Dokumentacja projektu: Bezpiecznik Inżyniera Oprogramowania

Wojciech Kajstura, Konrad Lubera, Remigiusz Drobinski

21 stycznia 2021

Politechnika Śląska

Spis treści

1 Opis projektu

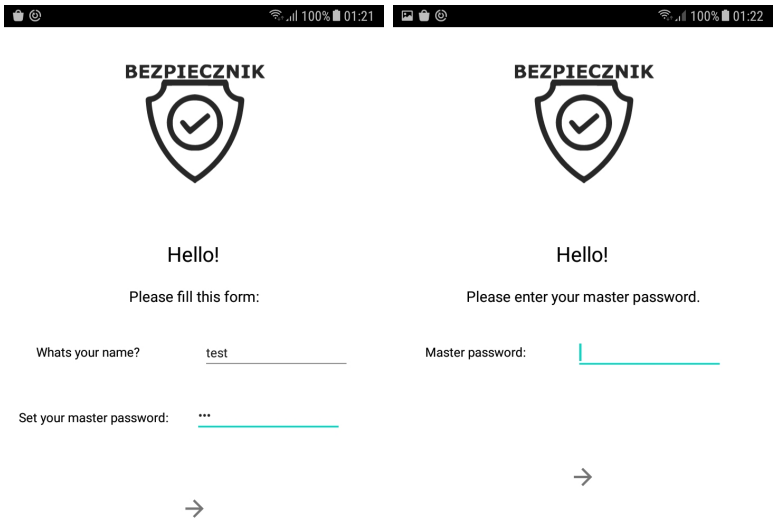
1.1 Opis programu

Bezpiecznik to aplikacja na urządzenia mobilne, pozwalająca na weryfikację utworzonego przez użytkownika wzoru. Użytkownik może zmieniać liczbę kolumn i wierszy czy wpływać na czysto wizualny wygląd planszy. Po narysowaniu wzoru otrzymuje informację jak bezpieczne jest jego hasło. Ponadto ma podgląd do historii. Aby korzystać z aplikacji wymagane jest utworzenie konta, podając nazwę oraz hasło, które zostaje przypisane do konkretnego urządzenia mobilnego.

1.2 Instrukcja obsługi

1.2.1 Ekran powitalny

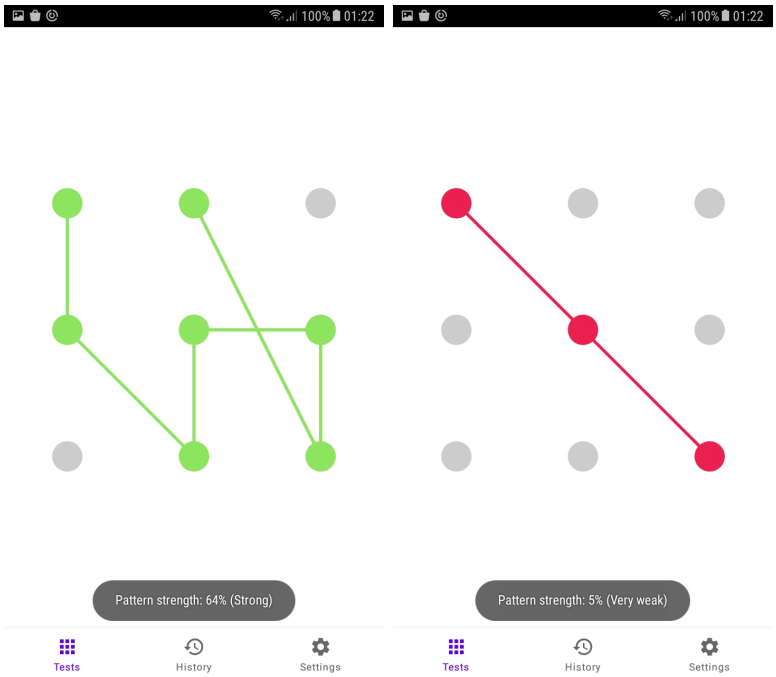
Podczas pierwszego uruchomienia aplikacja wyświetli okno wymagające podania imienia i hasła, które zapisze w danych aplikacji. W przypadku ponownego uruchomienia aplikacji będziemy wpisywać już tylko hasło.



