ДОДАТОК А

Графічний матеріал атестаційної роботи

Харківський національний університет радіоелектроніки
кафедра ЕОМ

Програмні методи розпізнавання стилю водіння транспортним засобом

Атестаційна робота
Другий рівень (магістр)

Автор
Тулей О.Ю.
ст. гр. КСМзм-19-1

Керівник
доц. Шматко О.В.

МЕТА АТЕСТАЦІЙНОЇ РОБОТИ

- розробка підходу до аналізу й оцінки стилю водіння водія при управлінні транспортним засобом з використанням сенсорів смартфона, орієнтованого на застосування в системах підвищення безпеки водія.

ЗАДАЧІ АТЕСТАЦІЙНОЇ РОБОТИ

- розробка інформаційної моделі поведінки водія;
- аналіз існуючих проектів і досліджень по розробці систем забезпечення безпеки і підтримки водіїв;
- розробка алгоритму оцінки стилю водіння водія ТЗ;
- аналіз отриманих результатів.

3

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Згідно зі статистикою Патрульній поліції України понад 80% дорожньо-транспортних пригод відбуваються з вини самих водіїв. Фактори ризику, що відповідають психофізіологічної складової поведінки водія, роблять значний вплив на виникнення дорожньо-транспортних пригод. До таких факторів підвищеного ризику настання дорожньо-транспортної пригоди можна віднести втому і ослаблену увагу водія, які проявляються в ситуаціях, в яких водії часто самі не усвідомлюють наступ ознак того чи іншого небезпечного стану.

Одним з варіантів підвищення безпеки водія є використання систем активної безпеки, спрямованих на запобігання аварійних ситуацій на основі моніторингу поведінки водія та своєчасного його оповіщення про поточну ситуацію за рахунок генерації йому контекстно-орієнтованих рекомендацій.

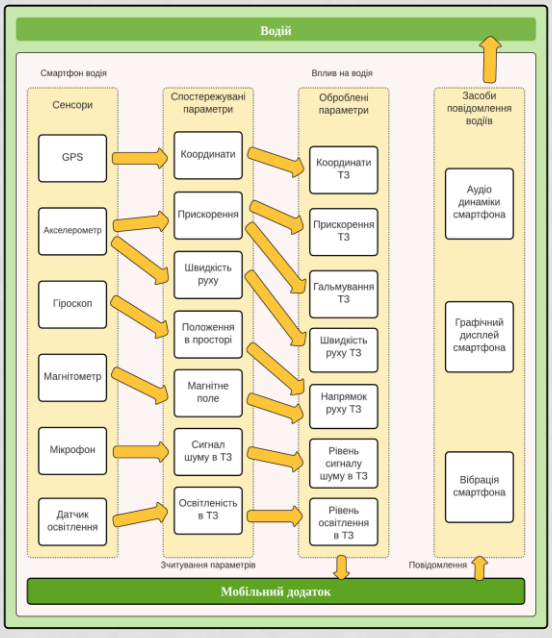
4

ПОНЯТТЯ СИСТЕМИ АКТИВНОЇ БЕЗПЕКИ

Безпека на дорозі визначається не тільки дорожніми умовами, технічним станом транспортних засобів та дотриманням правил дорожнього руху, а й навичками, фізичним станом, здатністю до концентрації та дотриманням водіями заходів безпеки.

