*/

interface NetworkAvailabilityPresenter {

/\*\*

\* Sets network state on resume

\*/

fun onResume()

/\*\*

\* Initializes network availability manager

\*/

fun init()

/\*\*

\* Network state change event handler

\*/

fun onNetworkStateChanged()

}

// ResultsPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.views.ResultsView

/\*\*

\* Presenter for survey results functionality

\* @author lidaamber

\*/

interface ResultsPresenter : BasePresenter {

/\*\*

\* Sets survey to display results for

\*/

fun setSurvey(survey: Survey)

}

class ResultsPresenterImpl(context: Context, val view: ResultsView) : ResultsPresenter,

BasePresenterImpl(context, view) {

/\*\*

\* Survey id

\*/

var id: Int = 0

override fun setSurvey(survey: Survey) {

this.id = survey.id

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.getResultsForSurvey(id, {

view.displayResults(it)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onConnected() {

super.onConnected()

AdminProvider.getResultsForSurvey(id, {

view.displayResults(it)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

}

// SurveyListPresenter.kt

package com.lidaamber.adminpanel.presenters

import android.content.Context

import com.lidaamber.adminpanel.models.AdminProvider

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.views.SurveyListView

/\*\*

\* Presenter for working with surveys

\* @author lidaamber

\*/

interface SurveyListPresenter : BasePresenter {

/\*\*

\* Event handler for add survey click

\*/

fun onAddSurveyClicked()

/\*\*

\* Event handler for survey creation confirmation

\*/

fun onSurveyCreationConfirmed(title: String)

/\*\*

\* Event handler for edit survey click

\*/

fun onEditSurveyClicked(survey: Survey)

/\*\*

\* Event handler for survey results click

\*/

fun onSurveyResultsClicked(survey: Survey)

/\*\*

\* Event handler for delete survey click

\*/

fun onDeleteSurveyClicked(survey: Survey)

}

/\*\*

\* SurveyListPresenter implementation

\*/

class SurveyListPresenterImpl(context: Context, val view: SurveyListView) :

SurveyListPresenter, BasePresenterImpl(context, view) {

/\*\*

\* Surveys list

\*/

lateinit var surveys: MutableList<Survey>

override fun onResume() {

super.onResume()

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.getSurveys({

surveys = it

view.displaySurveys(surveys)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onAddSurveyClicked() {

view.showAddSurveyDialog()

}

override fun onSurveyCreationConfirmed(title: String) {

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.createSurvey(title, {

view.showNewSurveyScreen(it)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onEditSurveyClicked(survey: Survey) {

view.showEditSurveyScreen(survey)

}

override fun onSurveyResultsClicked(survey: Survey) {

view.showResultsScreen(survey)

}

override fun onDeleteSurveyClicked(survey: Survey) {

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AdminProvider.deleteSurvey(survey, {

surveys.remove(survey)

view.updateSurveysList()

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

override fun onConnected() {

super.onConnected()

AdminProvider.getSurveys({

surveys = it

view.displaySurveys(surveys)

}, { errorId ->

view.displayMessage(context.getString(errorId))

})

}

}

// AnswersAdapter.kt

package com.lidaamber.adminpanel.ui.adapters

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import kotlinx.android.synthetic.main.answer\_list\_item.view.\*

/\*\*

\* Adapter for answers list

\* @author lidaamber

\*/

class AnswersAdapter(val answers: List<String>,

val onAnswerDeleteClicked: (String) -> Unit) : RecyclerView.Adapter<AnswersAdapter.AnswerHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): AnswerHolder {

val v = LayoutInflater.from(parent.context).inflate(R.layout.answer\_list\_item, parent, false)

return AnswerHolder((v))

}

override fun getItemCount() = answers.size

override fun onBindViewHolder(holder: AnswerHolder, position: Int) {

holder.bind(answers[position])

}

inner class AnswerHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

fun bind(answer: String) {

itemView.answerTitle.text = answer

itemView.deleteAnswerButton.setOnClickListener { onAnswerDeleteClicked(answer) }

}

}

}

// InfoAdapter.kt

package com.lidaamber.adminpanel.ui.adapters

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import kotlinx.android.synthetic.main.info\_item.view.\*

/\*\*

\* @author lidaamber

\*/

class InfoAdapter(private val infoList: List<String>,

val onItemClicked: (String) -> Unit) : RecyclerView.Adapter<InfoAdapter.Holder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {

val v = LayoutInflater.from(parent.context).inflate(R.layout.info\_item, parent, false)

return Holder(v)

}

override fun getItemCount() = infoList.size

override fun onBindViewHolder(holder: Holder, position: Int) {

holder.bind(infoList[position])

}

inner class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {

fun bind(info: String) {

itemView.infoTextView.text = info

itemView.setOnClickListener { onItemClicked(info) }

}

}

}

// QuestionsAdapter.kt

package com.lidaamber.adminpanel.ui.adapters

import android.content.Context

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Question

import com.lidaamber.adminpanel.models.QuestionType

import kotlinx.android.synthetic.main.question\_list\_item.view.\*

/\*\*

\* Adapter for questions list

\* @author lidaamber

\*/

class QuestionsAdapter(val context: Context,

private val questions: List<Question>,

val onQuestionOptionsClicked: (Question, View) -> Unit) :

RecyclerView.Adapter<QuestionsAdapter.QuestionHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): QuestionHolder {

val v = LayoutInflater.from(parent.context).inflate(R.layout.question\_list\_item, parent, false)

return QuestionHolder(v)

}

override fun getItemCount() = questions.size

override fun onBindViewHolder(holder: QuestionHolder, position: Int) {

holder.bind(questions[position])

}

inner class QuestionHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

fun bind(question: Question) {

itemView.questionTitle.text = question.title

itemView.questionType.text = context.getString(when (question.type) {

QuestionType.TEXT\_INPUT -> R.string.text\_input

QuestionType.NUMBER\_INPUT -> R.string.number\_input

QuestionType.SELECT\_MANY -> R.string.many\_from\_many

QuestionType.SELECT\_ONE -> R.string.one\_from\_many

})

itemView.optionsButton.setOnClickListener { onQuestionOptionsClicked(question, itemView) }

}

}

}

// ResultsAdapter.kt

package com.lidaamber.adminpanel.ui.adapters

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Result

import kotlinx.android.synthetic.main.result\_header\_list\_item.view.\*

import kotlinx.android.synthetic.main.result\_list\_item.view.\*

/\*\*

\* Adapter for results list

\* @author lidaamber

\*/

class ResultsAdapter(val results: Map<String, List<Result>>) : RecyclerView.Adapter<ResultsAdapter.Holder>() {

companion object {

const val HEADER = 0

const val RESULT = 1

}

private val flattenResults: List<Any> by lazy {

val list = ArrayList<Any>()

results.forEach {

list.add(it.key)

it.value.forEach { value -> list.add(value) }

}

return@lazy list

}

private val headerPositions: List<Int> by lazy {

val list = ArrayList<Int>()

var lastQuestionIndex = 0

results.forEach { \_, results ->

list.add(lastQuestionIndex)

lastQuestionIndex += 1 + results.size

}

return@lazy list

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {

return if (viewType == RESULT) {

val v = LayoutInflater.from(parent.context)

.inflate(R.layout.result\_list\_item, parent, false)

ResultHolder(v)

} else {

val v = LayoutInflater.from(parent.context)

.inflate(R.layout.result\_header\_list\_item, parent, false)

QuestionHolder(v)

}

}

override fun getItemCount(): Int = flattenResults.size

override fun onBindViewHolder(holder: Holder, position: Int) {

if (holder is QuestionHolder) holder.bind(flattenResults[position] as String)

else (holder as ResultHolder).bind(flattenResults[position] as Result)

}

override fun getItemViewType(position: Int): Int {

return if (headerPositions.contains(position)) HEADER else RESULT

}

open class Holder(itemView: View) : RecyclerView.ViewHolder(itemView)

class QuestionHolder(itemView: View) : Holder(itemView) {

fun bind(question: String) {

itemView.questionTitle.text = question

}

}

class ResultHolder(itemView: View) : Holder(itemView) {

fun bind(result: Result) {

itemView.resultAnswer.text = result.answer

itemView.resultAmount.text = result.amount.toString()

}

}

}

// SurveysAdapter.kt

package com.lidaamber.adminpanel.ui.adapters

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Survey

import kotlinx.android.synthetic.main.survey\_list\_item.view.\*

/\*\*

\* Adapter for surveys list

\* @author lidaamber

\*/

class SurveysAdapter(val surveys: List<Survey>,

val onSurveyOptionsClicked: (Survey, View) -> Unit) :

RecyclerView.Adapter<SurveysAdapter.SurveyHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SurveyHolder {

val v = LayoutInflater.from(parent.context).inflate(R.layout.survey\_list\_item, parent, false)

return SurveyHolder(v)

}

override fun getItemCount() = surveys.size

override fun onBindViewHolder(holder: SurveyHolder, position: Int) {

holder.bind(surveys[position])

}

inner class SurveyHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

fun bind(survey: Survey) {

itemView.surveyTitle.text = survey.title

itemView.optionsButton.setOnClickListener { onSurveyOptionsClicked(survey, itemView) }

}

}

}

// BaseActivity.kt

package com.lidaamber.adminpanel.ui.base

import android.content.BroadcastReceiver

import android.content.Context

import android.content.Intent

import android.content.IntentFilter

import android.net.ConnectivityManager

import android.support.v7.app.AppCompatActivity

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.presenters.BasePresenter

import com.lidaamber.adminpanel.ui.extensions.showMessage

import com.lidaamber.adminpanel.views.BaseView

/\*\*

\* BaseView implementation for Activity

\* @author lidaamber

\*/

open class BaseActivity<T : BasePresenter> : AppCompatActivity(), BaseView {

/\*\*

\* Presenter

\*/

protected lateinit var presenter: T

/\*\*

\* Network events receiver

\*/

private lateinit var networkReceiver: BroadcastReceiver

override fun onStart() {

super.onStart()

networkReceiver = object : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

presenter.onNetworkStateChanged()

}

}

registerReceiver(networkReceiver,

IntentFilter(ConnectivityManager.CONNECTIVITY\_ACTION))

}

override fun showNotConnectedMessage() {

baseContext?.let {

window.decorView?.showMessage(it.getString(R.string.not\_connected))

}

}

override fun onResume() {

super.onResume()

presenter.onResume()

}

override fun onStop() {

super.onStop()

unregisterReceiver(networkReceiver)

}

override fun displayMessage(message: String) {

window.decorView?.showMessage(message)

}

}

// BaseFragment.kt

package com.lidaamber.adminpanel.ui.base

import android.content.BroadcastReceiver

import android.content.Context

import android.content.Intent

import android.content.IntentFilter

import android.net.ConnectivityManager

import android.support.v4.app.Fragment

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.presenters.BasePresenter

import com.lidaamber.adminpanel.ui.extensions.showMessage

import com.lidaamber.adminpanel.views.BaseView

/\*\*

\* BaseView implementation for Fragment

\* @author lidaamber

\*/

open class BaseFragment<T : BasePresenter> : Fragment(), BaseView {

/\*\*

\* Presenter

\*/

protected lateinit var presenter: T

/\*\*

\* Network events receiver

\*/

private lateinit var networkReceiver: BroadcastReceiver

override fun onStart() {

super.onStart()

networkReceiver = object : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

presenter.onNetworkStateChanged()

}

}

activity!!.registerReceiver(networkReceiver,

IntentFilter(ConnectivityManager.CONNECTIVITY\_ACTION))

}

override fun showNotConnectedMessage() {

context?.let {

view?.showMessage(it.getString(R.string.not\_connected))

}

}

override fun onResume() {

super.onResume()

presenter.onResume()

}

override fun onStop() {

super.onStop()

activity!!.unregisterReceiver(networkReceiver)

}

override fun displayMessage(message: String) {

view?.showMessage(message)

}

}

// InputDialogFragment.kt

package com.lidaamber.adminpanel.ui.dialogs

import android.app.AlertDialog

import android.app.Dialog

import android.os.Bundle

import android.support.annotation.StringRes

import android.support.design.widget.TextInputEditText

import android.support.v4.app.DialogFragment

import com.lidaamber.adminpanel.R

/\*\*

\* Dialog fragment for simple text input and report

\* @author lidaamber

\*/

class InputDialogFragment : DialogFragment() {

/\*\*

\* Handler for text input event

\*/

var onInputSubmit: ((String) -> Unit)? = null

/\*\*

\* Title resource identifier

\*/

@StringRes var titleId: Int = 0

/\*\*

\* Hint resource identifier

\*/

@StringRes var hintId: Int = 0

override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

val builder = AlertDialog.Builder(activity)

val v = activity?.layoutInflater?.inflate(R.layout.fragment\_input\_dialog, null)

val editText = v?.findViewById<TextInputEditText>(R.id.inputEditText)

editText?.hint = getString(hintId)

builder.setTitle(titleId)

builder.setView(v)

builder.setNegativeButton(R.string.decline) { \_, \_ ->

dismiss()

}

builder.setPositiveButton(R.string.add) { \_, \_ ->

dismiss()

onInputSubmit?.invoke(editText?.text.toString())

}

return builder.create()

}

}

// ListDialog.kt

package com.lidaamber.adminpanel.ui.dialogs

import android.app.AlertDialog

import android.app.Dialog

import android.os.Bundle

import android.support.v4.app.DialogFragment

import android.support.v7.widget.LinearLayoutManager

import android.support.v7.widget.RecyclerView

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.ui.adapters.InfoAdapter

/\*\*

\* @author lidaamber

\*/

class ListDialog : DialogFragment() {

var onChooseInfo: ((String) -> Unit)? = null

var infoList: List<String>? = null

override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

val v = activity!!.layoutInflater.inflate(R.layout.dialog\_list, null)

val list = v.findViewById<RecyclerView>(R.id.infoList)

list.layoutManager = LinearLayoutManager(activity)

list.adapter = InfoAdapter(infoList!!) {

onChooseInfo?.invoke(it)

dismiss()

}

val builder = AlertDialog.Builder(activity)

builder.setView(v)

return builder.create()

