File - C:\Users\Akhilesh\Downloads\BloodDonorCompanionApp-master\app\src\main\java\com\mczuba\VIT_20BCE7010\vm\InfoFragment.kt

```kotlin
package com.mczuba.VIT_20BCE7010.vm

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import com.google.android.material.button.MaterialButton
import com.mczuba.VIT_20BCE7010.R
import com.mczuba.VIT_20BCE7010.databinding.FragmentInfoBinding
import com.mczuba.VIT_20BCE7010.util.FoldableLayoutHelper

class InfoFragment : Fragment() {
    private val viewModel by lazy {
        ViewModelProvider(this).get(InfoViewModel::class.java)
    }
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val binding = FragmentInfoBinding.inflate(inflater, container, false)

        binding.lifecycleOwner = this
        binding.viewmodel = viewModel
```

```
28          binding.fragment = this
29
30          val buttonExpand = binding.root.findViewById<MaterialButton>(R.id.
    button_Expand)
31          val layoutExpand = binding.root.findViewById<View>(R.id.layout_details)
32          FoldableLayoutHelper(requireContext(), layoutExpand, buttonExpand, true)
33
34          val buttonExpand2 = binding.root.findViewById<MaterialButton>(R.id.
    button_Expand2)
35          val layoutExpand2 = binding.root.findViewById<View>(R.id.layoutDetails2)
36          FoldableLayoutHelper(requireContext(), layoutExpand2, buttonExpand2, true)
37
38          return binding.root
39      }
40
41      fun link()
42      {
43          val url = "https://krew.info/zapasy/"
44          val i = Intent(Intent.ACTION_VIEW)
45          i.data = Uri.parse(url)
46          startActivity(i)
47      }
48  }
```

```kotlin
package com.mczuba.VIT_20BCE7010.vm

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.mczuba.VIT_20BCE7010.R

class BlankFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_blank, container, false)
    }
}
```

File - C:\Users\Akhilesh\Downloads\BloodDonorCompanionApp-master\app\src\main\java\com\mczuba\VIT_20BCE7010\vm\InfoViewModel.kt

```kotlin
package com.mczuba.VIT_20BCE7010.vm

import android.app.Application
import androidx.lifecycle.AndroidViewModel

class InfoViewModel(application: Application): AndroidViewModel(application) {

}
```

File - C:\Users\Akhilesh\Downloads\BloodDonorCompanionApp-master\app\src\main\java\com\mczuba\VIT_20BCE7010\vm\SummaryFragment.kt

```kotlin
package com.mczuba.VIT_20BCE7010.vm

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.fragment.findNavController
import com.mczuba.VIT_20BCE7010.R
import com.mczuba.VIT_20BCE7010.databinding.FragmentSummaryBinding
import kotlinx.android.synthetic.main.fragment_summary.*

class SummaryFragment : Fragment() {
    private lateinit var viewModel: SummaryViewModel

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

        val binding = FragmentSummaryBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this
        viewModel = ViewModelProvider(this).get(SummaryViewModel::class.java)
        binding.viewmodel = viewModel
```

```
28          val navController = findNavController()
29
30          binding.btnScheduleDonation.setOnClickListener {
31              val direction = SummaryFragmentDirections.