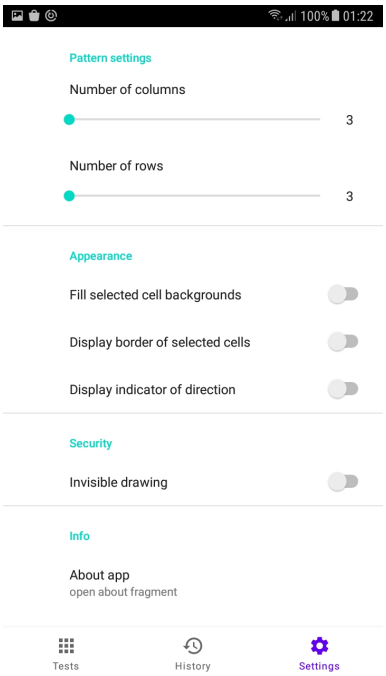
Po zalogowaniu naszym oczom ukaże się ekran rysowania wzoru. Po jego narysowaniu otrzymamy informację o sile hasła w postaci informacji zwrotnej zawartej w powiadomieniu(dymek „Toast") oraz zabarwieniu wzoru na odpowiadający sile kolor:

- Czerwony - bardzo słaby
- Pomarańczowy - słaby
- Żółty - średni
- Jasnozielony - mocny