}

}

// Extensions.kt

package com.lidaamber.adminpanel.ui.extensions

import android.content.Context

import android.net.Uri

import android.support.design.widget.Snackbar

import android.support.design.widget.TextInputEditText

import android.text.Editable

import android.text.TextWatcher

import android.view.View

import java.io.BufferedInputStream

import java.lang.StringBuilder

/\*\*

\* Displays message

\* @author lidaamber

\*/

fun View.showMessage(string: String) {

Snackbar.make(this, string, Snackbar.LENGTH\_LONG).show()

}

/\*\*

\* Gets file content from URI

\* @author lidaamber

\*/

fun Uri.getFileContent(context: Context): String {

val fileStream = BufferedInputStream(context.contentResolver.openInputStream(this))

val reader = fileStream.bufferedReader()

var line: String? = reader.readLine()

val builder = StringBuilder()

while (line != null) {

builder.append(line)

builder.append('\n')

line = reader.readLine()

}

reader.close()

fileStream.close()

return builder.toString()

}

/\*\*

\* Adds simple text changed watcher

\*/

fun TextInputEditText.addTextWatcher(onTextChanged: (CharSequence?) -> Unit) {

this.addTextChangedListener(object : TextWatcher {

override fun afterTextChanged(s: Editable?) {

}

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

onTextChanged(s)

}

})

}

// LoginActivity.kt

package com.lidaamber.adminpanel.ui.login

import android.content.Intent

import android.os.Bundle

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.presenters.LoginPresenter

import com.lidaamber.adminpanel.presenters.LoginPresenterImpl

import com.lidaamber.adminpanel.ui.base.BaseActivity

import com.lidaamber.adminpanel.ui.main.MainActivity

import com.lidaamber.adminpanel.views.LoginView

import kotlinx.android.synthetic.main.activity\_login.\*

/\*\*

\* LoginView implementation

\* @author lidaamber

\*/

class LoginActivity : BaseActivity<LoginPresenter>(), LoginView {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_login)

presenter = LoginPresenterImpl(applicationContext, this)

presenter.init()

signInButton.setOnClickListener {

presenter.onSignInClicked(emailInputEditText.text.toString(),

passwordInputEditText.text.toString())

}

}

override fun showMainScreen() {

val intent = Intent(this, MainActivity::class.java)

startActivity(intent)

}

override fun setEmailErrorVisible(visible: Boolean) {

if (visible) {

emailInputLayout.isErrorEnabled = true

emailInputLayout.error = getString(R.string.data\_error)

} else emailInputLayout.isErrorEnabled = false

}

override fun setPasswordErrorVisible(visible: Boolean) {

if (visible) {

passwordInputLayout.isErrorEnabled = true

passwordInputLayout.error = getString(R.string.data\_error)

} else passwordInputLayout.isErrorEnabled = false

}

}

// EditQuestionActivity.kt

package com.lidaamber.adminpanel.ui.main

import android.os.Bundle

import android.support.v4.content.res.ResourcesCompat

import android.support.v7.widget.LinearLayoutManager

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Question

import com.lidaamber.adminpanel.models.QuestionType

import com.lidaamber.adminpanel.presenters.EditQuestionPresenter

import com.lidaamber.adminpanel.presenters.EditQuestionPresenterImpl

import com.lidaamber.adminpanel.ui.adapters.AnswersAdapter

import com.lidaamber.adminpanel.ui.base.BaseActivity

import com.lidaamber.adminpanel.ui.dialogs.InputDialogFragment

import com.lidaamber.adminpanel.ui.extensions.addTextWatcher

import com.lidaamber.adminpanel.views.EditQuestionView

import kotlinx.android.synthetic.main.activity\_edit\_question.\*

/\*\*

\* EditQuestionView implementation

\* @author lidaamber

\*/

class EditQuestionActivity : BaseActivity<EditQuestionPresenter>(), EditQuestionView {

companion object {

/\*\*

\* Tag for adding answer dialog fragment

\*/

const val ADD\_ANSWER = "add\_answer"

/\*\*

\* Extra keys constants

\*/

const val KEY\_MODE = "key\_mode"

const val KEY\_QUESTION = "key\_question"

const val KEY\_SURVEY\_ID = "key\_survey\_id"

const val KEY\_INDEX = "key\_index"

/\*\*

\* Edit question modes

\*/

const val MODE\_NEW = 1

const val MODE\_EDIT = 0

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_edit\_question)

val mode = intent.getIntExtra(KEY\_MODE, 0)

radioGroup.setOnCheckedChangeListener { \_, checkedId ->

when (checkedId) {

R.id.textInput -> presenter.onTextInputQuestionClicked()

R.id.numberInput -> presenter.onNumberInputQuestionClicked()

R.id.oneFromMany -> presenter.onChooseOneQuestionClicked()

R.id.manyFromMany -> presenter.onChooseManyQuestionClicked()

}

}

toolbar.setNavigationOnClickListener { onBackPressed() }

confirmQuestionButton.setOnClickListener {

presenter.onConfirmQuestionClicked(questionTitleEditText.text.toString())

}

addAnswerButton.setOnClickListener {

presenter.onAddAnswerClicked()

}

answersRecyclerView.layoutManager = LinearLayoutManager(this)

presenter = EditQuestionPresenterImpl(applicationContext, this)

presenter.init()

presenter.setMode(intent.getIntExtra(KEY\_MODE, 0) == MODE\_NEW)

presenter.setSurveyId(intent.getIntExtra(KEY\_SURVEY\_ID, 0))

if (mode == MODE\_NEW) {

confirmQuestionButton.setImageDrawable(

ResourcesCompat.getDrawable(resources, R.drawable.ic\_add\_survey, null))

toolbar.setTitle(R.string.create\_question)

} else {

val question = intent.getSerializableExtra(KEY\_QUESTION) as Question

val index = intent.getIntExtra(KEY\_INDEX, 0)

presenter.setSurveyQuestion(question, index)

}

}

override fun setTitle(title: String) {

questionTitleEditText.setText(title)

}

override fun setInputQuestionOptionsEnabled(enabled: Boolean) {

val color = if (enabled) {

ResourcesCompat.getColor(resources, android.R.color.black, null)

} else {

ResourcesCompat.getColor(resources, android.R.color.darker\_gray, null)

}

rangesTitle.setTextColor(color)

minRangeInputLayout.isEnabled = enabled

maxRangeInputLayout.isEnabled = enabled

if (!enabled) {

minRangeEditText.setText(String())

maxRangeEditText.setText(String())

} else {

minRangeEditText.addTextWatcher {

if (it.isNullOrEmpty()) {

presenter.setMinRange(null)

} else {

presenter.setMinRange(it!!.toString().toDouble())

}

}

maxRangeEditText.addTextWatcher {

if (it.isNullOrEmpty()) {

presenter.setMaxRange(null)

} else {

presenter.setMaxRange(it!!.toString().toDouble())

}

}

}

}

override fun setChooseQuestionOptionsEnabled(enabled: Boolean) {

val color = if (enabled) {

ResourcesCompat.getColor(resources, android.R.color.black, null)

} else {

ResourcesCompat.getColor(resources, android.R.color.darker\_gray, null)

}

answersRecyclerView.isEnabled = enabled

answersTitle.setTextColor(color)

addAnswerButton.isEnabled = enabled

val textColor = if (enabled) {

ResourcesCompat.getColor(resources, R.color.colorAccent, null)

} else {

ResourcesCompat.getColor(resources, android.R.color.darker\_gray, null)

}

addAnswerButton.setTextColor(textColor)

}

override fun close() {

onBackPressed()

}

override fun openAddAnswerDialog() {

val dialog = InputDialogFragment()

dialog.titleId = R.string.add\_answer

dialog.hintId = R.string.answer

dialog.onInputSubmit = {

presenter.onCreateAnswerConfirmed(it)

}

dialog.show(supportFragmentManager, ADD\_ANSWER)

}

override fun displayAnswers(answers: List<String>) {

answersRecyclerView.adapter = AnswersAdapter(answers, {

presenter.onDeleteAnswerClicked(it)

})

}

override fun updateAnswersList() {

answersRecyclerView.adapter.notifyDataSetChanged()

}

override fun setMinRange(min: Double) {

minRangeEditText.setText(min.toString())

}

override fun setMaxRange(max: Double) {

maxRangeEditText.setText(max.toString())

}

override fun setQuestionType(questionType: QuestionType) {

radioGroup.check(when (questionType) {

QuestionType.SELECT\_ONE -> R.id.oneFromMany

QuestionType.NUMBER\_INPUT -> R.id.numberInput

QuestionType.SELECT\_MANY -> R.id.manyFromMany

QuestionType.TEXT\_INPUT -> R.id.textInput

})

}

}

// EditSurveyActivity.kt

package com.lidaamber.adminpanel.ui.main

import android.content.Intent

import android.os.Bundle

import android.support.v4.content.res.ResourcesCompat

import android.support.v7.widget.LinearLayoutManager

import android.support.v7.widget.PopupMenu

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Question

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.presenters.EditSurveyPresenter

import com.lidaamber.adminpanel.presenters.EditSurveyPresenterImpl

import com.lidaamber.adminpanel.ui.adapters.AnswersAdapter

import com.lidaamber.adminpanel.ui.adapters.QuestionsAdapter

import com.lidaamber.adminpanel.ui.base.BaseActivity

import com.lidaamber.adminpanel.ui.dialogs.ListDialog

import com.lidaamber.adminpanel.views.EditSurveyView

import kotlinx.android.synthetic.main.activity\_edit\_survey.\*

/\*\*

\* EditSurveyView implementation

\* @author lidaamber

\*/

class EditSurveyActivity : BaseActivity<EditSurveyPresenter>(), EditSurveyView {

companion object {

/\*\*

\* Dialog tag

\*/

const val DIALOG\_GROUP = "group"

/\*\*

\* Extra constants

\*/

const val KEY\_MODE = "key\_mode"

const val KEY\_SURVEY = "key\_survey"

/\*\*

\* Edit survey modes

\*/

const val MODE\_NEW = 1

const val MODE\_EDIT = 0

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_edit\_survey)

val mode = intent.getIntExtra(KEY\_MODE, 0)

presenter = EditSurveyPresenterImpl(applicationContext, this)

presenter.init()

if (mode == MODE\_NEW) {

toolbar.setTitle(R.string.create\_survey)

confirmSurveyButton.setImageDrawable(

ResourcesCompat.getDrawable(resources, R.drawable.ic\_add\_survey, null))

}

toolbar.setNavigationOnClickListener { onBackPressed() }

confirmSurveyButton.setOnClickListener {

presenter.onConfirmButtonClicked(surveyTitleEditText.text.toString())

}

addQuestionButton.setOnClickListener {

presenter.onAddQuestionClicked()

}

addGroupButton.setOnClickListener {

presenter.onAddGroupClicked()

}

groupsRecyclerView.layoutManager = LinearLayoutManager(this)

questionsRecyclerView.layoutManager = LinearLayoutManager(this)

}

override fun onResume() {

super.onResume()

val mode = intent.getIntExtra(KEY\_MODE, 0)

val survey = intent.getSerializableExtra(KEY\_SURVEY) as Survey

presenter.setSurveyMode(survey, mode == MODE\_NEW)

}

override fun displaySurveyQuestions(questions: List<Question>) {

questionsRecyclerView.adapter = QuestionsAdapter(applicationContext, questions, { question, v ->

val menu = PopupMenu(this, v)

menu.menu.add(0, 0, 0, R.string.edit\_question)

menu.menu.add(0, 1, 0, R.string.delete\_question)

menu.setOnMenuItemClickListener {

if (it.itemId == 0) {

presenter.onEditQuestionClicked(question)

} else {

presenter.onRemoveQuestionClicked(question)

}

return@setOnMenuItemClickListener true

}

menu.show()

})

}

override fun displayTitle(title: String) {

surveyTitleEditText.setText(title)

}

override fun showNewQuestionScreen(surveyId: Int) {

val intent = Intent(this, EditQuestionActivity::class.java)

intent.putExtra(EditQuestionActivity.KEY\_MODE, EditQuestionActivity.MODE\_NEW)

intent.putExtra(EditQuestionActivity.KEY\_SURVEY\_ID, surveyId)

startActivity(intent)

}

override fun showEditQuestionScreen(surveyId: Int, question: Question, index: Int) {

val intent = Intent(this, EditQuestionActivity::class.java)

intent.putExtra(EditQuestionActivity.KEY\_MODE, EditQuestionActivity.MODE\_EDIT)

intent.putExtra(EditQuestionActivity.KEY\_QUESTION, question)

intent.putExtra(EditQuestionActivity.KEY\_INDEX, index)

intent.putExtra(EditQuestionActivity.KEY\_SURVEY\_ID, surveyId)

startActivity(intent)

}

override fun updateSurveyQuestions() {

questionsRecyclerView?.adapter?.notifyDataSetChanged()

}

override fun displayGroups(list: List<String>) {

groupsRecyclerView?.adapter = AnswersAdapter(list, { group ->

presenter.onGroupDeleteClicked(group)

})

}

override fun updateGroupsList() {

groupsRecyclerView?.adapter?.notifyDataSetChanged()

}

override fun close() {

onBackPressed()

}

override fun openGroupsList(groups: List<String>) {

val dialog = ListDialog()

dialog.infoList = groups

dialog.onChooseInfo = {

presenter.onChooseGroupConfirmed(it)

}

dialog.show(supportFragmentManager, DIALOG\_GROUP)

}

}

// ImportUsersFragment.kt

package com.lidaamber.adminpanel.ui.main

import android.content.Intent

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.presenters.ImportUsersPresenter

import com.lidaamber.adminpanel.presenters.ImportUsersPresenterImpl

import com.lidaamber.adminpanel.ui.base.BaseFragment

import com.lidaamber.adminpanel.views.ImportUsersView

import kotlinx.android.synthetic.main.fragment\_import\_users.\*

/\*\*

\* ImportUsersView implementation

\* @author lidaamber

\*/

class ImportUsersFragment : BaseFragment<ImportUsersPresenter>(), ImportUsersView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_import\_users, container, false)

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

presenter = ImportUsersPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

importUsersButton.setOnClickListener {

presenter.onImportClicked()