    actionNavigationSummaryToScheduleRecordFragment()
32              navController.graph.startDestination = R.id.summaryFragment;
33              navController.navigate(direction)
34          }
35
36          viewModel.schedule.observe( viewLifecycleOwner, Observer() {
37              if(it.scheduleId == -1) {
38                  binding.textScheduleNone.visibility = View.VISIBLE
39                  binding.layoutScheduleDetails.visibility = View.GONE
40                  btnScheduleRemove.visibility = View.GONE
41              }
42              else {
43                  binding.textScheduleNone.visibility = View.GONE
44                  binding.layoutScheduleDetails.visibility = View.VISIBLE
45                  btnScheduleRemove.visibility = View.VISIBLE
46
47                  if (it.note == "")
48                      binding.viewScheduleNote.visibility = View.GONE
49                  else
50                      binding.viewScheduleNote.visibility = View.VISIBLE
51
52                  if (it.location == "")
53                      binding.viewScheduleLocation.visibility = View.GONE
```

```
54                  else
55                      binding.viewScheduleLocation.visibility = View.VISIBLE
56
57              }
58          })
59
60          return binding.root
61      }
62
63      override fun onResume() {
64          super.onResume()
65          viewModel?.updateData()
66      }
67  }
```

File - C:\Users\Akhilesh\Downloads\BloodDonorCompanionApp-master\app\src\main\java\com\mczuba\VIT_20BCE7010\vm\SummaryViewModel.kt

```kotlin
package com.mczuba.VIT_20BCE7010.vm

import android.app.Application
import androidx.databinding.Observable
import androidx.databinding.Observable.OnPropertyChangedCallback
import androidx.databinding.ObservableInt
import androidx.lifecycle.*
import com.mczuba.VIT_20BCE7010.R
import com.mczuba.VIT_20BCE7010.data.DonationSummary
import com.mczuba.VIT_20BCE7010.data.DonorRepository
import com.mczuba.VIT_20BCE7010.data.models.Donation
import com.mczuba.VIT_20BCE7010.data.models.Schedule
import com.mczuba.VIT_20BCE7010.data.models.User
import com.mczuba.VIT_20BCE7010.util.InjectorUtils
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.time.LocalDateTime
import java.time.ZoneOffset
import java.time.temporal.ChronoUnit
import java.util.*


class SummaryViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: DonorRepository = InjectorUtils.getUserRepository(
application)
    private var _user = MutableLiveData<User>()
    private lateinit var _totalSummary : DonationSummary
```

```
27      private lateinit var _wholeSummary : DonationSummary
28      private lateinit var _plasmaSummary : DonationSummary
29      private lateinit var _plateletsSummary : DonationSummary
30      private lateinit var _totalSummaryYear : DonationSummary
31      private lateinit var _wholeSummaryYear : DonationSummary
32      private lateinit var _plasmaSummaryYear : DonationSummary
33      private lateinit var _plateletsSummaryYear : DonationSummary
34
35      private val _currentSummary = MutableLiveData<DonationSummary>(DonationSummary.
    Factory.getEmptyDonationSummary(
36          Donation.DonationType.WHOLE))
37      private val _yearlySummary = MutableLiveData<DonationSummary>(DonationSummary.
    Factory.getEmptyDonationSummary(
38          Donation.DonationType.WHOLE))
39      private var _daysLeftWhole = 0
40      private var _daysLeftPlatelets = 0
41      private var _daysLeftPlasma = 0
42      private val _daysLeft = MutableLiveData(0)
43
44      val user: LiveData<User> = _user
45      var currentSummary: LiveData<DonationSummary> = _currentSummary;
46      var yearlySummary: LiveData<DonationSummary> = _yearlySummary;
47      val checkedBtnObs = ObservableInt(R.id.summary_chipgroup)
48      val daysLeft: LiveData<Int> = _daysLeft
49
50      fun updateData() {
51          viewModelScope.launch {
```

```
52              val users = repository.readAllUsers()
53              val user = users.first()
54              _user.postValue(user)
55
56              val donations = repository.readUserDonations(user.userId)
57              val latest = donations.sortedBy { x -> x.date }.firstOrNull()
58
59              _wholeSummary = DonationSummary.getTypeSummary(donations, Donation.
    DonationType.WHOLE)
60              _plasmaSummary = DonationSummary.getTypeSummary(donations, Donation.
    DonationType.PLASMA)
61              _plateletsSummary = DonationSummary.getTypeSummary(donations, Donation.