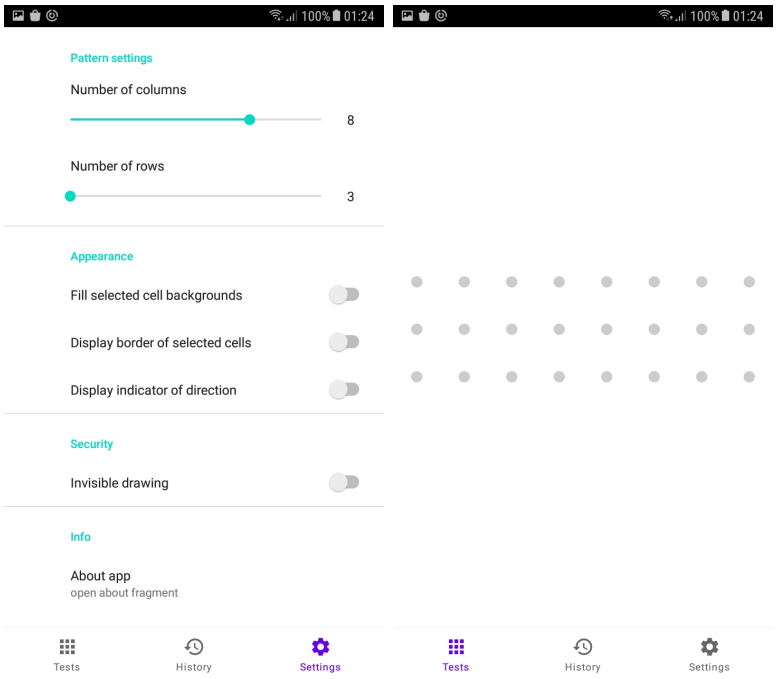
- Ciemnozielony - bardzo mocny



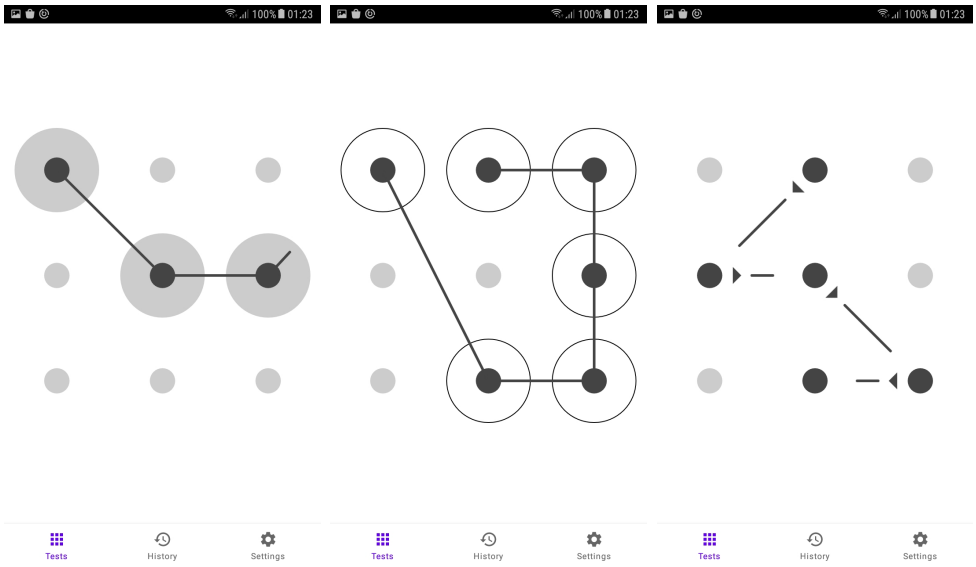
Wygląd siatki możemy zmienić przechodząc do ustawień w dolnym menu.



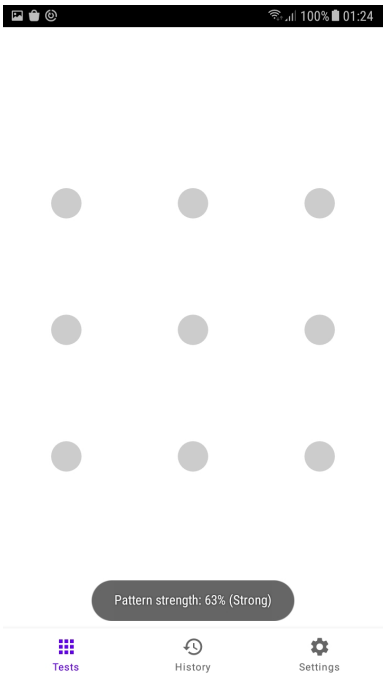
Możemy przede wszystkim zmienić rozmiar siatki za pomocą suwaków w sekcji „Pattern Settings”



W sekcji „Appearance” możemy zaznaczyć takie opcje jak podświetlenie tła kropki, narysowanie obramowania kropki, czy narysowanie wskaźnika kierunku.

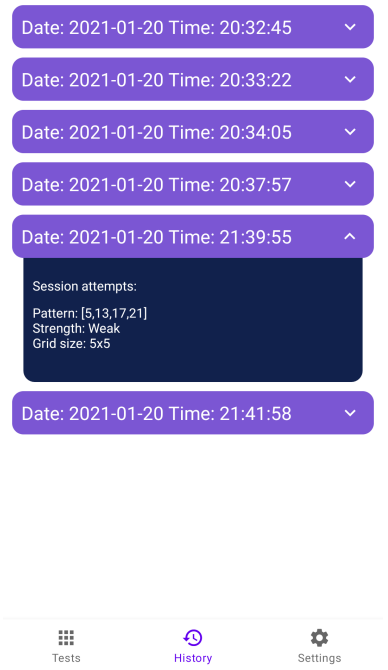


W sekcji „Secure” możemy zaznaczyć opcje, która sprawia, że rysowanie jest niewidoczne.



Natomiast przycisk „About” przenosi nas do widoku informacyjnego o aplikacji.

Przechodząc do sekcji „History” w dolnym menu, możemy podejrzeć ostatnio wykonywane testy.



1.3 Przebieg realizacji projektu

Pierwotny pomysł zrealizowania aplikacji od razu w postaci mobilnej wymusił na nas niejako szybsze i dogłębniejsze poznanie języka programowania Kotlin i zasad panujących pisząc kod w tym języku.

Po opanowaniu w stopniu wymagającym języka Kotlin, zasobu sieciowego oraz struktury wzorca MVVM w Kotlin/Android przystąpiliśmy do projektowania aplikacji.