}

}

override fun openFileChooser(requestCode: Int) {

val intent = Intent(Intent.ACTION\_GET\_CONTENT)

intent.type = "text/csv"

startActivityForResult(intent, requestCode)

}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)

presenter.processActivityResult(requestCode, resultCode, data)

}

}

// MainActivity.kt

package com.lidaamber.adminpanel.ui.main

import android.os.Bundle

import android.support.design.widget.NavigationView

import android.support.v4.view.GravityCompat

import android.support.v7.app.ActionBarDrawerToggle

import android.support.v7.app.AppCompatActivity

import android.view.MenuItem

import com.lidaamber.adminpanel.R

import kotlinx.android.synthetic.main.activity\_main.\*

import kotlinx.android.synthetic.main.content\_main.\*

/\*\*

\* Activity for hosting main application views

\* @author lidaamber

\*/

class MainActivity : AppCompatActivity(), NavigationView.OnNavigationItemSelectedListener {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

setSupportActionBar(toolbar)

val toggle = ActionBarDrawerToggle(

this, drawerLayout, toolbar, R.string.drawer\_open, R.string.drawer\_close)

drawerLayout.addDrawerListener(toggle)

toggle.syncState()

if (supportFragmentManager.findFragmentById(R.id.fragmentContainer) == null)

supportFragmentManager

.beginTransaction()

.add(R.id.fragmentContainer, SurveyListFragment())

.commit()

navigationView.setNavigationItemSelectedListener(this)

}

override fun onBackPressed() {

if (drawerLayout.isDrawerOpen(GravityCompat.START)) {

drawerLayout.closeDrawer(GravityCompat.START)

} else {

super.onBackPressed()

}

}

override fun onNavigationItemSelected(item: MenuItem): Boolean {

val fragment = when (item.itemId) {

R.id.import\_users -> ImportUsersFragment()

else -> SurveyListFragment()

}

supportFragmentManager.beginTransaction()

.replace(R.id.fragmentContainer, fragment)

.commit()

drawerLayout.closeDrawer(GravityCompat.START)

return true

}

}

// ResultsActivity.kt

package com.lidaamber.adminpanel.ui.main

import android.os.Bundle

import android.support.v7.widget.LinearLayoutManager

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Result

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.presenters.ResultsPresenter

import com.lidaamber.adminpanel.presenters.ResultsPresenterImpl

import com.lidaamber.adminpanel.ui.adapters.ResultsAdapter

import com.lidaamber.adminpanel.ui.base.BaseActivity

import com.lidaamber.adminpanel.views.ResultsView

import kotlinx.android.synthetic.main.activity\_results.\*

/\*\*

\* ResultsView implementation

\* @author lidaamber

\*/

class ResultsActivity : BaseActivity<ResultsPresenter>(), ResultsView {

companion object {

/\*\*

\* Extra constants

\*/

const val KEY\_SURVEY = "key\_survey"

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_results)

toolbar.setNavigationOnClickListener { onBackPressed() }

resultsRecyclerView.layoutManager = LinearLayoutManager(this)

presenter = ResultsPresenterImpl(applicationContext, this)

presenter.init()

presenter.setSurvey(intent.getSerializableExtra(KEY\_SURVEY) as Survey)

}

override fun displayResults(results: Map<String, List<Result>>) {

resultsRecyclerView.adapter = ResultsAdapter(results)

}

}

// SurveyListFragment.kt

package com.lidaamber.adminpanel.ui.main

import android.content.Intent

import android.os.Bundle

import android.support.v7.widget.LinearLayoutManager

import android.support.v7.widget.PopupMenu

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.adminpanel.R

import com.lidaamber.adminpanel.models.Survey

import com.lidaamber.adminpanel.presenters.SurveyListPresenter

import com.lidaamber.adminpanel.presenters.SurveyListPresenterImpl

import com.lidaamber.adminpanel.ui.adapters.SurveysAdapter

import com.lidaamber.adminpanel.ui.base.BaseFragment

import com.lidaamber.adminpanel.ui.dialogs.InputDialogFragment

import com.lidaamber.adminpanel.views.SurveyListView

import kotlinx.android.synthetic.main.fragment\_surveys\_list.\*

/\*\*

\* SurveyListView implementation

\* @author lidaamber

\*/

class SurveyListFragment : BaseFragment<SurveyListPresenter>(), SurveyListView {

companion object {

/\*\*

\* Tag for adding survey dialog fragment

\*/

const val ADD\_SURVEY = "add\_survey"

}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_surveys\_list, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

addSurveyButton.setOnClickListener {

presenter.onAddSurveyClicked()

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

surveysRecyclerView.layoutManager = LinearLayoutManager(activity)

presenter = SurveyListPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun displaySurveys(surveys: List<Survey>) {

surveysRecyclerView?.adapter = SurveysAdapter(surveys, { survey, v ->

activity?.let {

val menu = PopupMenu(it, v)

menu.menu.add(0, 0, 0, R.string.edit\_survey)

menu.menu.add(0, 1, 0, R.string.show\_results)

menu.menu.add(0, 2, 0, R.string.delete\_survey)

menu.setOnMenuItemClickListener {

when (it.itemId) {

0 -> presenter.onEditSurveyClicked(survey)

1 -> presenter.onSurveyResultsClicked(survey)

2 -> presenter.onDeleteSurveyClicked(survey)

}

return@setOnMenuItemClickListener true

}

menu.show()

}

})

}

override fun showResultsScreen(survey: Survey) {

val intent = Intent(activity, ResultsActivity::class.java)

intent.putExtra(ResultsActivity.KEY\_SURVEY, survey)

startActivity(intent)

}

override fun updateSurveysList() {

surveysRecyclerView?.adapter?.notifyDataSetChanged()

}

override fun showEditSurveyScreen(survey: Survey) {

val intent = Intent(activity, EditSurveyActivity::class.java)

intent.putExtra(EditSurveyActivity.KEY\_SURVEY, survey)

intent.putExtra(EditSurveyActivity.KEY\_MODE, EditSurveyActivity.MODE\_EDIT)

startActivity(intent)

}

override fun showNewSurveyScreen(survey: Survey) {

val intent = Intent(activity, EditSurveyActivity::class.java)

intent.putExtra(EditSurveyActivity.KEY\_SURVEY, survey)

intent.putExtra(EditSurveyActivity.KEY\_MODE, EditSurveyActivity.MODE\_NEW)

startActivity(intent)

}

override fun showAddSurveyDialog() {

val dialog = InputDialogFragment()

dialog.onInputSubmit = { presenter.onSurveyCreationConfirmed(it) }

dialog.hintId = R.string.survey\_title

dialog.titleId = R.string.add\_survey

dialog.show(activity?.supportFragmentManager, ADD\_SURVEY)

}

}

// BaseView.kt

package com.lidaamber.adminpanel.views

/\*\*

\* Base application view

\* @author lidaamber

\*/

interface BaseView : NetworkAvailabilityView {

/\*\*

\* Displays informing message

\*/

fun displayMessage(message: String)

}

// EditQuestionView.kt

package com.lidaamber.adminpanel.views

import com.lidaamber.adminpanel.models.QuestionType

/\*\*

\* View for creating or editing question functionality

\* @author lidaamber

\*/

interface EditQuestionView : BaseView {

/\*\*

\* Sets question title

\*/

fun setTitle(title: String)

/\*\*

\* Sets input question options enabled status

\*/

fun setInputQuestionOptionsEnabled(enabled: Boolean)

/\*\*

\* Sets choose question options enabled status

\*/

fun setChooseQuestionOptionsEnabled(enabled: Boolean)

/\*\*

\* Closes view

\*/

fun close()

/\*\*

\* Opens dialog for adding new answer

\*/

fun openAddAnswerDialog()

/\*\*

\* Sets up answers list

\*/

fun displayAnswers(answers: List<String>)

/\*\*

\* Updates answers list

\*/

fun updateAnswersList()

/\*\*

\* Sets minimum range

\*/

fun setMinRange(min: Double)

/\*\*

\* Sets maximum range

\*/

fun setMaxRange(max: Double)

/\*\*

\* Sets question type

\*/

fun setQuestionType(questionType: QuestionType)

}

// EditSurveyView.kt

package com.lidaamber.adminpanel.views

import com.lidaamber.adminpanel.models.Question

/\*\*

\* View for creating or editing survey functionality

\* @author lidaamber

\*/

interface EditSurveyView : BaseView {

/\*\*

\* Displays survey questions list

\*/

fun displaySurveyQuestions(questions: List<Question>)

/\*\*

\* Displays survey title

\*/

fun displayTitle(title: String)

/\*\*

\* Shows new question screen

\*/

fun showNewQuestionScreen(surveyId: Int)

/\*\*

\* Shows edit question screen

\*/

fun showEditQuestionScreen(surveyId: Int, question: Question, index: Int)

/\*\*

\* Updates survey questions list

\*/

fun updateSurveyQuestions()

/\*\*

\* Displays list of groups

\*/

fun displayGroups(list: List<String>)

/\*\*

\* Update list of groups

\*/

fun updateGroupsList()

/\*\*

\* Closes the view

\*/

fun close()

/\*\*

\* Open group list to choose from

\*/

fun openGroupsList(groups: List<String>)

}

// ImportUsersView.kt

package com.lidaamber.adminpanel.views

/\*\*

\* View for importing users functionality

\* @author lidaamber

\*/

interface ImportUsersView : BaseView {

/\*\*

\* Opens file chooser

\*/

fun openFileChooser(requestCode: Int)

}

// LoginView.kt

package com.lidaamber.adminpanel.views

/\*\*

\* View for login functionality

\* @author lidaamber

\*/

interface LoginView : BaseView {

/\*\*

\* Shows main screen

\*/

fun showMainScreen()

/\*\*

\* Sets email error visibility status

\*/

fun setEmailErrorVisible(visible: Boolean)

/\*\*

\* Sets password error visibility status

\*/

fun setPasswordErrorVisible(visible: Boolean)

}

// NetworkAvailabilityView.kt

package com.lidaamber.adminpanel.views

/\*\*

\* View for tracking network availability

\* @author lidaamber

\*/

interface NetworkAvailabilityView {

/\*\*

\* Shows disconnected informational message

\*/

fun showNotConnectedMessage()

}

// ResultsView.kt

package com.lidaamber.adminpanel.views

import com.lidaamber.adminpanel.models.Result

/\*\*

\* View for survey results functionality

\* @author lidaamber

\*/

interface ResultsView : BaseView {

/\*\*

\* Shows survey results list

\*/

fun displayResults(results: Map<String, List<Result>>)

}

// SurveyListView.kt

package com.lidaamber.adminpanel.views

import com.lidaamber.adminpanel.models.Survey

/\*\*

\* View for survey list functionality

\* @author lidaamber

\*/

interface SurveyListView : BaseView {

/\*\*

\* Displays surveys list

\*/

fun displaySurveys(surveys: List<Survey>)

/\*\*

\* Updates surveys list

\*/

fun updateSurveysList()

/\*\*

\* Shows edit survey screen

\*/

fun showEditSurveyScreen(survey: Survey)

/\*\*

\* Shows survey results screen

\*/

fun showResultsScreen(survey: Survey)

/\*\*

\* Shows new survey screen

\*/

fun showNewSurveyScreen(survey: Survey)

/\*\*

\* Shows add survey dialog

\*/

fun showAddSurveyDialog()

}

**Опис програми клієнтського застосування**

// AuthenticationProvider.kt

package com.lidaamber.kpisurvey.model

import android.os.Handler

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.network.JSTPConverter

import com.lidaamber.kpisurvey.network.NetworkManager

import com.metarhia.jstp.handlers.OkErrorHandler

/\*\*

\* Provider of methods for authentication flow

\* @author lidaamber

\*/

object AuthenticationProvider {

/\*\*

\* Android Handler used for handling events on main thread after network responses

\*/

private val handler = Handler()

/\*\*

\* Error codes and descriptions for auth interface

\*/

private val authErrors by lazy {

mapOf(

1025 to R.string.invalid\_credentials,

1026 to R.string.must\_be\_authenticated,

1027 to R.string.invalid\_token,

1028 to R.string.already\_registered,

1029 to R.string.email\_in\_use

)

}

/\*\*

\* Authenticates user

\*

\* @param answeredQuestions authentication questions with answers

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun authenticate(answeredQuestions: List<AuthenticationQuestion>, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

val method = NetworkManager.AUTHENTICATE

val arguments = listOf(JSTPConverter.convertToCredentials(answeredQuestions))

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method,

arguments, object : OkErrorHandler() {

override fun handleError(errorCode: Int?, arguments: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(arguments: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Registers user's personal data

\*

\* @param email user's email

\* @param password user's password

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun register(email: String, password: String, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

val method = NetworkManager.REGISTER

val arguments = JSTPConverter.createRegistrationData(email, password)

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method, arguments,

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Confirms user's data with token

\*

\* @param token confirmation token

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun confirmToken(token: String, callback: () -> Unit, errorCallback: (Int) -> Unit) {

val method = NetworkManager.CONFIRM\_EMAIL

val arguments = listOf(token)

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method, arguments,

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Signs in by answers on authentication questions

\*

\* @param answeredQuestions authentication questions with answers

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun login(answeredQuestions: List<AuthenticationQuestion>, password: String,

callback: () -> Unit, errorCallback: (Int) -> Unit) {

val method = NetworkManager.LOGIN

val arguments = JSTPConverter.createLoginData(answeredQuestions, password)

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method, arguments,

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Restores users password

\*

\* @param email email bound to user's account

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun restorePassword(email: String, callback: () -> Unit, errorCallback: (Int) -> Unit) {

val method = NetworkManager.RESTORE\_PASSWORD

val arguments = listOf(email)

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method, arguments,

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

/\*\*

\* Updates user password to new one

\*

\* @param token confirmation token

\* @param password new password

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun updatePassword(token: String, password: String, callback: () -> Unit,

errorCallback: (Int) -> Unit) {

val method = NetworkManager.UPDATE\_PASSWORD

val arguments = listOf(token, password)

NetworkManager.call(NetworkManager.AUTH\_INTERFACE, method, arguments,

object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !authErrors.containsKey(errorCode)) return

handler.post { errorCallback(authErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