    DonationType.PLATELETS)
62
63              val yearDate = Date(Date().year, 0, 1)
64              _wholeSummaryYear = DonationSummary.getTypeSummarySinceDate(donations,
    Donation.DonationType.WHOLE, yearDate)
65              _plasmaSummaryYear = DonationSummary.getTypeSummarySinceDate(donations,
    Donation.DonationType.PLASMA, yearDate)
66              _plateletsSummaryYear = DonationSummary.getTypeSummarySinceDate(
    donations, Donation.DonationType.PLATELETS, yearDate)
67              _totalSummaryYear = DonationSummary.getTotalSummarySinceDate(donations,
    yearDate)
68
69              _totalSummary = DonationSummary.getTotalSummary(donations)
70
71              latest?.run {
```

```
72                  val today = LocalDateTime.now()
73                  val thenWhole = repository.getNextDonationDate(user.userId, Donation
   .DonationType.WHOLE).toInstant().atZone(ZoneOffset.ofHours(0)).toLocalDateTime()
74                  val thenPlasm = repository.getNextDonationDate(user.userId, Donation
   .DonationType.PLASMA).toInstant().atZone(ZoneOffset.ofHours(0)).toLocalDateTime()
75                  val thenPlate = repository.getNextDonationDate(user.userId, Donation
   .DonationType.PLATELETS).toInstant().atZone(ZoneOffset.ofHours(0)).toLocalDateTime()
76
77
78                  _daysLeftWhole = ChronoUnit.DAYS.between(today, thenWhole).toInt()+1
79                  _daysLeftPlasma = ChronoUnit.DAYS.between(today, thenPlasm).toInt()+
   1
80                  _daysLeftPlatelets = ChronoUnit.DAYS.between(today, thenPlate).toInt
   ()+1
81              }
82
83              getSchedule()
84
85              _currentSummary.postValue(_totalSummary)
86              _yearlySummary.postValue(_totalSummaryYear)
87              checkedBtnObs.set(R.id.summary_chiptotal)
88          }
89
90          checkedBtnObs.addOnPropertyChangedCallback(object :
   OnPropertyChangedCallback() {
91              override fun onPropertyChanged(sender: Observable, propertyId: Int) {
92                  val newSelection = when(checkedBtnObs.get())
```

```
 93                  {
 94                      R.id.summary_chipwhole -> _wholeSummary;
 95                      R.id.summary_chipplasma -> _plasmaSummary;
 96                      R.id.summary_chipplatelets -> _plateletsSummary;
 97                      else -> _totalSummary;
 98                  }
 99                  _currentSummary.postValue(newSelection)
100                  val newYearSelection = when(checkedBtnObs.get())
101                  {
102                      R.id.summary_chipwhole -> _wholeSummaryYear;
103                      R.id.summary_chipplasma -> _plasmaSummaryYear;
104                      R.id.summary_chipplatelets -> _plateletsSummaryYear;
105                      else -> _totalSummaryYear;
106                  }
107                  _yearlySummary.postValue(newYearSelection)
108                  _daysLeft.postValue(
109                      when(checkedBtnObs.get()) {
110                          R.id.summary_chipwhole -> _daysLeftWhole;
111                          R.id.summary_chipplasma -> _daysLeftPlasma;
112                          R.id.summary_chipplatelets -> _daysLeftPlatelets;
113                          else -> maxOf(_daysLeftWhole, _daysLeftPlasma,
    _daysLeftPlatelets);
114                      }
115                  )
116              }
117          })
118      }
```

```
119
120     private val _schedule = MutableLiveData<Schedule>(getDefaultSchedule())
121     private val _daysTo = MutableLiveData<Int>(0)
122
123     val schedule: LiveData<Schedule> = _schedule
124     val daysTo: LiveData<Int> = _daysTo
125
126     val formattedDate = Transformations.map(schedule) {
127         SimpleDateFormat("EE, d MMM yyy").format(it.date)
128     }
129
130     val scheduleType = Transformations.map(schedule) {
131         when(it.type) {
132             Donation.DonationType.WHOLE -> application.resources.getString(R.string
    .donation_whole)
133             Donation.DonationType.PLASMA -> application.resources.getString(R.
    string.donation_plasma)
134             Donation.DonationType.PLATELETS -> application.resources.getString(R.
    string.donation_platelets)
135             else -> application.resources.getString(R.string.donation_whole)
136         }
137     }
138
139     fun removeSchedule() {
140         val scheduleToRemove = _schedule.value
141         _schedule.postValue(getDefaultSchedule())
142         _daysTo.postValue(0)
```

```
143
144         viewModelScope.launch {
145             scheduleToRemove?.run { repository.removeSchedule(this) }
146             getSchedule()
147         }
148     }
149
150     suspend fun getSchedule() {
151         repository.readUserSchedules(0).sortedBy { x -> x.date }.firstOrNull()?.run
     {
152             val then = this.date.toInstant().atZone(ZoneOffset.ofHours(0)).
    toLocalDateTime()
153             _daysTo.postValue(ChronoUnit.DAYS.between(LocalDateTime.now(), then).