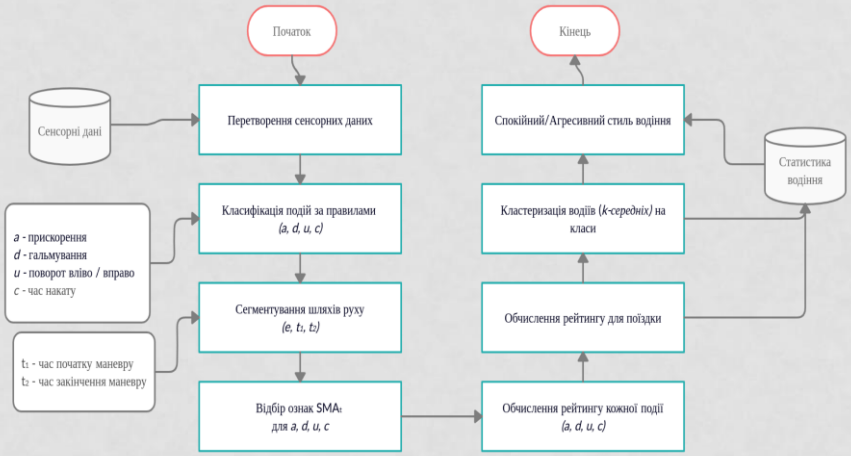
При високопріоритетних попередженнях водія схема роботи системи активної безпеки в загальному випадку може бути описана такою послідовністю команд «сприйняття – реагування»: виявлення можливості Зіткнення ТЗ, система виводить інформацію про можливе зіткнення при відсутності належної реакції у водія ТЗ, оповіщення про аварійну ситуацію за допомогою попереджувального сигналу і вироблення рекомендації щодо запобігання настання ДТП, привернення уваги водія, усвідомлення (ідентифікація) водієм відбувається аварійної ситуації, вибір рішення, реагування і прийняття водієм заходів щодо запобігання ДТП. Можна виділити найбільш популярні технології, які складають сучасні системи допомоги водієві: технологія контролю сліпих зон (СКСЗ); технологія попередження про схід зі смуги (СПСС), що обчислює час до перетину розмітки і попереджає водія у разі виявлення догляду; технологія виявлення пішоходів і велосипедистів (СВПВ); технологія розпізнавання дорожніх знаків (СРДЗ); технологія попередження про фронтальному зіткненні і пом'якшення наслідків аварії (СПФС); технологія контролю дотримання безпечної дистанції (СКБД).

5



6

РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ.



7

РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ.
ОБРАНІ ТЕХНОЛОГІЇ ДЛЯ РЕАЛІЗАЦІЇ ДОДАТКУ

Java™

MySQL®

8

ВИСНОВКИ

Розроблений алгоритм аналізу стилю керування транспортним засобом індивідуально для водія, що дозволяє підвищити якість розпізнавання небезпечних станів у поведінці водія під час водіння і враховує попередній досвід використання системи, а також надає можливість оцінити стиль водіння кожного водія окремо, тим самим звернувши його увагу на навички керування транспортним засобом; розроблений прототип системи оцінки стилю водіння водія при управлінні транспортним засобом на основі обробки даних з сенсорів смартфона за рахунок виділення зв'язків між ними і характеристиками керування транспортним засобом, а також аналізу історії взаємодії водія з системою попередження аварійних ситуацій.

Отримані результати дозволять підвищити точність розпізнавання небезпечних станів і, як наслідок, ефективність системи запобігання аварійним ситуаціям, а також знайдуть застосування при формуванні та відображенні звітів для представників страхових компаній, адміністраторів автопарків та керівників логістичних компаній, які здійснюють спостереження і контроль за статистикою скоєння поїздок водіями зі штату автопарку.

9

ДЯКУЮ ЗА УВАГУ!