}

// ConfigProvider.kt

package com.lidaamber.kpisurvey.model

import android.content.Context

import android.os.Handler

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.network.JSTPConverter

import com.lidaamber.kpisurvey.network.NetworkManager

import com.metarhia.jstp.handlers.OkErrorHandler

import java.io.File

import java.io.ObjectInputStream

import java.io.ObjectOutputStream

/\*\*

\* Manager for configuration

\* @author lidaamber

\*/

object ConfigProvider {

/\*\*

\* Config filename

\*/

private const val CONFIG = "config"

/\*\*

\* Android Handler used for handling events on main thread after network responses

\*/

private val handler = Handler()

/\*\*

\* Updates configuration

\*

\* @param context application context

\* @param callback callback for successful network response

\*/

private fun updateConfig(context: Context, callback: ((Config) -> Unit)? = null) {

val method = NetworkManager.GET

NetworkManager.call(NetworkManager.CONFIG\_INTERFACE, method,

arrayListOf(LocaleProvider.getLocale()), object : OkErrorHandler() {

override fun handleError(errorCode: Int?, arguments: MutableList<\*>?) {

}

override fun handleOk(arguments: MutableList<\*>?) {

arguments?.let {

val config = JSTPConverter.getConfig(it[0]!!)

updatePreferences(context, config)

callback?.invoke(config)

}

}

})

}

/\*\*

\* Updates user preferences

\*/

private fun updatePreferences(context: Context, config: Config) {

val configFile = File(context.filesDir, CONFIG)

if (!configFile.exists()) {

configFile.createNewFile()

}

val os = ObjectOutputStream(configFile.outputStream())

os.writeObject(config)

os.flush()

os.close()

}

/\*\*

\* Gets config from server or local storage

\*

\* @param context application context

\* @param callback callback for successful network response

\*/

private fun getConfig(context: Context, callback: (Config) -> Unit) {

try {

val configFile = File(context.filesDir, CONFIG)

if (!configFile.exists()) updateConfig(context) {

handler.post { callback(it) }

} else {

val inputStream = ObjectInputStream(configFile.inputStream())

val config = inputStream.readObject() as Config

inputStream.close()

callback(config)

}

} catch (e: Exception) {

e.printStackTrace()

}

}

/\*\*

\* Gets credentials questions

\*

\* @param context application context

\* @param callback callback for successful network response

\*/

fun getCredentialsQuestions(context: Context, callback: (List<AuthenticationQuestion>) -> Unit) {

ConfigProvider.getConfig(context) { config ->

val authQuestions = ArrayList(config.questions)

val resultQuestions = ArrayList<AuthenticationQuestion>()

(0 until config.authLimit).forEach {

val index = Math.floor(Math.random() \* (config.questions.size - it)).toInt()

val question = authQuestions[index]

resultQuestions.add(question)

authQuestions.remove(question)

}

callback(resultQuestions)

}

}

/\*\*

\* Gets login credentials questions

\*

\* @param context application context

\* @param callback callback for successful network response

\*/

fun getLoginCredentialsQuestions(context: Context, callback: (List<AuthenticationQuestion>) -> Unit) {

callback(arrayListOf(

AuthenticationQuestion(

context.getString(R.string.email),

AuthenticationQuestion.TEXT,

"email")

))

}

}

//Data models.kt

package com.lidaamber.kpisurvey.model

import java.io.Serializable

import java.util.\*

/\*\*

\* AuthenticationQuestion model

\* @author lidaamber

\*/

data class AuthenticationQuestion(val title: String,

val type: Int,

val id: String,

var answer: Any? = null) : Serializable {

companion object {

const val TEXT = 0

const val NUMBER = 1

const val DATE = 2

}

}

/\*\*

\* Question model

\* @author lidaamber

\*/

data class Question(val title: String,

val type: Int,

val answers: MutableList<String>? = null,

val min: Double? = null,

val max: Double? = null,

val info: Map<String, String>? = null,

var answer: Any? = null) {

companion object {

const val TEXT\_QUESTION = 0

const val SELECT\_ONE\_QUESTION = 1

const val SELECT\_MANY\_QUESTION = 2

const val NUMBER\_QUESTION = 3

}

}

/\*\*

\* Survey identifier

\* @author lidaamber

\*/

typealias SurveyId = Int

/\*\*

\* Survey model

\* @author lidaamber

\*/

data class Survey(val id: SurveyId,

val title: String,

val created: Date,

var passed: Boolean)

/\*\*

\* Config model

\* @author lidaamber

\*/

data class Config(val questions: List<AuthenticationQuestion>,

val authLimit: Int,

val modeDark: Boolean) : Serializable

// LocaleProvider.kt

package com.lidaamber.kpisurvey.model

import com.lidaamber.kpisurvey.BuildConfig

/\*\*

\* Gets application locale for configuration

\* @author lidaamber

\*/

object LocaleProvider {

fun getLocale() = BuildConfig.LOCALE

}

// SurveysProvider.kt

package com.lidaamber.kpisurvey.model

import android.os.Handler

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.network.JSTPConverter

import com.lidaamber.kpisurvey.network.NetworkManager

import com.metarhia.jstp.handlers.OkErrorHandler

/\*\*

\* Provider of methods for survey flow

\* @author lidaamber

\*/

object SurveysProvider {

/\*\*

\* Android Handler used for handling events on main thread after network responses

\*/

val handler = Handler()

/\*\*

\* Error codes and descriptions for survey interface

\*/

val surveyErrors = mapOf(

1025 to R.string.must\_be\_logged\_in,

1026 to R.string.survey\_not\_found,

1027 to R.string.question\_not\_found,

1028 to R.string.invalid\_answer

)

/\*\*

\* Initializes connection

\*

\* @param initCallback callback after successful network layer initialization

\*/

fun init(initCallback: (() -> Unit)? = null) {

NetworkManager.init({ handler.post { initCallback?.invoke() } })

}

/\*\*

\* Gets list of available surveys

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun getSurveys(callback: (List<Survey>) -> Unit, errorCallback: (Int) -> Unit) {

val method = NetworkManager.GET\_SURVEYS

val args = arrayListOf<Any>()

NetworkManager.call(NetworkManager.SURVEY\_INTERFACE, method, args, object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !surveyErrors.containsKey(errorCode)) return

handler.post { errorCallback(surveyErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

val surveys = JSTPConverter.getSurveys(args)

handler.post { callback(surveys) }

}

})

}

/\*\*

\* Gets survey questions

\*

\* @param id survey identifier

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun getSurveyQuestions(id: SurveyId, callback: (List<Question>) -> Unit,

errorCallback: (Int) -> Unit) {

val method = NetworkManager.GET\_QUESTIONS

val args = arrayListOf(id)

NetworkManager.call(NetworkManager.SURVEY\_INTERFACE, method, args, object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !surveyErrors.containsKey(errorCode)) return

handler.post { errorCallback(surveyErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

val questions = JSTPConverter.getQuestions(args)

handler.post { callback(questions) }

}

})

}

/\*\*

\* Answers on survey question

\*

\* @param surveyId identifier

\* @param questionIndex index of question in survey

\* @param answer answer on the question

\* @param callback callback for successful network response

\* @param errorCallback callback for error network response

\*/

fun answer(surveyId: SurveyId, questionIndex: Int, answer: Any,

callback: () -> Unit, errorCallback: (Int) -> Unit) {

val method = NetworkManager.ANSWER

val args = arrayListOf(surveyId, questionIndex, answer)

NetworkManager.call(NetworkManager.SURVEY\_INTERFACE, method, args, object : OkErrorHandler() {

override fun handleError(errorCode: Int?, args: MutableList<\*>?) {

if (errorCode == null || !surveyErrors.containsKey(errorCode)) return

handler.post { errorCallback(surveyErrors[errorCode]!!) }

}

override fun handleOk(args: MutableList<\*>?) {

handler.post { callback() }

}

})

}

}

// JSTPConverter.kt

package com.lidaamber.kpisurvey.network

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.model.Config

import com.lidaamber.kpisurvey.model.Question

import com.lidaamber.kpisurvey.model.Survey

import java.text.SimpleDateFormat

import java.util.\*

import kotlin.collections.LinkedHashMap

/\*\*

\* Manager to convert JSTP arguments

\* @author lidaamber

\*/

object JSTPConverter {

/\*\*

\* Constants used to form JSTP messages' arguments

\*/

private const val ID = "id"

private const val TITLE = "title"

private const val CREATED = "created"

private const val COMPLETED = "completed"

private const val TYPE = "type"

private const val ANSWERS = "answers"

private const val INFO = "info"

private const val ANSWER = "submittedAnswer"

private const val ACCEPTABLE\_QUESTIONS = "acceptableQuestions"

private const val AUTH\_LIMIT = "requiredAmountOfQuestions"

private const val DARK\_MODE = "darkMode"

private const val LOCALIZATION = "localization"

private const val MAX = "max"

private const val MIN = "min"

/\*\*

\* Map of dependencies between question types and corresponding constants

\*/

private val types by lazy {

mapOf("chooseOne" to Question.SELECT\_ONE\_QUESTION,

"chooseMany" to Question.SELECT\_MANY\_QUESTION,

"number" to Question.NUMBER\_QUESTION,

"text" to Question.TEXT\_QUESTION)

}

/\*\*

\* Converts answered questions to user's credentials in appropriate format

\*

\* @param answeredQuestions authentication questions with user's answers

\*

\* @return formatted credentials

\*/

fun convertToCredentials(answeredQuestions: List<AuthenticationQuestion>): Map<String, Any> {

return LinkedHashMap<String, Any>(answeredQuestions.map {

val answer: Any = when (it.answer) {

is Number -> it.answer!!

is Date -> {

val format = SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss.sssZ", Locale.getDefault())

format.format(it.answer as Date)

}

else -> it.answer.toString()

}

return@map it.id to answer

}.toMap())

}

/\*\*

\* Creates registration data in appropriate format

\*

\* @param email user's email

\* @param password user's password

\*

\* @return registration data in appropriate format

\*/

fun createRegistrationData(email: String, password: String): List<String> {

return listOf(email, password)

}

/\*\*

\* Creates login data in appropriate format

\*

\* @param questions authentication questions with user's answers

\* @param password user's password

\*

\* @return login data in appropriate format

\*/

fun createLoginData(questions: List<AuthenticationQuestion>, password: String): List<Any> {

return listOf(convertToCredentials(questions), password)

}

/\*\*

\* Gets surveys information from formatted data

\*

\* @param args JSTP arguments

\*

\* @return surveys information

\*/

fun getSurveys(args: MutableList<\*>?): List<Survey> {

if (args == null) return arrayListOf()

val surveysList = args[0] as? List<\*>

return surveysList?.map {

val simpleDateFormat =

SimpleDateFormat("EEE MMM dd yyyy hh:mm:ss z '(UTC)'", Locale.ENGLISH)

val surveyData = it as? Map<String, Any>

return@map Survey(id = surveyData?.get(ID) as Int,

title = surveyData[TITLE] as String,

created = simpleDateFormat.parse(surveyData[CREATED] as String),

passed = surveyData[COMPLETED] as Boolean)

} ?: arrayListOf()

}

/\*\*

\* Gets questions information from formatted data

\*

\* @param args JSTP arguments

\*

\* @return questions information

\*/

fun getQuestions(args: MutableList<\*>?): List<Question> {

if (args == null) return arrayListOf()

val questionsList = args[0] as? List<\*>

return questionsList?.map {

val questionData = it as? Map<\*, \*>

return@map Question(title = questionData?.get(TITLE) as String,

type = types[questionData[TYPE] as String] as Int,

answers = questionData[ANSWERS] as? MutableList<String>?,

info = questionData[INFO] as? Map<String, String>?,

answer = questionData[ANSWER],

max = (questionData[MAX] as? Int?)?.toDouble(),

min = (questionData[MIN] as? Int)?.toDouble())

} ?: arrayListOf()

}

/\*\*

\* Gets config information from formatted data

\*

\* @param args JSTP arguments

\*

\* @return config information

\*/

fun getConfig(configObject: Any): Config {

val configMap = configObject as Map<String, Any>

val questions = configMap[ACCEPTABLE\_QUESTIONS] as Map<String, String>

val localizations = configMap[LOCALIZATION] as Map<String, String>

return Config(questions.map { authenticationQuestion ->

return@map AuthenticationQuestion(id = authenticationQuestion.key,

type = when (authenticationQuestion.value) {

"string" -> AuthenticationQuestion.TEXT

"number" -> AuthenticationQuestion.NUMBER

else -> AuthenticationQuestion.DATE

},

title = localizations[authenticationQuestion.key] as String)

}, configMap[AUTH\_LIMIT] as Int, configMap[DARK\_MODE] as Boolean)

}

}

// NetworkAvailabilityManager.kt

package com.lidaamber.kpisurvey.network

import android.content.Context

import android.net.ConnectivityManager

/\*\*

\* Manager for tracking network state and sending corresponding network events

\* @author lidaamber

\*/

class NetworkAvailabilityManager(context: Context) {

/\*\*

\* Connectivity manager

\*/

private val manager: ConnectivityManager = context.getSystemService(Context.CONNECTIVITY\_SERVICE) as ConnectivityManager

/\*\*

\* Network events listener

\*/

private var listener: NetworkListener? = null

/\*\*

\* Network state

\*/

private var currentNetworkState: Boolean = false

/\*\*

\* Network availability state

\*/

val isNetworkAvailable: Boolean

get() {

val activeNetwork = manager.activeNetworkInfo

return activeNetwork != null && activeNetwork.isConnectedOrConnecting

}

/\*\*

\* Network state change event handler

\*/

fun onNetworkStateChanged() {

val lastState = currentNetworkState

currentNetworkState = isNetworkAvailable

if (listener == null) return

if (currentNetworkState && lastState != currentNetworkState) {

listener!!.onConnected()

} else if (!currentNetworkState && lastState != currentNetworkState) {

listener!!.onDisconnected()

}

}

/\*\*

\* Sets network state on resume

\*/

fun onResume() {

currentNetworkState = isNetworkAvailable

}

/\*\*

\* Sets network availability listener

\*/

fun setNetworkAvailabilityListener(listener: NetworkListener) {

this.listener = listener

}

/\*\*

\* Network events listener

\*/

interface NetworkListener {

/\*\*

\* Handler for connection network event

\*/

fun onConnected()