    toInt()+1)
154
155             _schedule.postValue(this)
156         }
157     }
158
159     private fun getDefaultSchedule(): Schedule {
160         return Schedule(
161             -1,
162             0,
163             Donation.DonationType.WHOLE,
164             Date(),
165             "",
166             "",
```

```
167                 Schedule.NotificationSetting.THREE_DAYS_BEFORE
168             )
169         }
170 }
```

```
 1 package com.mczuba.VIT_20BCE7010.vm.editors
 2
 3 import android.app.DatePickerDialog
 4 import android.os.Bundle
 5 import android.view.LayoutInflater
 6 import android.view.View
 7 import android.view.ViewGroup
 8 import android.widget.ArrayAdapter
 9 import android.widget.DatePicker
10 import androidx.databinding.Observable
11 import androidx.databinding.Observable.OnPropertyChangedCallback
12 import androidx.fragment.app.DialogFragment
13 import androidx.fragment.app.Fragment
14 import androidx.fragment.app.FragmentTransaction
15 import androidx.lifecycle.Observer
16 import androidx.lifecycle.ViewModelProvider
17 import androidx.navigation.Navigation
18 import androidx.navigation.fragment.navArgs
19 import com.google.android.material.button.MaterialButton
20 import com.mczuba.VIT_20BCE7010.BR
21 import com.mczuba.VIT_20BCE7010.R
22 import com.mczuba.VIT_20BCE7010.data.models.Donation
23 import com.mczuba.VIT_20BCE7010.databinding.FragmentRecordNewBinding
24 import com.mczuba.VIT_20BCE7010.util.DatePickerDialogFragment
25 import com.mczuba.VIT_20BCE7010.util.FoldableLayoutHelper
26 import java.util.*
27
```

```
28
29 class NewRecordFragment : Fragment(), DatePickerDialog.OnDateSetListener {
30     private val args: NewRecordFragmentArgs by navArgs()
31     var detailsExpanded: Boolean = false;
32     private val viewModel by lazy {
33         ViewModelProvider(this).get(NewRecordViewModel::class.java)
34     }
35
36     override fun onCreateView(
37         inflater: LayoutInflater, container: ViewGroup?,
38         savedInstanceState: Bundle?
39     ): View? {
40         val binding = FragmentRecordNewBinding.inflate(inflater, container, false)
41         binding.lifecycleOwner = this
42         binding.viewmodel = viewModel
43
44         val buttonExpand = binding.root.findViewById<MaterialButton>(R.id.
   button_Expand)
45         val layoutExpand = binding.root.findViewById<View>(R.id.layout_details)
46         FoldableLayoutHelper(requireContext(), layoutExpand, buttonExpand, true)
47
48         if (args.donationArgument>=0) {
49             viewModel.setupEditMode(args.donationArgument)
50             binding.newrecordSubmit.text = getString(R.string.donation_submitedit)
51             binding.newrecordTitle.text = getString(R.string.donation_edit)
52         }
53
```

```
54          binding.editDonationType.setAdapter(
55              ArrayAdapter(
56                  requireContext(),
57                  R.layout.support_simple_spinner_dropdown_item,
58                  requireContext().resources.getStringArray(
59                      R.array.donation_array
60                  )
61              )
62          )
63          binding.editDonationType.setOnItemClickListener { parent, view, position, id
    ->
64              run {
65                  viewModel.setDonationType(
66                      when (position) {
67                          0 -> Donation.DonationType.WHOLE
68                          1 -> Donation.DonationType.PLASMA
69                          2 -> Donation.DonationType.PLATELETS
70                          else -> Donation.DonationType.DISQUALIFIED
71                      }
72                  ) }
73          }
74          binding.editDonationType.setText(
75              binding.editDonationType.adapter.getItem(0).toString(),
76              false
77          );
78
79          viewModel.donation.addOnPropertyChangedCallback(object :
```

```
 80              OnPropertyChangedCallback() {
 81              override fun onPropertyChanged(sender: Observable?, propertyId: Int) {
 82                  when(propertyId) {
 83                      BR.type-> {
 84                          binding.editDonationType.setText(
 85                              resources.getStringArray(R.array.donation_array)[
    viewModel.donation.getType()!!.ordinal],
 86                              false
 87                          );
 88                      }
 89                      BR.arm -> {
 90                          binding.editUsedArm.setText(
 91                              resources.getStringArray(R.array.arm_array)[viewModel.