10

ДОДАТОК Б

Лістнінг програми

```
package davydov.dmytro.breweries_list.base

import android.app.Activity
import android.util.TypedValue
import android.view.WindowManager
import androidx.annotation.ColorRes
import androidx.core.content.ContextCompat
import davydov.dmytro.breweries_list.R

fun Activity.changeStatusBarColor(@ColorRes color: Int) {
    window.run {
        clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS)

addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
        statusBarColor                                                    =
ContextCompat.getColor(this@changeStatusBarColor, color)
    }
}

fun Activity.changeStatusBarColorToPrimaryDark() {
    val primaryDarkColor = TypedValue()
    theme.resolveAttribute(R.attr.colorPrimaryDark,    primaryDarkColor,
true)

    window.run {

clearFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
        statusBarColor = primaryDarkColor.data
    }
}

package davydov.dmytro.breweries_list.base

import davydov.dmytro.breweries_list.root.network.NetworkError
import io.reactivex.Completable
import io.reactivex.Flowable
import io.reactivex.Single
import io.reactivex.schedulers.Schedulers
import javax.inject.Inject

class     CacheFirstLoader     @Inject     constructor(private     val
connectionStateService: ConnectionStateService) {

    fun <T> loadData(
        localSingle: Single<T>,
        remoteSingle: Single<T>,
        updateLocalFunc: (T) -> Completable
    ): Flowable<T> {
        return Flowable
            .concat(
                localSingle.toFlowable(),
                remoteSingle
                    .flatMapPublisher { remoteData ->
                        updateLocalFunc(remoteData)
                            .onErrorComplete()
```

```
                                .andThen(Flowable.just(remoteData))
                        }
                        .retryWhen { errorFlowable ->
                            errorFlowable
                                .flatMap { error ->
                                    if (error is NetworkError) {
                                        connectionStateService
                                            .trackConnectionState()
                                            .observeOn(Schedulers.io())
                                            .filter { it }
                                    } else {
                                        Flowable.error(error)
                                    }
                                }
                                .onErrorResumeNext { _: Throwable ->
Flowable.empty() }
                        }
                )
                .subscribeOn(Schedulers.io())
        }
    }

    package davydov.dmytro.breweries_list.base

    import android.content.BroadcastReceiver
    import android.content.Context
    import android.content.Intent
    import android.content.IntentFilter
    import android.net.ConnectivityManager
    import io.reactivex.Flowable
    import io.reactivex.processors.BehaviorProcessor
    import java.io.Closeable

    class ConnectionStateService private constructor(private val context:
Context) : Closeable {

        private val connectivityManager =
context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

        private val receiver = object: BroadcastReceiver() {
            override fun onReceive(context: Context?, intent: Intent?) {
                val networkInfo = connectivityManager.activeNetworkInfo
                connectionStateProcessor.onNext(networkInfo?.isConnected ?:
false)
            }
        }

        private val connectionStateProcessor =
BehaviorProcessor.create<Boolean>()

        override fun close() {
            context.unregisterReceiver(receiver)
        }

        fun trackConnectionState(): Flowable<Boolean> =
connectionStateProcessor.distinctUntilChanged()

        companion object {
            fun create(context: Context): ConnectionStateService {
                val service = ConnectionStateService(context)

                context.registerReceiver(service.receiver, INTENT_FILTER)

                return service
```

```
            }

            private             val             INTENT_FILTER             =
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION)
        }
    }
    package davydov.dmytro.breweries_list.base

    import android.content.res.Resources
    import kotlin.math.roundToInt

    fun dpToPx(dp: Int): Int {
        return dpToPxFloat(dp).roundToInt()
    }

    fun dpToPxFloat(dp: Int): Float {
        return (dp * Resources.getSystem().displayMetrics.scaledDensity)
    }
    package davydov.dmytro.breweries_list.base

    import androidx.lifecycle.ViewModel
    import androidx.lifecycle.ViewModelProvider
    import javax.inject.Inject
    import javax.inject.Provider

    class ViewModelFactory<VM : ViewModel> @Inject constructor(private val
viewModelProvider: Provider<VM>) :
        ViewModelProvider.Factory {
        @Suppress("UNCHECKED_CAST")
        override fun <T : ViewModel> create(modelClass: Class<T>): T {
            return viewModelProvider.get() as T
        }
    }
    package davydov.dmytro.breweries_list.breweries.logic

    import androidx.lifecycle.LiveData
    import androidx.lifecycle.MutableLiveData
    import androidx.lifecycle.ViewModel
    import davydov.dmytro.breweries_list.base.storeTo
    import davydov.dmytro.breweries_list.breweries.LocationOnMapPresenter
    import davydov.dmytro.breweries_list.breweries.data.BreweriesService
    import io.reactivex.Flowable
    import io.reactivex.android.schedulers.AndroidSchedulers
    import io.reactivex.internal.disposables.SequentialDisposable