/\*\*

\* Handler for disconnection network event

\*/

fun onDisconnected()

}

}

// NetworkManager.kt

package com.lidaamber.kpisurvey.network

import com.lidaamber.kpisurvey.BuildConfig

import com.metarhia.jstp.connection.Connection

import com.metarhia.jstp.connection.SimpleConnectionListener

import com.metarhia.jstp.handlers.OkErrorHandler

import com.metarhia.jstp.transport.TCPTransport

/\*\*

\* Manager to work with network requests

\* @author lidaamber

\*/

object NetworkManager : SimpleConnectionListener() {

/\*\*

\* Server settings

\*/

private const val HOST = BuildConfig.HOST

private const val PORT = BuildConfig.PORT

/\*\*

\* JSTP application name

\*/

private const val APPLICATION\_NAME = "survey"

/\*\*

\* Auth interface constants

\*/

const val AUTH\_INTERFACE = "auth"

const val AUTHENTICATE = "authenticate"

const val REGISTER = "register"

const val CONFIRM\_EMAIL = "confirmEmail"

const val LOGIN = "login"

const val RESTORE\_PASSWORD = "restorePassword"

const val UPDATE\_PASSWORD = "updatePassword"

/\*\*

\* Survey interface constants

\*/

const val SURVEY\_INTERFACE = "survey"

const val GET\_SURVEYS = "getSurveys"

const val GET\_QUESTIONS = "getQuestions"

const val ANSWER = "answer"

/\*\*

\* Config interface constants

\*/

const val CONFIG\_INTERFACE = "config"

const val GET = "get"

/\*\*

\* JSTP connection

\*/

private var connection: Connection? = null

/\*\*

\* Initialized connection

\*

\* @param callback callback after connection

\*/

fun init(callback: (() -> Unit)? = null) {

if (connection == null) {

connection = Connection(TCPTransport(HOST, PORT, true))

connection?.addListener(object : SimpleConnectionListener() {

override fun onConnected(connected: Boolean) {

connection?.removeListener(this)

callback?.invoke()

}

})

connection?.setReconnectCallback { \_, transportConnector ->

transportConnector.connect(null)

}

connection?.connect(APPLICATION\_NAME)

} else {

connection?.addListener(object : SimpleConnectionListener() {

override fun onConnected(connected: Boolean) {

connection?.removeListener(this)

callback?.invoke()

}

})

connection?.let {

if (it.isConnected) {

callback?.invoke()

return@let

}

it.addListener(object : SimpleConnectionListener() {

override fun onConnected(connected: Boolean) {

it.removeListener(this)

callback?.invoke()

}

})

it.connect(it.appData)

}

}

}

/\*\*

\* Makes JSTP call

\*

\* @param interfaceName interface name

\* @param method method name

\* @param arguments method arguments

\* @param okErrorHandler handler for successful and error results

\*/

fun call(interfaceName: String, method: String, arguments: List<Any>,

okErrorHandler: OkErrorHandler) {

connection?.callResendable(interfaceName, method, arguments, okErrorHandler)

}

}

// AuthenticationPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.model.ConfigProvider

import com.lidaamber.kpisurvey.views.AuthenticationView

/\*\*

\* Presenter for authentication flow

\* @author lidaamber

\*/

interface AuthenticationPresenter : BasePresenter {

/\*\*

\* Event handler for authenticate click

\*/

fun onAuthenticateClicked()

/\*\*

\* Event handler for sign in click

\*/

fun onSignInClicked()

/\*\*

\* Reports question state change to enable/disable authentication button

\*/

fun reportQuestionsStateChange()

}

/\*\*

\* AuthenticationPresenter implementation

\* @author lidaamber

\*/

class AuthenticationPresenterImpl(context: Context,

val view: AuthenticationView) :

BasePresenterImpl(context, view), AuthenticationPresenter {

/\*\*

\* Authentication questions

\*/

private lateinit var questions: List<AuthenticationQuestion>

override fun init() {

super.init()

ConfigProvider.getCredentialsQuestions(context) {

this.questions = it

view.showQuestions(questions)

}

}

override fun onAuthenticateClicked() {

if (!validateAnswers()) return

AuthenticationProvider.authenticate(questions, {

view.showRegistrationScreen()

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

private fun validateAnswers(): Boolean {

return questions.all { it.answer != null }

}

override fun onSignInClicked() {

view.showLoginScreen()

}

override fun reportQuestionsStateChange() {

if (questions.all { it.answer != null }) view.showAuthenticateButton() else view.hideAuthenticateButton()

}

}

// BasePresenter.kt

package com.lidaamber.kpisurvey.presenters

/\*\*

\* Base presenter for views

\* @author lidaamber

\*/

interface BasePresenter: NetworkAvailabilityPresenter

// BasePresenterImpl.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.SurveysProvider

import com.lidaamber.kpisurvey.network.NetworkAvailabilityManager

import com.lidaamber.kpisurvey.views.NetworkAvailabilityView

/\*\*

\* BasePresenter implementation

\* @author lidaamber

\*/

open class BasePresenterImpl(val context: Context,

private val networkView: NetworkAvailabilityView) : BasePresenter,

NetworkAvailabilityManager.NetworkListener {

/\*\*

\* Network availability manager

\*/

protected val networkAvailabilityManager = NetworkAvailabilityManager(context)

override fun onResume() {

networkAvailabilityManager.onResume()

}

override fun init() {

networkAvailabilityManager.setNetworkAvailabilityListener(this)

}

override fun onNetworkStateChanged() {

networkAvailabilityManager.onNetworkStateChanged()

}

override fun onConnected() {

SurveysProvider.init()

}

override fun onDisconnected() {

networkView.showNotConnectedMessage()

}

}

// ConfirmationPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.views.ConfirmationView

/\*\*

\* Presenter for token confirmation flow

\* @author lidaamber

\*/

interface ConfirmationPresenter : BasePresenter {

/\*\*

\* Event handler for submit click

\*/

fun onSubmitClicked(token: String)

}

/\*\*

\* ConfirmationPresenter implementation

\* @author lidaamber

\*/

class ConfirmationPresenterImpl(context: Context,

private val view: ConfirmationView) :

BasePresenterImpl(context, view),

ConfirmationPresenter {

override fun onSubmitClicked(token: String) {

AuthenticationProvider.confirmToken(token, {

view.showMainScreen()

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

}

// LoginPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.model.ConfigProvider

import com.lidaamber.kpisurvey.model.SurveysProvider

import com.lidaamber.kpisurvey.views.LoginView

/\*\*

\* Presenter for login flow

\* @author lidaamber

\*/

interface LoginPresenter : BasePresenter {

/\*\*

\* Event handler for submit click

\*/

fun onSubmitClicked(password: String?)

/\*\*

\* Event handler for register click

\*/

fun onRegisterClicked()

/\*\*

\* Reports question state change to enable/disable login button

\*/

fun reportQuestionsStateChange()

/\*\*

\* Event handler for reset password click

\*/

fun onResetPasswordClicked()

}

/\*\*

\* LoginPresenter implementation

\* @author lidaamber

\*/

class LoginPresenterImpl(context: Context,

private val view: LoginView) :

BasePresenterImpl(context, view), LoginPresenter {

/\*\*

\* Authentication questions

\*/

private lateinit var questions: List<AuthenticationQuestion>

override fun onResume() {

super.onResume()

if (networkAvailabilityManager.isNetworkAvailable)

SurveysProvider.init {

ConfigProvider.getLoginCredentialsQuestions(context) {

this.questions = it

view.showQuestions(it)

}

}

}

override fun onSubmitClicked(password: String?) {

if (password == null || password.isEmpty()) return

AuthenticationProvider.login(questions, password, {

view.showMainScreen()

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

override fun onRegisterClicked() {

view.showAuthenticationScreen()

}

override fun reportQuestionsStateChange() {

if (questions.all { it.answer != null }) view.showSubmitButton()

else view.hideSubmitButton()

}

override fun onResetPasswordClicked() {

view.showResetPasswordScreen()

}

}

// NetworkAvailabilityPresenter.kt

package com.lidaamber.kpisurvey.presenters

/\*\*

\* Presenter to track network availability

\* @author lidaamber

\*/

interface NetworkAvailabilityPresenter {

/\*\*

\* Sets network state on resume

\*/

fun onResume()

/\*\*

\* Initializes network availability manager

\*/

fun init()

/\*\*

\* Network state change event handler

\*/

fun onNetworkStateChanged()

}

// NewPasswordPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.views.NewPasswordView

/\*\*

\* Presenter for setting new password flow

\* @author lidaamber

\*/

interface NewPasswordPresenter : BasePresenter {

/\*\*

\* Confirmation token

\*/

var token: String

/\*\*

\* Event handler for confirmation password

\*/

fun onConfirmClicked(password: String)

}

/\*\*

\* NewPasswordPresenter implementation

\* @author lidaamber

\*/

class NewPasswordPresenterImpl(context: Context,

val view: NewPasswordView) :

BasePresenterImpl(context, view), NewPasswordPresenter {

override lateinit var token: String

override fun onConfirmClicked(password: String) {

AuthenticationProvider.updatePassword(token, password, {

view.showSignIn()

}, {

view.displayError(context.getString(it))

})

}

}

// RegistrationPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.views.RegistrationView

/\*\*

\* Presenter for registration flow

\* @author lidaamber

\*/

interface RegistrationPresenter : BasePresenter {

/\*\*

\* Event handler for register account click

\*/

fun onRegisterAccountClicked(email: String, password: String)

}

/\*\*

\* RegistrationPresenter implementation

\* @author lidaamber

\*/

class RegistrationPresenterImpl(context: Context,

val view: RegistrationView) :

BasePresenterImpl(context, view),

RegistrationPresenter {

override fun onRegisterAccountClicked(email: String, password: String) {

AuthenticationProvider.register(email, password, {

view.showConfirmationScreen()

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

}

// ResetPasswordEmailPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.AuthenticationProvider

import com.lidaamber.kpisurvey.views.ResetPasswordEmailView

/\*\*

\* Presenter for reset password flow

\* @author lidaamber

\*/

interface ResetPasswordEmailPresenter : BasePresenter {

/\*\*

\* Event handler for email text change

\*/

fun onEmailTextChanged(email: String)

/\*\*

\* Event handler for sign in click

\*/

fun onSignInClicked()

/\*\*

\* Event handler for email confirmation

\*/

fun onConfirmClicked(email: String)

}

/\*\*

\* ResetPasswordEmailPresenter implementation

\* @author lidaamber

\*/

class ResetPasswordEmailPresenterImpl(context: Context,

val view: ResetPasswordEmailView) :

BasePresenterImpl(context, view), ResetPasswordEmailPresenter {

override fun onEmailTextChanged(email: String) {

if (email.isEmpty()) view.hideConfirmButton() else view.showConfirmButton()

}

override fun onSignInClicked() {

view.showLoginScreen()

}

override fun onConfirmClicked(email: String) {

if (!networkAvailabilityManager.isNetworkAvailable) {

view.showNotConnectedMessage()

return

}

AuthenticationProvider.restorePassword(email, {

view.showResetPasswordTokenScreen()

}, {

view.displayError(context.getString(it))

})

}

}

// ResetPasswordTokenPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.views.ResetPasswordTokenView

/\*\*

\* Presenter for reset password flow

\*/

interface ResetPasswordTokenPresenter : BasePresenter {

/\*\*

\* Event handler for token text change

\*/

fun onTokenTextChanged(token: String)

/\*\*

\* Event handler for changing email

\*/

fun onChangeEmailClicked()

/\*\*

\* Event handler for token confirmation

\*/

fun onConfirmClicked(token: String)

}

/\*\*

\* ResetPasswordTokenPresenter implementation

\*/

class ResetPasswordTokenPresenterImpl(context: Context, val view: ResetPasswordTokenView) :

BasePresenterImpl(context, view), ResetPasswordTokenPresenter {

override fun onTokenTextChanged(token: String) {

if(token.isEmpty()) view.hideConfirmButton() else view.showConfirmButton()

}

override fun onChangeEmailClicked() {

view.showResetEmailScreen()

}

override fun onConfirmClicked(token: String) {

view.showNewPasswordScreen(token)

}

}

// SurveyPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.Question

import com.lidaamber.kpisurvey.model.SurveyId

import com.lidaamber.kpisurvey.model.SurveysProvider

import com.lidaamber.kpisurvey.views.SurveyView

/\*\*

\* Presenter for survey passing flow

\* @author lidaamber

\*/

interface SurveyPresenter : BasePresenter {

/\*\*

\* Sets survey id

\*/

fun setSurveyId(surveyId: SurveyId)

/\*\*

\* Event handler for answer event

\*/

fun onAnswer(question: Question, answer: Any)

}

/\*\*

\* SurveyPresenter implementation

\* @author lidaamber

\*/

class SurveyPresenterImpl(context: Context,

private val view: SurveyView) :

BasePresenterImpl(context, view), SurveyPresenter {

/\*\*

\* Survey id

\*/

private var surveyId: SurveyId? = null

/\*\*

\* Survey questions

\*/

private lateinit var questions: List<Question>

/\*\*

\* Displaying survey passed state

\*/

private var shouldDisplaySurveyPassed: Boolean = true

override fun setSurveyId(surveyId: SurveyId) {

this.surveyId = surveyId

SurveysProvider.getSurveyQuestions(surveyId, {

this.questions = it

shouldDisplaySurveyPassed = !questions.all { it.answer != null }

view.displayQuestions(questions)

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

override fun onAnswer(question: Question, answer: Any) {

if (surveyId == null) return

SurveysProvider.answer(surveyId!!, questions.indexOf(question), answer, {

question.answer = answer

if (questions.all { it.answer != null } && shouldDisplaySurveyPassed) {

shouldDisplaySurveyPassed = false

view.displaySurveyPassed()

}

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

}

// SurveysListPresenter.kt

package com.lidaamber.kpisurvey.presenters

import android.content.Context

import com.lidaamber.kpisurvey.model.Survey

import com.lidaamber.kpisurvey.model.SurveysProvider

import com.lidaamber.kpisurvey.views.SurveysListView

/\*\*

\* Presenter for survey list flow

\* @author lidaamber

\*/

interface SurveysListPresenter : BasePresenter {

/\*\*

\* Event handler for survey click

\*/

fun onSurveyClicked(survey: Survey)