    donation.getArm()!!.ordinal],
 92                              false
 93                          );
 94                      }
 95                  }
 96
 97              }
 98          })
 99
100          binding.editUsedArm.setAdapter(
101              ArrayAdapter(
102                  requireContext(),
103                  R.layout.support_simple_spinner_dropdown_item,
104                  requireContext().resources.getStringArray(
```

```
105                     R.array.arm_array
106                 )
107             )
108         )
109         binding.editUsedArm.setText(binding.editUsedArm.adapter.getItem(2).toString
    (), false);
110         binding.editUsedArm.setOnItemClickListener { parent, view, position, id ->
111             run {
112                 viewModel.setArmType(
113                     when (position) {
114                         0 -> Donation.ArmType.RIGHT
115                         1 -> Donation.ArmType.LEFT
116                         else -> Donation.ArmType.UNKNOWN
117                     }
118                 ) }
119         }
120
121         viewModel.completeState.observe(viewLifecycleOwner, Observer {
122             if (it == true) {
123                 Navigation.findNavController(binding.root).popBackStack()
124             }
125         })
126
127         binding.newrecordDate.setOnClickListener {
128             val ft: FragmentTransaction = requireFragmentManager().beginTransaction
    ()
129             val newFragment: DialogFragment = DatePickerDialogFragment(
```

```
130                     this,
131                     viewModel.donation.getDate() ?: Date(),
132                     viewModel.getMinTime(),
133                     viewModel.getMaxTime()
134                 )
135                 newFragment.show(ft, "dialog")
136             }
137 
138             return binding.root
139         }
140 
141         override fun onDateSet(view: DatePicker?, year: Int, month: Int, dayOfMonth:
    Int) {
142             viewModel.setNewDateTime(year, month, dayOfMonth)
143         }
144 }
145 
```

```kotlin
package com.mczuba.VIT_20BCE7010.vm.editors

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.Transformations
import androidx.lifecycle.viewModelScope
import com.hadilq.liveevent.LiveEvent
import com.mczuba.VIT_20BCE7010.data.DonationLiveData
import com.mczuba.VIT_20BCE7010.data.DonorRepository
import com.mczuba.VIT_20BCE7010.data.models.Donation
import com.mczuba.VIT_20BCE7010.util.InjectorUtils
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.time.ZoneOffset
import java.util.*

class NewRecordViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: DonorRepository = InjectorUtils.getUserRepository(
application)

    private val _completeState = LiveEvent<Boolean>()

    val completeState: LiveData<Boolean> = _completeState
    val donation = DonationLiveData(getDefaultDonation())
    var editMode = false

```

```
27      fun setupEditMode(donationId: Int) {
28          viewModelScope.launch {
29              editMode = true;
30              repository.readUserDonations(0).find { it.donationId == donationId }
31                  ?.let { donation.setDonation(it) }
32          }
33      }
34
35      fun setDonationType(type: Donation.DonationType) {
36          donation.setType(type)
37          donation.setAmount(
38              when (type) {
39                  Donation.DonationType.WHOLE -> 450
40                  Donation.DonationType.PLASMA -> 600
41                  Donation.DonationType.PLATELETS -> 500
42                  else -> 0
43              }
44          )
45      }
46
47      fun setArmType(type: Donation.ArmType) {
48          donation.setArm(type)
49      }
50
51      fun submit() {
52          viewModelScope.launch {
53              val donations = repository.readUserDonations(0)
```

```
54
55              if (editMode)
56                  repository.updateDonation(donation.getDonation())
57              else
58                  repository.addDonation(donation.getDonation())
59
60              _completeState.value = true
61          }
62      }
63
64      val formattedAmount = Transformations.map(donation.lamount) {
65          if (it > 0)
66              it.toString()
67          else
68              "-"
69      }
70
71      val formattedDate = Transformations.map(donation.ldate) {
72          SimpleDateFormat("EE, d MMM yyy").format(it)
73      }
74
75      fun setNewDateTime(year: Int, month: Int, dayOfMonth: Int) {
76          donation.setDate(Date(year - 1900, month, dayOfMonth))
77      }
78
79      fun getMinTime(): Date {
80          var date = repository.user.value!!.birthDay!!.toInstant().atZone(ZoneOffset.