    import io.reactivex.processors.PublishProcessor
    import java.util.concurrent.TimeUnit
    import javax.inject.Inject

    class BreweriesViewModel @Inject constructor(
        private val breweriesService: BreweriesService,
        private val locationOnMapPresenter: LocationOnMapPresenter
    ) : ViewModel() {

        private val breweriesDisposable = SequentialDisposable()

        private val _breweriesState = MutableLiveData<List<Brewery>>()
        val breweriesState: LiveData<List<Brewery>> = _breweriesState

        private val queryProcessor = PublishProcessor.create<String>()

        fun onViewCreated() {
            queryProcessor
                .debounce(DEBOUNCE_TIME, TimeUnit.MILLISECONDS)
                .distinctUntilChanged()
```

```
                .switchMap { query ->
                    val request = if (query.isEmpty()) {
                        breweriesService.getBreweries()
                    } else {
                        breweriesService.searchBreweriesBy(query)
                    }

                    request.onErrorResumeNext    {    _:    Throwable    ->
Flowable.never() }
                }
                .distinctUntilChanged()
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(
                    { newData -> _breweriesState.value = newData },
                    {}
                )
                .storeTo(breweriesDisposable)

            onBrewerySearchQueryChanged("")
        }

        fun onBrewerySearchQueryChanged(query: String) {
            queryProcessor.offer(query)
        }

        fun onBreweryOnMapClicked(location: Location) {
            locationOnMapPresenter.showLocationOnMap(location)
        }

        override fun onCleared() {
            super.onCleared()
            breweriesDisposable.update(null)
        }

        companion object {
            private const val DEBOUNCE_TIME = 300L
        }
    }
    package davydov.dmytro.breweries_list.breweries.logic

    data class Brewery(
        val id: Int,
        val name: String,
        val street: String?,
        val city: String?,
        val state: String?,
        val postalCode: String?,
        val country: String?,
        val location: Location?,
        val phone: String?,
        val websiteUrl: String?
    )

    package davydov.dmytro.breweries_list.breweries.ui

    import android.graphics.Rect
    import android.os.Bundle
    import android.text.SpannableString
    import android.text.Spanned
    import android.text.TextUtils
    import android.text.style.URLSpan
    import android.view.LayoutInflater
    import android.view.View
    import android.view.ViewGroup
```

```
import android.view.WindowManager
import androidx.core.widget.doOnTextChanged
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.observe
import androidx.recyclerview.widget.RecyclerView
import dagger.android.support.DaggerFragment
import davydov.dmytro.breweries_list.R
import davydov.dmytro.breweries_list.base.ViewModelFactory
import davydov.dmytro.breweries_list.base.changeStatusBarColor
import
davydov.dmytro.breweries_list.base.changeStatusBarColorToPrimaryDark
import davydov.dmytro.breweries_list.base.dpToPx
import davydov.dmytro.breweries_list.base.dpToPxFloat
import davydov.dmytro.breweries_list.base.showKeyboard
import davydov.dmytro.breweries_list.breweries.logic.BreweriesViewModel
import davydov.dmytro.breweries_list.breweries.logic.Brewery
import kotlinx.android.synthetic.main.fragment_breweries.*
import javax.inject.Inject

class BreweriesFragment : DaggerFragment() {

    @Inject
    lateinit var viewModelFactory: ViewModelFactory<BreweriesViewModel>

    private lateinit var viewModel: BreweriesViewModel

    private lateinit var showKeyboardRunnable: Runnable

    private val titles = arrayOf(
        R.string.country_title,
        R.string.state_title,
        R.string.city_title,
        R.string.postal_code_title,
        R.string.street_title,
        R.string.phone_title,
        R.string.website_title
    )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        viewModel                =                ViewModelProvider(this,
viewModelFactory)[BreweriesViewModel::class.java]
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return  inflater.inflate(R.layout.fragment_breweries,  container,
false)
    }

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

        showKeyboardRunnable = Runnable {
            searchField.showKeyboard()
        }

        requireActivity().run {
            changeStatusBarColor(R.color.green)
```

```
window.setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_ADJUST_NOTHING)
            }

            searchField.doOnTextChanged { text, _, _, _ ->
                viewModel.onBrewerySearchQueryChanged(
                    text?.trim().toString()
                )
            }

            breweries.adapter = BreweryAdapter {
                viewModel.onBreweryOnMapClicked(it)
            }
            breweries.addItemDecoration(object                                  :
RecyclerView.ItemDecoration() {
                override fun getItemOffsets(
                    outRect: Rect,
                    view: View,
                    parent: RecyclerView,
                    state: RecyclerView.State
                ) {
                    val offset = dpToPx(12)
                    val bottomOffset =
                        if      (parent.getChildAdapterPosition(view)      ==
parent.adapter!!.itemCount - 1) {
                            offset
                        } else {
                            0
                        }

                    outRect.set(offset, offset, offset, bottomOffset)