}

/\*\*

\* SurveysListPresenter implementation

\* @author lidaamber

\*/

class SurveysListPresenterImpl(context: Context,

val view: SurveysListView) :

BasePresenterImpl(context, view), SurveysListPresenter {

override fun onResume() {

super.onResume()

SurveysProvider.getSurveys({

view.showSurveys(it)

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

override fun onSurveyClicked(survey: Survey) {

view.showSurveyScreen(survey.id, survey.title)

}

override fun onConnected() {

super.onConnected()

SurveysProvider.getSurveys({

view.showSurveys(it)

}, { errorRes ->

view.displayError(context.getString(errorRes))

})

}

}

// AuthenticationAdapter.kt

package com.lidaamber.kpisurvey.ui.adapters

import android.content.Context

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.utils.addTextChangedListener

import com.lidaamber.kpisurvey.utils.setDatePicker

import com.lidaamber.kpisurvey.utils.toFormattedDate

import kotlinx.android.synthetic.main.auth\_question\_date.view.\*

import kotlinx.android.synthetic.main.auth\_question\_number.view.\*

import kotlinx.android.synthetic.main.auth\_question\_text.view.\*

/\*\*

\* Adapter for authentication questions

\* @author lidaamber

\*/

class AuthenticationAdapter(private val questions: List<AuthenticationQuestion>,

private val context: Context,

private val onFieldsFilled: () -> Unit) :

RecyclerView.Adapter<AuthenticationAdapter.Holder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {

val resource = when (viewType) {

AuthenticationQuestion.DATE -> R.layout.auth\_question\_date

AuthenticationQuestion.NUMBER -> R.layout.auth\_question\_number

else -> R.layout.auth\_question\_text

}

val v = LayoutInflater.from(parent.context).inflate(resource, parent, false)

return when (viewType) {

AuthenticationQuestion.DATE -> DateHolder(v)

AuthenticationQuestion.NUMBER -> NumberHolder(v)

else -> TextHolder(v)

}

}

override fun getItemViewType(position: Int) = questions[position].type

override fun getItemCount() = questions.size

override fun onBindViewHolder(holder: Holder, position: Int) {

holder.bind(questions[position])

}

/\*\*

\* Authentication question holder

\*/

abstract inner class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {

/\*\*

\* Binds question data to view

\*/

abstract fun bind(question: AuthenticationQuestion)

}

/\*\*

\* Text authentication question holder

\*/

inner class TextHolder(itemView: View) : Holder(itemView) {

override fun bind(question: AuthenticationQuestion) {

itemView.textInputLayout.hint = question.title

itemView.textEditText.addTextChangedListener {

if (it == null) return@addTextChangedListener

question.answer = if (it.isEmpty()) null else it.toString()

onFieldsFilled()

}

}

}

/\*\*

\* Number authentication question holder

\*/

inner class NumberHolder(itemView: View) : Holder(itemView) {

override fun bind(question: AuthenticationQuestion) {

itemView.numberInputLayout.hint = question.title

itemView.numberEditText.addTextChangedListener {

if (it == null) return@addTextChangedListener

question.answer = if (it.isEmpty()) null else it.toString().toDouble()

onFieldsFilled()

}

}

}

/\*\*

\* Date authentication question holder

\*/

inner class DateHolder(itemView: View) : Holder(itemView) {

override fun bind(question: AuthenticationQuestion) {

itemView.dateInputLayout.hint = question.title

itemView.dateEditText.setDatePicker(context) {

question.answer = it.time

itemView.dateEditText.setText(it.toFormattedDate())

onFieldsFilled()

}

}

}

}

// SurveyAdapter.kt

package com.lidaamber.kpisurvey.ui.adapters

import android.content.Context

import android.support.v4.content.ContextCompat

import android.support.v4.content.res.ResourcesCompat

import android.support.v7.widget.AppCompatCheckBox

import android.support.v7.widget.AppCompatRadioButton

import android.support.v7.widget.RecyclerView

import android.text.Editable

import android.text.TextWatcher

import android.util.TypedValue

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.RadioButton

import android.widget.RadioGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.Question

import kotlinx.android.synthetic.main.number\_question\_item.view.\*

import kotlinx.android.synthetic.main.select\_many\_question\_item.view.\*

import kotlinx.android.synthetic.main.select\_one\_question\_item.view.\*

import kotlinx.android.synthetic.main.text\_question\_item.view.\*

/\*\*

\* Survey questions adapter

\* @author lidaamber

\*/

class SurveyAdapter(private val context: Context,

private val questions: List<Question>,

private val onAnswer: (Question, Any) -> Unit) :

RecyclerView.Adapter<SurveyAdapter.QuestionHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): QuestionHolder {

val inflater = LayoutInflater.from(parent.context)

return when (viewType) {

Question.TEXT\_QUESTION -> {

val v = inflater.inflate(R.layout.text\_question\_item, parent, false)

TextQuestionHolder(v)

}

Question.SELECT\_ONE\_QUESTION -> {

val v = inflater.inflate(R.layout.select\_one\_question\_item, parent, false)

SelectOneQuestionHolder(v)

}

Question.SELECT\_MANY\_QUESTION -> {

val v = inflater.inflate(R.layout.select\_many\_question\_item, parent, false)

SelectManyQuestionHolder(v)

}

else -> {

val v = inflater.inflate(R.layout.number\_question\_item, parent, false)

NumberQuestionHolder(v)

}

}

}

override fun getItemViewType(position: Int): Int {

return questions[position].type

}

override fun getItemCount() = questions.size

override fun onBindViewHolder(holder: QuestionHolder, position: Int) {

holder.bind(questions[position], position + 1)

}

/\*\*

\* Survey question holder

\*/

abstract inner class QuestionHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

/\*\*

\* Binds question data to view

\*/

abstract fun bind(question: Question, index: Int)

}

/\*\*

\* Text survey question holder

\*/

inner class TextQuestionHolder(itemView: View) : QuestionHolder(itemView) {

override fun bind(question: Question, index: Int) {

itemView.textQuestionTitle.text = question.title

itemView.textIndexTextView.text = index.toString()

if (question.answer != null) {

itemView.textQuestionEditText.setText(question.answer as String)

}

val textWatcher = object : TextWatcher {

override fun afterTextChanged(s: Editable?) {

}

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

if (s == null) return

question.answer = s.toString()

onAnswer(question, s.toString())

if (question.max != null && s.length > question.max) {

val errorText = String.format(context.getString(R.string.max\_error),

question.max)

itemView.textQuestionEditText.error = errorText

} else if (question.min != null && s.length < question.min) {

val errorText = String.format(context.getString(R.string.min\_error),

question.min)

itemView.textQuestionEditText.error = errorText

} else {

itemView.textQuestionEditText.error = null

question.answer = s.toString()

onAnswer(question, s.toString())

}

}

}

itemView.textQuestionEditText.addTextChangedListener(textWatcher)

}

}

/\*\*

\* Select one survey question holder

\*/

inner class SelectOneQuestionHolder(itemView: View) : QuestionHolder(itemView) {

override fun bind(question: Question, index: Int) {

itemView.oneQuestionTitle.text = question.title

itemView.oneIndexTextView.text = index.toString()

if (question.answers == null) return

question.answers.forEach {

val radioButton = AppCompatRadioButton(context)

val params = RadioGroup.LayoutParams(ViewGroup.LayoutParams.MATCH\_PARENT,

ViewGroup.LayoutParams.WRAP\_CONTENT)

radioButton.layoutParams = params

radioButton.text = it

radioButton.setTextColor(ContextCompat.getColor(context, R.color.surveyTitleColor))

radioButton.typeface = ResourcesCompat.getFont(context, R.font.muli)

radioButton.setButtonDrawable(R.drawable.check\_box)

radioButton.setPadding(40, 20, 10, 20)

radioButton.setTextSize(TypedValue.COMPLEX\_UNIT\_SP, 16.0f)

radioButton.setOnCheckedChangeListener { \_, isChecked ->

if (isChecked) {

question.answer = it

onAnswer(question, it)

}

}

itemView.selectOneGroup.addView(radioButton)

}

if (question.answer == null) return

val answerIndex = question.answers.indexOf(question.answer)

(itemView.selectOneGroup.getChildAt(answerIndex) as RadioButton).isChecked = true

}

}

/\*\*

\* Select many survey question holder

\*/

inner class SelectManyQuestionHolder(itemView: View) : QuestionHolder(itemView) {

override fun bind(question: Question, index: Int) {

itemView.manyQuestionTitle.text = question.title

itemView.manyIndexTextView.text = index.toString()

if (question.answers == null) return

val userAnswers = ArrayList<String>()

question.answers.forEach {

val checkBox = AppCompatCheckBox(context)

val params = RadioGroup.LayoutParams(ViewGroup.LayoutParams.MATCH\_PARENT,

ViewGroup.LayoutParams.WRAP\_CONTENT)

checkBox.layoutParams = params

checkBox.typeface = ResourcesCompat.getFont(context, R.font.muli)

checkBox.setTextColor(ContextCompat.getColor(context, R.color.surveyTitleColor))

checkBox.text = it

checkBox.setTextSize(TypedValue.COMPLEX\_UNIT\_SP, 16.0f)

checkBox.setButtonDrawable(R.drawable.check\_box)

checkBox.setPadding(40, 20, 10, 20)

if (question.answer != null && (question.answer!! as List<\*>).contains(it))

checkBox.isChecked = true

checkBox.setOnCheckedChangeListener { \_, isChecked ->

if (isChecked) {

userAnswers.add(it)

} else {

userAnswers.remove(it)

}

question.answer = userAnswers

onAnswer(question, userAnswers)

}

itemView.multipleAnswersContainer.addView(checkBox, params)

}

}

}

/\*\*

\* Number survey question holder

\*/

inner class NumberQuestionHolder(itemView: View) : QuestionHolder(itemView) {

override fun bind(question: Question, index: Int) {

itemView.numberQuestionTitle.text = question.title

itemView.numberIndexTextView.text = index.toString()

if (question.answer != null) {

itemView.numberEditText.setText(question.answer.toString())

}

val textWatcher = object : TextWatcher {

override fun afterTextChanged(s: Editable?) {

if (s == null) return

}

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

if (s == null || s.isEmpty()) return

val number = s.toString().toDouble()

if (question.max != null && number > question.max) {

val errorText = String.format(context.getString(R.string.max\_error),

question.max)

itemView.numberEditText.error = errorText

} else if (question.min != null && number < question.min) {

val errorText = String.format(context.getString(R.string.min\_error),

question.min)

itemView.numberEditText.error = errorText

} else {

itemView.numberEditText.error = null

question.answer = number

onAnswer(question, number)

}

}

}

itemView.numberEditText.addTextChangedListener(textWatcher)

}

}

}

// SurveysListAdapter.kt

package com.lidaamber.kpisurvey.ui.adapters

import android.content.Context

import android.support.v4.content.ContextCompat

import android.support.v7.widget.RecyclerView

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.Survey

import kotlinx.android.synthetic.main.survey\_list\_item.view.\*

import java.text.SimpleDateFormat

import java.util.\*

/\*\*

\* Surveys adapter

\* @author lidaamber

\*/

class SurveysListAdapter(private val surveys: List<Survey>,

private val context: Context,

private val onClick: (Survey) -> Unit) : RecyclerView.Adapter<SurveysListAdapter.SurveyHolder>() {

companion object {

/\*\*

\* Survey date pattern

\*/

private const val DATE\_PATTERN = "dd/MM/yyyy"

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SurveyHolder {

val v = LayoutInflater.from(parent.context)

.inflate(R.layout.survey\_list\_item, parent, false)

return SurveyHolder(v)

}

override fun getItemCount() = surveys.size

override fun onBindViewHolder(holder: SurveyHolder, position: Int) {

holder.bind(surveys[position])

}

/\*\*

\* Survey data holder

\*/

inner class SurveyHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

/\*\*

\* Binds survey data to view

\*/

fun bind(survey: Survey) {

itemView.surveyTitleTextView.text = survey.title

val dateFormat = SimpleDateFormat(DATE\_PATTERN, Locale.getDefault())

itemView.surveyDateTextView.text = dateFormat.format(survey.created)

val alpha = if (survey.passed) 0.7f else 1.0f

val indicatorColor = if (survey.passed) R.color.registerTintColor

else R.color.signInTintColor

itemView.surveyDateTextView.alpha = alpha

itemView.surveyTitleTextView.alpha = alpha

itemView.passedIndicator

.setBackgroundColor(ContextCompat.getColor(context, indicatorColor))

itemView?.setOnClickListener { onClick(survey) }

}

}

}

// AuthenticationFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.animation.AnimatorSet

import android.animation.ObjectAnimator

import android.os.Bundle

import android.support.v7.widget.LinearLayoutManager

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.presenters.AuthenticationPresenter

import com.lidaamber.kpisurvey.presenters.AuthenticationPresenterImpl

import com.lidaamber.kpisurvey.ui.adapters.AuthenticationAdapter

import com.lidaamber.kpisurvey.views.AuthenticationView

import kotlinx.android.synthetic.main.fragment\_authentication.\*

/\*\*

\* AuthenticationView implementation

\* @author lidaamber

\*/

class AuthenticationFragment : SlideAnimationFragment<AuthenticationPresenter>(), AuthenticationView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_authentication, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

buttonsContainer

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

signInButton.setOnClickListener {

presenter.onSignInClicked()

}

authenticateButton.setOnClickListener {

if (activity == null) return@setOnClickListener

presenter.onAuthenticateClicked()

}

}

override fun showAuthenticateButton() {

if (signInButton.getWeight() == 0.5f) return

val rightView = ObjectAnimator.ofFloat(signInButton, "Weight",

signInButton.getWeight(), 0.5f)

val leftView = ObjectAnimator.ofFloat(authenticateButton, "Weight",

authenticateButton.getWeight(), 0.5f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun hideAuthenticateButton() {

if (signInButton.getWeight() == 0.0f) return

val rightView = ObjectAnimator.ofFloat(signInButton, "Weight",

signInButton.getWeight(), 0.0f)

val leftView = ObjectAnimator.ofFloat(authenticateButton, "Weight",

authenticateButton.getWeight(), 1f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

questionsRecyclerView.layoutManager = LinearLayoutManager(activity)

presenter = AuthenticationPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun showRegistrationScreen() {

if (activity == null) return

val transaction = createSlideTransaction(RegistrationFragment(), false)

transaction.commit()