```

```
 80 ofHours(0)).toLocalDateTime()
 81         date = date.plusYears(18)
 82         return Date.from(date.atZone(ZoneOffset.ofHours(0)).toInstant())
 83     }
 84
 85     fun getMaxTime(): Date = Date()
 86
 87
 88     private fun getDefaultDonation(): Donation {
 89         return Donation(
 90             0,
 91             0,
 92             Donation.DonationType.WHOLE,
 93             450,
 94             Date(),
 95             "",
 96             "",
 97             Donation.ArmType.UNKNOWN,
 98             null,
 99             null,
100             null,
101             null
102         )
103     }
104 }
```

```kotlin
package com.mczuba.VIT_20BCE7010.vm.editors

import android.app.DatePickerDialog
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.DatePicker
import androidx.databinding.Observable
import androidx.databinding.Observable.OnPropertyChangedCallback
import androidx.fragment.app.DialogFragment
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentTransaction
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.Navigation
import com.mczuba.VIT_20BCE7010.BR
import com.mczuba.VIT_20BCE7010.R
import com.mczuba.VIT_20BCE7010.data.models.Donation
import com.mczuba.VIT_20BCE7010.data.models.Schedule
import com.mczuba.VIT_20BCE7010.databinding.FragmentRecordScheduleBinding
import com.mczuba.VIT_20BCE7010.util.DatePickerDialogFragment
import java.util.*


class ScheduleRecordFragment : Fragment(), DatePickerDialog.OnDateSetListener {
```

```
28
29      private val viewModel by lazy {
30          ViewModelProvider(this).get(ScheduleRecordViewModel::class.java)
31      }
32
33      override fun onCreateView(
34          inflater: LayoutInflater, container: ViewGroup?,
35          savedInstanceState: Bundle?
36      ): View? {
37          val binding = FragmentRecordScheduleBinding.inflate(inflater, container,
    false)
38          binding.lifecycleOwner = this
39          binding.viewmodel = viewModel
40
41          binding.editDonationType.setAdapter(
42              ArrayAdapter(
43                  requireContext(),
44                  R.layout.support_simple_spinner_dropdown_item,
45                  requireContext().resources.getStringArray(
46                      R.array.donation_array
47                  ).dropLast(1)
48              )
49          )
50          binding.editDonationType.setOnItemClickListener { parent, view, position, id
     ->
51              run {
52                  viewModel.setDonationType(
```

```
53                      when (position) {
54                          0 -> Donation.DonationType.WHOLE
55                          1 -> Donation.DonationType.PLASMA
56                          2 -> Donation.DonationType.PLATELETS
57                          else -> Donation.DonationType.DISQUALIFIED
58                      }
59                  ) }
60          }
61          binding.editDonationType.setText(
62              binding.editDonationType.adapter.getItem(0).toString(),
63              false
64          );
65
66          binding.editNotification.setAdapter(
67              ArrayAdapter(
68                  requireContext(),
69                  R.layout.support_simple_spinner_dropdown_item,
70                  requireContext().resources.getStringArray(
71                      R.array.notification_array
72                  )
73              )
74          )
75          binding.editNotification.setOnItemClickListener { parent, view, position, id
    ->
76              run {
77                  viewModel.setNotification(
78                      when (position) {
```

```
 79                          0 -> Schedule.NotificationSetting.WEEK_BEFORE
 80                          1 -> Schedule.NotificationSetting.THREE_DAYS_BEFORE
 81                          2 -> Schedule.NotificationSetting.DAY_BEFORE
 82                          else -> Schedule.NotificationSetting.NO_NOTIFICATION
 83                      }
 84                  ) }
 85          }
 86          binding.editNotification.setText(
 87              binding.editNotification.adapter.getItem(3).toString(),
 88              false
 89          );
 90
 91          viewModel.schedule.addOnPropertyChangedCallback(object :
 92              OnPropertyChangedCallback() {
 93              override fun onPropertyChanged(sender: Observable?, propertyId: Int) {
 94                  when(propertyId) {
 95                      BR.type-> {
 96                          binding.editDonationType.setText(
 97                              resources.getStringArray(R.array.donation_array)[