                }
            })

            breweries.addOnScrollListener(object                                :
RecyclerView.OnScrollListener() {
                override fun onScrolled(recyclerView: RecyclerView, dx: Int,
dy: Int) {
                    super.onScrolled(recyclerView, dx, dy)

                    val             translationZ             =             if
(!recyclerView.canScrollVertically(-1)) {
                        0f
                    } else {
                        dpToPxFloat(10)
                    }

                    searchContainer.translationZ = translationZ
                    toolbar.translationZ = translationZ
                }
            })

            viewModel.breweriesState.observe(viewLifecycleOwner) { breweries
->
                val items = breweries.map {
                    val info = createInfoFrom(it)
                    BreweryItem(it.id, it.name, info, it.location)
                }

                (this.breweries.adapter as BreweryAdapter).submitList(items)
{
                    this.breweries.scrollToPosition(0)
                }
            }
```

```
            viewModel.onViewCreated()
        }

        override fun onStart() {
            super.onStart()

            if (searchField.isFocused) {
                searchField.postDelayed(showKeyboardRunnable,
SHOW_KEYBOARD_DELAY)
            }
        }

        override fun onDestroyView() {
            searchField.removeCallbacks(showKeyboardRunnable)
            requireActivity().changeStatusBarColorToPrimaryDark()
            super.onDestroyView()
        }

        private fun createInfoFrom(brewery: Brewery): Map<String,
CharSequence> {
            return brewery.run {
                val websiteSpannable = createWebsiteSpannableStr(websiteUrl)

                val values = listOf(
                    country,
                    state,
                    city,
                    postalCode,
                    street,
                    phone,
                    websiteSpannable
                )

                titles
                    .zip(values)
                    .filter { pair -> pair.second.isNullOrEmpty().not() }
                    .associate { (titleId: Int, value: CharSequence?) ->
                        val titleStr = resources.getString(titleId)

                        "$titleStr:" to TextUtils.concat(" ", value!!)
                    }
            }
        }

        private fun createWebsiteSpannableStr(websiteUrl: String?):
CharSequence? {
            return if (websiteUrl != null) {
                val spannableStr = SpannableString(websiteUrl)

                spannableStr.setSpan(
                    URLSpan(websiteUrl),
                    0,
                    websiteUrl.length,
                    Spanned.SPAN_INCLUSIVE_EXCLUSIVE
                )

                spannableStr
            } else {
                null
            }
        }

        companion object {
            private const val SHOW_KEYBOARD_DELAY = 50L
```

```
            fun newInstance(): BreweriesFragment {
                return BreweriesFragment()
            }
        }
    }

    package davydov.dmytro.breweries_list.breweries.ui

    import android.text.method.LinkMovementMethod
    import android.view.LayoutInflater
    import android.view.View
    import android.view.ViewGroup
    import androidx.core.view.children
    import androidx.core.view.isVisible
    import androidx.recyclerview.widget.DiffUtil
    import androidx.recyclerview.widget.ListAdapter
    import androidx.recyclerview.widget.RecyclerView
    import davydov.dmytro.breweries_list.R
    import davydov.dmytro.breweries_list.breweries.logic.Location
    import kotlinx.android.synthetic.main.item_brewery.view.*
    import kotlinx.android.synthetic.main.item_brewery.view.title
    import kotlinx.android.synthetic.main.item_info.view.*

    class BreweryAdapter(private val showOnMapClickListener: (Location) ->
Unit) :
        ListAdapter<BreweryItem, BreweryVH>(diffUtilCallback) {

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
BreweryVH {
            val                              view                              =
LayoutInflater.from(parent.context).inflate(R.layout.item_brewery,     parent,
false)

            val vh = BreweryVH(view)

            vh.itemView.showOnMap.setOnClickListener {
                val pos = vh.adapterPosition

                if (pos != RecyclerView.NO_POSITION) {
                    showOnMapClickListener(getItem(pos).location!!)
                }
            }

            return vh
        }

        override fun onBindViewHolder(holder: BreweryVH, position: Int) {
            val item = getItem(position)
            holder.bind(item)
        }

        companion object {
            val         diffUtilCallback         =         object         :
DiffUtil.ItemCallback<BreweryItem>() {
                override fun areItemsTheSame(oldItem: BreweryItem, newItem:
BreweryItem): Boolean {
                    return oldItem.id == newItem.id
                }

                override   fun   areContentsTheSame(oldItem:   BreweryItem,
newItem: BreweryItem): Boolean {
                    return oldItem == newItem
                }
```

```
            }
        }
    }

    class BreweryVH(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val titleTv = itemView.title
        private val infoLayout = itemView.info
        private val showOnMap = itemView.showOnMap

        fun bind(breweryItem: BreweryItem) {
            titleTv.text = breweryItem.title

            infoLayout.removeAllViews()

            for ((title, value) in breweryItem.info) {