}

override fun showLoginScreen() {

val transaction = createSlideTransaction(LoginFragment(), false)

transaction.commit()

}

override fun showQuestions(questions: List<AuthenticationQuestion>) {

questionsRecyclerView.adapter = AuthenticationAdapter(questions, activity!!) {

presenter.reportQuestionsStateChange()

}

}

}

// ConfirmationFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.content.Intent

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.ConfirmationPresenter

import com.lidaamber.kpisurvey.presenters.ConfirmationPresenterImpl

import com.lidaamber.kpisurvey.ui.BaseFragment

import com.lidaamber.kpisurvey.ui.survey.SurveysActivity

import com.lidaamber.kpisurvey.views.ConfirmationView

import kotlinx.android.synthetic.main.fragment\_confirmation.\*

/\*\*

\* ConfirmationView implementation

\* @author lidaamber

\*/

class ConfirmationFragment : BaseFragment<ConfirmationPresenter>(), ConfirmationView {

companion object {

/\*\*

\* Animation duration constants

\*/

const val ANIMATION\_DURATION = 500L

const val ANIMATION\_DELAY = 500L

}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_confirmation, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

confirmButton

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

confirmButton.setOnClickListener {

presenter.onSubmitClicked(tokenEditText.text.toString())

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

presenter = ConfirmationPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun showMainScreen() {

if (activity == null) return

val i = Intent(activity, SurveysActivity::class.java)

i.flags = Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_CLEAR\_TASK

startActivity(i)

}

}

// LoginFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.animation.AnimatorSet

import android.animation.ObjectAnimator

import android.content.Intent

import android.os.Bundle

import android.support.v4.content.res.ResourcesCompat

import android.support.v7.widget.LinearLayoutManager

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

import com.lidaamber.kpisurvey.presenters.LoginPresenter

import com.lidaamber.kpisurvey.presenters.LoginPresenterImpl

import com.lidaamber.kpisurvey.ui.adapters.AuthenticationAdapter

import com.lidaamber.kpisurvey.ui.survey.SurveysActivity

import com.lidaamber.kpisurvey.views.LoginView

import kotlinx.android.synthetic.main.fragment\_login.\*

/\*\*

\* LoginView implementation

\* @author lidaamber

\*/

class LoginFragment : SlideAnimationFragment<LoginPresenter>(), LoginView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_login, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

buttonsContainer

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

registerButton.setOnClickListener { presenter.onRegisterClicked() }

resetPasswordButton.setOnClickListener { presenter.onResetPasswordClicked() }

submitButton.setOnClickListener {

presenter.onSubmitClicked(passwordEditText.text.toString())

}

}

override fun showSubmitButton() {

if (registerButton.getWeight() == 0.5f) return

val rightView = ObjectAnimator.ofFloat(registerButton, "Weight",

registerButton.getWeight(), 0.5f)

val leftView = ObjectAnimator.ofFloat(submitButton, "Weight",

submitButton.getWeight(), 0.5f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun hideSubmitButton() {

if (registerButton.getWeight() == 0.0f) return

val rightView = ObjectAnimator.ofFloat(registerButton, "Weight",

registerButton.getWeight(), 0.0f)

val leftView = ObjectAnimator.ofFloat(submitButton, "Weight", submitButton.getWeight(), 1f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

questionsRecyclerView.layoutManager = LinearLayoutManager(activity)

presenter = LoginPresenterImpl(activity!!, this)

presenter.init()

val font = ResourcesCompat

.getFont(activity!!.applicationContext, R.font.muli\_extralight)

passwordInputLayout.setTypeface(font)

passwordEditText.typeface = font

}

override fun showQuestions(questions: List<AuthenticationQuestion>) {

questionsRecyclerView.adapter = AuthenticationAdapter(questions, activity!!) {

presenter.reportQuestionsStateChange()

}

}

override fun showMainScreen() {

if (activity == null) return

val i = Intent(activity, SurveysActivity::class.java)

i.flags = Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_CLEAR\_TASK

startActivity(i)

}

override fun showAuthenticationScreen() {

val transaction = createSlideTransaction(AuthenticationFragment())

transaction.commit()

}

override fun showResetPasswordScreen() {

val transaction = createSlideTransaction(ResetPasswordEmailFragment())

transaction.commit()

}

}

// NewPasswordFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.NewPasswordPresenter

import com.lidaamber.kpisurvey.presenters.NewPasswordPresenterImpl

import com.lidaamber.kpisurvey.views.NewPasswordView

import kotlinx.android.synthetic.main.fragment\_new\_password.\*

/\*\*

\* NewPasswordView implementation

\* @author lidaamber

\*/

class NewPasswordFragment : SlideAnimationFragment<NewPasswordPresenter>(), NewPasswordView {

companion object {

/\*\*

\* Token key constant

\*/

private const val TOKEN = "token"

/\*\*

\* Creates new instance of NewPasswordFragment with token passed

\*/

fun newInstance(token: String) : NewPasswordFragment {

val args = Bundle()

args.putString(TOKEN, token)

return NewPasswordFragment().also { it.arguments = args }

}

}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_new\_password, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

confirmButton

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

confirmButton.setOnClickListener {

presenter.onConfirmClicked(passwordEditText.text.toString())

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

activity?.let {

presenter = NewPasswordPresenterImpl(it.applicationContext, this)

presenter.init()

arguments?.let {

presenter.token = it.getString(TOKEN)

}

}

}

override fun showSignIn() {

val transaction = createSlideTransaction(LoginFragment(), false)

transaction.commit()

}

}

// RegistrationFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.os.Bundle

import android.support.v4.content.res.ResourcesCompat

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.RegistrationPresenter

import com.lidaamber.kpisurvey.presenters.RegistrationPresenterImpl

import com.lidaamber.kpisurvey.views.RegistrationView

import kotlinx.android.synthetic.main.fragment\_registration.\*

/\*\*

\* RegistrationView implementation

\* @author lidaamber

\*/

class RegistrationFragment : SlideAnimationFragment<RegistrationPresenter>(), RegistrationView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_registration, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

registerButton

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

registerButton.setOnClickListener {

presenter.onRegisterAccountClicked(emailEditText.text.toString(),

passwordEditText.text.toString())

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

presenter = RegistrationPresenterImpl(activity!!, this)

presenter.init()

val font = ResourcesCompat

.getFont(activity!!.applicationContext, R.font.muli\_extralight)

passwordInputLayout.setTypeface(font)

passwordEditText.typeface = font

}

override fun showConfirmationScreen() {

val transaction = createSlideTransaction(ConfirmationFragment())

transaction.commit()

}

}

// ResetPasswordEmailFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.animation.AnimatorSet

import android.animation.ObjectAnimator

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.ResetPasswordEmailPresenter

import com.lidaamber.kpisurvey.presenters.ResetPasswordEmailPresenterImpl

import com.lidaamber.kpisurvey.utils.addTextChangedListener

import com.lidaamber.kpisurvey.views.ResetPasswordEmailView

import kotlinx.android.synthetic.main.fragment\_reset\_password\_email.\*

/\*\*

\* ResetPasswordEmailView implementation

\* @author lidaamber

\*/

class ResetPasswordEmailFragment : SlideAnimationFragment<ResetPasswordEmailPresenter>(),

ResetPasswordEmailView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_reset\_password\_email, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

buttonsContainer

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

emailEditText.addTextChangedListener {

if (it == null) return@addTextChangedListener

presenter.onEmailTextChanged(it.toString())

}

signInButton.setOnClickListener {

presenter.onSignInClicked()

}

confirmButton.setOnClickListener {

presenter.onConfirmClicked(emailEditText.text.toString())

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

presenter = ResetPasswordEmailPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun showConfirmButton() {

if (signInButton.getWeight() == 0.5f) return

val rightView = ObjectAnimator.ofFloat(signInButton, "Weight",

signInButton.getWeight(), 0.5f)

val leftView = ObjectAnimator.ofFloat(confirmButton, "Weight",

confirmButton.getWeight(), 0.5f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun hideConfirmButton() {

if (signInButton.getWeight() == 0.0f) return

val rightView = ObjectAnimator.ofFloat(signInButton, "Weight",

signInButton.getWeight(), 0.0f)

val leftView = ObjectAnimator.ofFloat(confirmButton, "Weight",

confirmButton.getWeight(), 1f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun showLoginScreen() {

val transaction = createSlideTransaction(LoginFragment(), false)

transaction.commit()

}

override fun showResetPasswordTokenScreen() {

val transaction = createSlideTransaction(ResetPasswordTokenFragment(), false)

transaction.commit()

}

}

// ResetPasswordTokenFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.animation.AnimatorSet

import android.animation.ObjectAnimator

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.ResetPasswordTokenPresenter

import com.lidaamber.kpisurvey.presenters.ResetPasswordTokenPresenterImpl

import com.lidaamber.kpisurvey.utils.addTextChangedListener

import com.lidaamber.kpisurvey.views.ResetPasswordTokenView

import kotlinx.android.synthetic.main.fragment\_reset\_password\_token.\*

/\*\*

\* ResetPasswordTokenView implementation

\* @author lidaamber

\*/

class ResetPasswordTokenFragment : SlideAnimationFragment<ResetPasswordTokenPresenter>(),

ResetPasswordTokenView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_reset\_password\_token, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

buttonsContainer

.animate()

.alpha(1.0f)

.setStartDelay(ANIMATION\_DELAY)

.setDuration(ANIMATION\_DURATION)

.start()

tokenEditText.addTextChangedListener {

if (it == null) return@addTextChangedListener

presenter.onTokenTextChanged(it.toString())

}

changeEmailButton.setOnClickListener {

presenter.onChangeEmailClicked()

}

confirmButton.setOnClickListener {

presenter.onConfirmClicked(tokenEditText.text.toString())

}

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

presenter = ResetPasswordTokenPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun showConfirmButton() {

if (changeEmailButton.getWeight() == 0.5f) return

val rightView = ObjectAnimator.ofFloat(changeEmailButton, "Weight",

changeEmailButton.getWeight(), 0.5f)

val leftView = ObjectAnimator.ofFloat(confirmButton, "Weight",

confirmButton.getWeight(), 0.5f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun hideConfirmButton() {

if (changeEmailButton.getWeight() == 0.0f) return

val rightView = ObjectAnimator.ofFloat(changeEmailButton, "Weight",

changeEmailButton.getWeight(), 0.0f)

val leftView = ObjectAnimator.ofFloat(confirmButton, "Weight",

confirmButton.getWeight(), 1f)

val animatorSet = AnimatorSet()

animatorSet.duration = BUTTON\_ANIMATION\_DURATION

animatorSet.playTogether(rightView, leftView)

animatorSet.start()

}

override fun showResetEmailScreen() {

val transaction = createSlideTransaction(ResetPasswordEmailFragment())

transaction.commit()

}

override fun showNewPasswordScreen(token: String) {

val transaction = createSlideTransaction(NewPasswordFragment.newInstance(token))

transaction.commit()

}

}

// SlideAnimationFragment.kt

package com.lidaamber.kpisurvey.ui.start

import android.support.v4.app.Fragment

import android.support.v4.app.FragmentTransaction

import android.view.View

import android.view.animation.Animation

import android.view.animation.AnimationUtils

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.BasePresenter

import com.lidaamber.kpisurvey.ui.BaseFragment

/\*\*

\* Simple fragment that handles slide animations

\* @author lidaamber

\*/

open class SlideAnimationFragment<T: BasePresenter> : BaseFragment<T>() {

companion object {

/\*\*

\* Animation duration constants

\*/

const val ANIMATION\_DURATION = 500L

const val ANIMATION\_DELAY = 500L

const val BUTTON\_ANIMATION\_DURATION = 1000L

}

override fun onCreateAnimation(transit: Int, enter: Boolean, nextAnim: Int): Animation? {

val result = super.onCreateAnimation(transit, enter, nextAnim)

if (activity?.supportFragmentManager?.backStackEntryCount == 0) {

return result

}

var animation = result

if (animation == null && nextAnim != 0) {

animation = AnimationUtils.loadAnimation(activity, nextAnim)

}

if (animation != null) {

view?.setLayerType(View.LAYER\_TYPE\_HARDWARE, null)

animation.setAnimationListener(object : Animation.AnimationListener {

override fun onAnimationRepeat(animation: Animation?) {

}

override fun onAnimationEnd(animation: Animation?) {

view?.setLayerType(View.LAYER\_TYPE\_NONE, null)

}

override fun onAnimationStart(animation: Animation?) {

}

})

}

return animation ?: result

}

fun createSlideTransaction(fragment: Fragment, fromLeft: Boolean = true): FragmentTransaction {

val transaction = activity!!.supportFragmentManager.beginTransaction()

if (fromLeft)

transaction.setCustomAnimations(R.anim.enter\_from\_left, R.anim.exit\_to\_right,

R.anim.enter\_from\_right, R.anim.exit\_to\_left)

else transaction.setCustomAnimations(R.anim.enter\_from\_right,

R.anim.exit\_to\_left, R.anim.enter\_from\_left, R.anim.exit\_to\_right)

return transaction.replace(R.id.fragmentContainer, fragment)

}

}

// StartActivity.kt

package com.lidaamber.kpisurvey.ui.start

import android.os.Bundle

import android.support.v7.app.AppCompatActivity

import com.lidaamber.kpisurvey.R

/\*\*

\* Start activity that handles authentication flow

\* @author lidaamber

\*/

class StartActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_one\_fragment)

var fragment = supportFragmentManager.findFragmentById(R.id.fragmentContainer)

if (fragment == null) {

fragment = LoginFragment()

supportFragmentManager.beginTransaction()

.add(R.id.fragmentContainer, fragment)

.commit()