    viewModel.schedule.getType()!!.ordinal],
 98                              false
 99                          );
100                      }
101                      BR.notification-> {
102                          binding.editNotification.setText(
103                              resources.getStringArray(R.array.notification_array)[
    viewModel.schedule.getNotification()!!.ordinal],
```

```
104                                 false
105                             );
106                         }
107                     }
108                 }
109             })
110
111             viewModel.completeState.observe(viewLifecycleOwner, Observer {
112                 if (it == true) {
113                     Navigation.findNavController(binding.root).popBackStack()
114                 }
115             })
116
117             binding.openDatePicker.setOnClickListener {
118                 val ft: FragmentTransaction = requireFragmentManager().beginTransaction
    ()
119                 val newFragment: DialogFragment = DatePickerDialogFragment(
120                     this,
121                     viewModel.schedule.getDate() ?: Date(),
122                     viewModel.getMinTime(),
123                     viewModel.getMaxTime()
124                 )
125                 newFragment.show(ft, "dialog")
126             }
127
128             return binding.root
129         }
```

```
130
131     override fun onDateSet(view: DatePicker?, year: Int, month: Int, dayOfMonth:
    Int) {
132         viewModel.setNewDateTime(year, month, dayOfMonth)
133     }
134 }
135
```

```kotlin
package com.mczuba.VIT_20BCE7010.vm.editors

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.viewModelScope
import com.hadilq.liveevent.LiveEvent
import com.mczuba.VIT_20BCE7010.data.ScheduleLiveData
import com.mczuba.VIT_20BCE7010.data.DonorRepository
import com.mczuba.VIT_20BCE7010.data.models.Donation
import com.mczuba.VIT_20BCE7010.data.models.Schedule
import com.mczuba.VIT_20BCE7010.util.InjectorUtils
import kotlinx.coroutines.launch
import java.time.ZoneOffset
import java.util.*

class ScheduleRecordViewModel(application: Application) : AndroidViewModel(application) {
    private val repository: DonorRepository = InjectorUtils.getUserRepository(application)

    private val _completeState = LiveEvent<Boolean>()

    val completeState: LiveData<Boolean> = _completeState
    val schedule = ScheduleLiveData(getDefaultSchedule())

    fun setDonationType(type: Donation.DonationType) {
```

```
26          schedule.setType(type)
27          updateMinDate(type)
28      }
29
30      fun setNotification(type: Schedule.NotificationSetting) {
31          schedule.setNotification(type)
32      }
33
34      fun submit() {
35          viewModelScope.launch {
36              val schedule = schedule.getSchedule()
37              repository.addSchedule(schedule)
38
39              _completeState.value = true
40          }
41      }
42
43      fun setNewDateTime(year: Int, month: Int, dayOfMonth: Int) {
44          schedule.setDate(Date(year - 1900, month, dayOfMonth))
45      }
46
47      private var minimumDate = Date();
48      fun getMinTime(): Date = minimumDate
49
50      fun getMaxTime(): Date {
51          var date = minimumDate.toInstant().atZone(ZoneOffset.ofHours(0)).
    toLocalDateTime()
```

```
52          date = date.plusMonths(12)
53          return Date.from(date.atZone(ZoneOffset.ofHours(0)).toInstant())
54      }
55
56      init {
57          updateMinDate(Donation.DonationType.WHOLE)
58      }
59
60      fun updateMinDate(type: Donation.DonationType) {
61          viewModelScope.launch {
62              minimumDate = repository.getNextDonationDate(0, type)
63              if(schedule.getDate()?.before(minimumDate) ?: true)
64                  schedule.setDate(minimumDate)
65          }
66      }
67
68      init {
69          viewModelScope.launch {
70              val latestSchedule = repository.readUserSchedules(0).sortedBy { x -> x.
    date }.firstOrNull()
71              if(latestSchedule!=null) {
72                  schedule.setSchedule(latestSchedule)
73              }
74          }
75      }
76
77      private fun getDefaultSchedule(): Schedule {
```

```
78          return Schedule(
79              0,
80              0,
81              Donation.DonationType.WHOLE,
82              Date(),
83              "",
84              "",
85              Schedule.NotificationSetting.THREE_DAYS_BEFORE
86          )
87      }
88  }
```