                val itemInfo =

LayoutInflater.from(itemView.context).inflate(R.layout.item_info,  infoLayout,
false)
                itemInfo.title.text = title
                itemInfo.value.text = value
                itemInfo.value.movementMethod                                =
LinkMovementMethod.getInstance()

                infoLayout.addView(itemInfo)
            }

            showOnMap.isVisible = breweryItem.location != null
        }
    }
```

```
    package davydov.dmytro.breweries_list.breweries.ui

    import davydov.dmytro.breweries_list.breweries.logic.Location

    data class BreweryItem(
        val id: Int,
        val title: String,
        val info: Map<String, CharSequence>,
        val location: Location?
    )
```

```
    package davydov.dmytro.breweries_list.breweries.data

    import davydov.dmytro.breweries_list.breweries.logic.Brewery
    import davydov.dmytro.breweries_list.root.storage.BreweriesDao
    import davydov.dmytro.breweries_list.root.storage.BrewerySearch
    import davydov.dmytro.breweries_list.root.storage.LocalBrewery
    import davydov.dmytro.breweries_list.root.storage.SearchCrossRef
    import io.reactivex.Completable
    import io.reactivex.Single
    import javax.inject.Inject

    class LocalBreweriesSource @Inject constructor(private val breweriesDao:
BreweriesDao) {
        fun loadBreweries(): Single<List<Brewery>> {
            return breweriesDao
                .getBreweries()
                .map { localBreweries -> localBreweries.map { it.toDomain()
} }
        }

        fun updateBreweriesList(breweries: List<Brewery>): Completable {
            return Completable
```

```
                .fromAction {
                    val localBreweries =
                        breweries.map { LocalBrewery.from(it) }
                    breweriesDao.updateGeneralInfoBreweries(localBreweries)
                }
        }

        fun searchBreweriesBy(query: String): Single<List<Brewery>> {
            return breweriesDao
                .getSearchedBreweriesBy(query)
                .map { localBreweries -> localBreweries.map { it.toDomain()
} }
        }

        fun      updateSearchedBreweries(query:      String,      breweries:
List<Brewery>): Completable {
            return Completable
                .fromAction {
                    val brewerySearch = BrewerySearch(query)
                    val      localBreweries      =      breweries.map      {
LocalBrewery.from(it, false) }
                    val searchCrossRefs =
                        breweries.mapIndexed    {    index,    value    ->
SearchCrossRef(query, value.id, index) }

                    breweriesDao.updateSearchedBreweries(localBreweries,
brewerySearch, searchCrossRefs)
                }
        }
    }

    package davydov.dmytro.breweries_list.root

    import android.os.Bundle
    import dagger.android.support.DaggerAppCompatActivity
    import davydov.dmytro.breweries_list.R
    import davydov.dmytro.breweries_list.base.ConnectionStateService
    import davydov.dmytro.breweries_list.breweries.ui.BreweriesFragment
    import java.io.Closeable
    import javax.inject.Inject

    class RootActivity : DaggerAppCompatActivity() {

        @Inject
        lateinit var connectionStateService: ConnectionStateService

        override fun onCreate(savedInstanceState: Bundle?) {
            setTheme(R.style.AppTheme)
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_root)

            savedInstanceState ?: addBreweriesFragment()
        }

        override fun onDestroy() {
            connectionStateService.close()
            super.onDestroy()
        }

        private fun addBreweriesFragment() {
            supportFragmentManager
                .beginTransaction()
                .add(R.id.container, BreweriesFragment.newInstance())
                .commit()
```

```
        }
    }
    package davydov.dmytro.breweries_list.root.storage

    import androidx.room.Entity
    import androidx.room.PrimaryKey
    import davydov.dmytro.breweries_list.breweries.logic.Brewery
    import davydov.dmytro.breweries_list.breweries.logic.Location

    @Entity(tableName = "Brewery")
    data class LocalBrewery(
        @PrimaryKey val breweryId: Int,
        val name: String,
        val street: String?,
        val city: String?,
        val country: String?,
        val postalCode: String?,
        val state: String?,
        val lat: Double?,
        val lng: Double?,
        val websiteUrl: String?,
        val phone: String?,
        val inGeneralList: Boolean
    ) {
        fun toDomain(): Brewery {
            val location = if (lat != null && lng != null) {
                Location(lat, lng)
            } else {
                null
            }

            return Brewery(
                breweryId,
                name,
                street,
                city,
                state,
                postalCode,
                country,
                location,
                phone,
                websiteUrl
            )
        }

        companion object {
            fun from(brewery: Brewery, inGeneralList: Boolean = true):
LocalBrewery {
                return brewery.run {
                    LocalBrewery(
                        id,
                        name,
                        street,
                        city,
                        country,
                        postalCode,
                        state,
                        location?.lat,
                        location?.lng,
                        websiteUrl,
                        phone,
                        inGeneralList
                    )
                }
```