}

}

}

// SurveyFragment.kt

package com.lidaamber.kpisurvey.ui.survey

import android.os.Bundle

import android.support.design.widget.Snackbar

import android.support.v4.content.ContextCompat

import android.support.v7.widget.LinearLayoutManager

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.Question

import com.lidaamber.kpisurvey.model.SurveyId

import com.lidaamber.kpisurvey.presenters.SurveyPresenter

import com.lidaamber.kpisurvey.presenters.SurveyPresenterImpl

import com.lidaamber.kpisurvey.ui.BaseFragment

import com.lidaamber.kpisurvey.ui.adapters.SurveyAdapter

import com.lidaamber.kpisurvey.views.SurveyView

import kotlinx.android.synthetic.main.fragment\_survey.\*

/\*\*

\* SurveyView implementation

\* @author lidaamber

\*/

class SurveyFragment : BaseFragment<SurveyPresenter>(), SurveyView {

companion object {

/\*\*

\* Keys constants

\*/

private const val KEY\_ID = "id"

private const val KEY\_TITLE = "title"

/\*\*

\* Creates new instance of SurveyFragment and sets id and title

\*/

fun newInstance(surveyId: SurveyId, name: String): SurveyFragment {

val args = Bundle()

args.putInt(KEY\_ID, surveyId)

args.putString(KEY\_TITLE, name)

val fragment = SurveyFragment()

fragment.arguments = args

return fragment

}

}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_survey, container, false)

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

surveyRecyclerView.layoutManager = LinearLayoutManager(activity!!)

title.text = arguments?.getString(KEY\_TITLE)

toolbar.setNavigationOnClickListener {

activity?.supportFragmentManager?.beginTransaction()

?.replace(R.id.fragmentContainer, SurveysListFragment())

?.commit()

}

presenter = SurveyPresenterImpl(activity!!.applicationContext, this)

presenter.init()

presenter.setSurveyId(arguments!!.getInt(KEY\_ID))

}

override fun displayQuestions(questions: List<Question>) {

surveyRecyclerView.adapter = SurveyAdapter(activity!!, questions) { question, answer ->

presenter.onAnswer(question, answer)

}

}

override fun displaySurveyPassed() {

if (activity == null || view == null) return

val color = ContextCompat.getColor(activity!!, R.color.signInTintColor)

val snackBar = Snackbar.make(view!!, R.string.congratulations, Snackbar.LENGTH\_LONG)

snackBar.view.setBackgroundColor(color)

snackBar.show()

}

}

// SurveysActivity.kt

package com.lidaamber.kpisurvey.ui.survey

import android.os.Bundle

import android.support.v7.app.AppCompatActivity

import com.lidaamber.kpisurvey.R

/\*\*

\* Simple activity that handles surveys flow

\* @author lidaamber

\*/

class SurveysActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_one\_fragment)

var fragment = supportFragmentManager.findFragmentById(R.id.fragmentContainer)

if (fragment == null) {

fragment = SurveysListFragment()

supportFragmentManager.beginTransaction()

.add(R.id.fragmentContainer, fragment)

.commit()

}

}

}

// SurveysListFragment.kt

package com.lidaamber.kpisurvey.ui.survey

import android.os.Bundle

import android.support.v7.widget.LinearLayoutManager

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.model.Survey

import com.lidaamber.kpisurvey.model.SurveyId

import com.lidaamber.kpisurvey.presenters.SurveysListPresenter

import com.lidaamber.kpisurvey.presenters.SurveysListPresenterImpl

import com.lidaamber.kpisurvey.ui.BaseFragment

import com.lidaamber.kpisurvey.ui.adapters.SurveysListAdapter

import com.lidaamber.kpisurvey.views.SurveysListView

import kotlinx.android.synthetic.main.fragment\_surveys\_list.\*

/\*\*

\* SurveysListView implementation

\* @author lidaamber

\*/

class SurveysListFragment : BaseFragment<SurveysListPresenter>(), SurveysListView {

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

return inflater.inflate(R.layout.fragment\_surveys\_list, container, false)

}

override fun onActivityCreated(savedInstanceState: Bundle?) {

super.onActivityCreated(savedInstanceState)

surveysRecyclerView.layoutManager = LinearLayoutManager(activity!!)

presenter = SurveysListPresenterImpl(activity!!.applicationContext, this)

presenter.init()

}

override fun showSurveys(surveys: List<Survey>) {

if (activity == null) return

surveysRecyclerView.adapter = SurveysListAdapter(surveys, activity!!) {

presenter.onSurveyClicked(it)

}

}

override fun showSurveyScreen(surveyId: SurveyId, title: String) {

if (activity == null) return

val fragment = SurveyFragment.newInstance(surveyId, title)

activity!!.supportFragmentManager.beginTransaction()

.replace(R.id.fragmentContainer, fragment)

.commit()

}

}

// BaseFragment.kt

package com.lidaamber.kpisurvey.ui

import android.content.BroadcastReceiver

import android.content.Context

import android.content.Intent

import android.content.IntentFilter

import android.net.ConnectivityManager

import android.support.v4.app.Fragment

import com.lidaamber.kpisurvey.R

import com.lidaamber.kpisurvey.presenters.BasePresenter

import com.lidaamber.kpisurvey.utils.showMessage

import com.lidaamber.kpisurvey.views.BaseView

/\*\*

\* BaseView implementation

\* @author lidaamber

\*/

open class BaseFragment<T : BasePresenter> : Fragment(), BaseView {

/\*\*

\* Presenter used with view

\*/

protected lateinit var presenter: T

/\*\*

\* Network events receiver

\*/

private lateinit var networkReceiver: BroadcastReceiver

override fun onStart() {

super.onStart()

networkReceiver = object : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

presenter.onNetworkStateChanged()

}

}

activity!!.registerReceiver(networkReceiver,

IntentFilter(ConnectivityManager.CONNECTIVITY\_ACTION))

}

override fun showNotConnectedMessage() {

context?.let{

view?.showMessage(it.getString(R.string.not\_connected))

}

}

override fun onResume() {

super.onResume()

presenter.onResume()

}

override fun onStop() {

super.onStop()

activity!!.unregisterReceiver(networkReceiver)

}

override fun displayError(message: String) {

view?.showMessage(message)

}

}

// Extensions.kt

package com.lidaamber.kpisurvey.utils

import android.app.DatePickerDialog

import android.content.Context

import android.support.design.widget.Snackbar

import android.support.design.widget.TextInputEditText

import android.text.Editable

import android.text.TextWatcher

import android.view.View

import android.widget.EditText

import java.text.SimpleDateFormat

import java.util.\*

/\*\*

\* Adds simple text change listener

\* @author lidaamber

\*/

fun EditText.addTextChangedListener(listener: (CharSequence?) -> Unit) {

addTextChangedListener(object : TextWatcher {

override fun afterTextChanged(s: Editable?) {

}

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

listener(s)

}

})

}

/\*\*

\* Shows message

\* @author lidaamber

\*/

fun View.showMessage(string: String) {

Snackbar.make(this, string, Snackbar.LENGTH\_LONG).show()

}

/\*\*

\* Formats calendar date to appropriate format

\* @author lidaamber

\*/

fun Calendar.toFormattedDate(): String {

val dateFormat = SimpleDateFormat("dd.MM.yyyy", Locale.getDefault())

return dateFormat.format(time)

}

/\*\*

\* Adds simple text change listener

\* @author lidaamber

\*/

fun TextInputEditText.addTextChangedListener(listener: (CharSequence?) -> Unit) {

addTextChangedListener(object : TextWatcher {

override fun afterTextChanged(s: Editable?) {

}

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

listener(s)

}

})

}

/\*\*

\* Sets date picker to view

\* @author lidaamber

\*/

fun View.setDatePicker(context: Context, dateCallback: (Calendar) -> Unit) {

val calendar = Calendar.getInstance()

setOnClickListener({

val datePicker = DatePickerDialog(context,

{ \_, year, month, date ->

calendar.set(year, month, date)

dateCallback(calendar)

}, calendar.get(Calendar.YEAR),

calendar.get(Calendar.MONTH),

calendar.get(Calendar.DAY\_OF\_MONTH))

datePicker.datePicker.maxDate = Calendar.getInstance().timeInMillis

datePicker.show()

})

}

// VerticalTextView.kt

package com.lidaamber.kpisurvey.utils

import android.content.Context

import android.graphics.Canvas

import android.util.AttributeSet

import android.view.Gravity

import android.widget.LinearLayout

import android.widget.TextView

/\*\*

\* Vertical text view

\* @author lidaamber

\*/

class VerticalTextView(context: Context,

attrs: AttributeSet) : TextView(context, attrs) {

/\*\*

\* Text displaying state

\*/

private val topDown: Boolean

init {

val gravity = gravity

topDown = if (Gravity.isVertical(gravity) && gravity and

Gravity.VERTICAL\_GRAVITY\_MASK == Gravity.BOTTOM) {

setGravity(gravity and Gravity.HORIZONTAL\_GRAVITY\_MASK or Gravity.TOP)

false

} else {

true

}

}

override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {

super.onMeasure(heightMeasureSpec, widthMeasureSpec)

setMeasuredDimension(measuredHeight, measuredWidth)

}

override fun onDraw(canvas: Canvas) {

val textPaint = paint

textPaint.color = currentTextColor

textPaint.typeface = typeface

textPaint.drawableState = drawableState

canvas.save()

if (topDown) {

canvas.translate(width.toFloat(), 0f)

canvas.rotate(90f)

} else {

canvas.translate(0f, height.toFloat())

canvas.rotate(-90f)

}

canvas.translate(compoundPaddingLeft.toFloat(), extendedPaddingTop.toFloat())

layout.draw(canvas)

canvas.restore()

}

/\*\*

\* Sets text weight in layout

\*/

fun setWeight(value: Float) {

val p = layoutParams as LinearLayout.LayoutParams

p.weight = value

requestLayout()

}

/\*\*

\* Gets text weight in layout

\*/

fun getWeight(): Float {

return (layoutParams as LinearLayout.LayoutParams).weight

}

}

// AuthenticationView.kt

package com.lidaamber.kpisurvey.views

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

/\*\*

\* View for authentication flow

\* @author lidaamber

\*/

interface AuthenticationView : BaseView {

/\*\*

\* Shows registration screen

\*/

fun showRegistrationScreen()

/\*\*

\* Shows authentication questions

\*/

fun showQuestions(questions: List<AuthenticationQuestion>)

/\*\*

\* Shows login screen

\*/

fun showLoginScreen()

/\*\*

\* Shows authentication button

\*/

fun showAuthenticateButton()

/\*\*

\* Hides authentication button

\*/

fun hideAuthenticateButton()

}

// BaseView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* Base view

\* @author lidaamber

\*/

interface BaseView : NetworkAvailabilityView {

/\*\*

\* Displays error message

\*/

fun displayError(message: String)

}

// ConfirmationView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for token confirmation flow

\* @author lidaamber

\*/

interface ConfirmationView : BaseView {

/\*\*

\* Shows main screen

\*/

fun showMainScreen()

}

// LoginView.kt

package com.lidaamber.kpisurvey.views

import com.lidaamber.kpisurvey.model.AuthenticationQuestion

/\*\*

\* View for login flow

\* @author lidaamber

\*/

interface LoginView : BaseView {

/\*\*

\* Shows authentication questions

\*/

fun showQuestions(questions: List<AuthenticationQuestion>)

/\*\*

\* Shows main screen

\*/

fun showMainScreen()

/\*\*

\* Shows authentication screen

\*/

fun showAuthenticationScreen()

/\*\*

\* Shows submit button

\*/

fun showSubmitButton()

/\*\*

\* Hides submit button

\*/

fun hideSubmitButton()

/\*\*

\* Shows reset password screen

\*/

fun showResetPasswordScreen()

}

// NetworkAvailabilityView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for tracking network availability

\* @author lidaamber

\*/

interface NetworkAvailabilityView {

/\*\*

\* Shows not connected message

\*/

fun showNotConnectedMessage()

}

// NewPasswordView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for new password flow

\* @author lidaamber

\*/

interface NewPasswordView : BaseView {

/\*\*

\* Shows sign in

\*/

fun showSignIn()

}

// RegistrationView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for registration flow

\* @author lidaamber

\*/

interface RegistrationView : BaseView {

/\*\*

\* Shows confirmation screen

\*/

fun showConfirmationScreen()

}

// ResetPasswordEmailView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for reset password flow

\* @author lidaamber

\*/

interface ResetPasswordEmailView : BaseView {

/\*\*

\* Shows login screen

\*/

fun showLoginScreen()

/\*\*

\* Shows reset password screen

\*/

fun showResetPasswordTokenScreen()

/\*\*

\* Shows confirmation button

\*/

fun showConfirmButton()

/\*\*

\* Hides confirmation button

\*/

fun hideConfirmButton()

}

// ResetPasswordTokenView.kt

package com.lidaamber.kpisurvey.views

/\*\*

\* View for reset password flow

\* @author lidaamber

\*/

interface ResetPasswordTokenView : BaseView {

/\*\*

\* Shows reset email screen

\*/

fun showResetEmailScreen()

/\*\*

\* Shows new password setting screen

\*/

fun showNewPasswordScreen(token: String)

/\*\*

\* Shows confirmation button

\*/

fun showConfirmButton()

/\*\*

\* Hides confirmation button

\*/

fun hideConfirmButton()

}

// SurveysListView.kt

package com.lidaamber.kpisurvey.views

import com.lidaamber.kpisurvey.model.Survey

import com.lidaamber.kpisurvey.model.SurveyId

/\*\*

\* View for surveys list flow

\* @author lidaamber

\*/

interface SurveysListView : BaseView {

/\*\*

\* Shows surveys list

\*/

fun showSurveys(surveys: List<Survey>)

/\*\*

\* Shows survey screen

\*/

fun showSurveyScreen(surveyId: SurveyId, title: String)

}

// SurveyView.kt

package com.lidaamber.kpisurvey.views

import com.lidaamber.kpisurvey.model.Question

/\*\*

\* View for survey flow

\* @author lidaamber

\*/

interface SurveyView : BaseView {

/\*\*

\* Shows questions list

\*/

fun displayQuestions(questions: List<Question>)

/\*\*

\* Shows survey passed message

\*/

fun displaySurveyPassed()

}