Wybór architektury MVVM miał na celu oddzielenie widoków od kodu, co pozwoliło na lepszy podział pracy.

Podstawowym wzorcem projektowym naszej aplikacji były domyślne narzędzia Pattern Lock Androida, jednak zważywszy iż istnieje ich wiele wersji trudno znaleźć w naszej aplikacji konkretne elementy, które byłyby wzorowane na konkretnej wersji konkretnego narzędzia.

Kiedy podstawowa struktura aplikacji była gotowa podzieliliśmy resztę realizacji projektu na 3 działy - „siatka”, „komunikacja” oraz „algorytm” - każdy zrealizował swoje zadanie, dbając jednocześnie o dobry dla oka wygląd aplikacji.

2 Część techniczna

2.1 Opis widoku siatki

2.1.1 Opis

Implementacja siatki to nic innego jak stworzenie nowych obiektów graficznych, tak zwanych „custom views" które za pomocą kodu zostaną wygenerowane. Wspomniane „custom views" dziedziczą po podstawowych klasach odpowiadających za pewien layout, co pozwala wykorzystać wiele pomocnych metod.

Następujące klasy odpowiadają za tworzenie widoków w przypadku naszego programu:

- PatternLockView.kt
- CellView.kt

Stworzony widok możemy bezproblemowo użyć w layoucie, zupełnie tak samo jak te wbudowane.

Listing 1: Użycie stworzonego widoku w layoucie

```
<com.example.bezpiecznik.views.customviews.PatternLockView
    android:id="@+id/pattern_lock_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.498"
    style="@style/PatternLockView"
/>
```

2.1.2 PatternLockView

Widok ten dziedziczy znany z Androida GridLayout. W naszym przypadku dziedziczymy MvvmGridLayout, czyli abstrakcyjną klasę, która dziedziczy po wspomnianym GridLayout oraz implementuje IMvvmCustomView - interfejs, który zaimplementowaliśmy, aby móc skorzystać z cyklu życia(Lifecycle). Jest to zabieg, który służy dołączeniu ViewModelu do naszego stworzonego widoku, aby trzymać się wzorca MVVM.

Pomimo zadania sobie trudu tej implementacji, nie została ona dotychczas wykorzystana, z racji na wbudowane w Androidzie „Preferences", czyli ustawienia(których wykorzystanie zostanie opisane poniżej). Niemniej jednak nie usunęliśmy jej z programu, z racji na perspektywe wykorzystania w kontynuacjach pracy nad tym projektem.

Klasa odpowiada ta za stworzenie siatki z wygenerowanych kropek, jest dla nich czymś w rodzaju kontenera. Odpowiada ona za większość operacji.

Listing 2: Klasa PatternLockView dziedziczy po MvvmGridLayout

```
class PatternLockView(context: Context, attributeSet: AttributeSet)
```

```
    : MvvmGridLayout<PatternLockViewState, PatternLockViewModel>(context
        , attributeSet)
```

Listing 3: Abstrakcyjna klasa MvvmGridLayout, która dziedziczy po GridLayout

```
abstract class MvvmGridLayout<V: IMvvmCustomViewState, T:
    IMvvmCustomViewModel<V>>(
    context: Context,
    attributeSet: AttributeSet) :
    GridLayout(context, attributeSet),
    IMvvmCustomView<V, T> {

override fun onAttachedToWindow() {
    super.onAttachedToWindow()
    val lifecycleOwner = context as? LifecycleOwner ?: throw
        LifecycleOwnerNotFoundException()
    onLifecycleOwnerAttached(lifecycleOwner)
}

override fun onSaveInstanceState() =
    MvvmCustomViewStateWrapper(super.onSaveInstanceState(),
        viewModel.state)

override fun onRestoreInstanceState(state: Parcelable?) {
    if (state is MvvmCustomViewStateWrapper) {
        viewModel.state = state.state as V?
        super.onRestoreInstanceState(state.superState)
    }
}
}
```

Opisy funkcji zastosowanych w klasie:

- override onTouchEvent(event: MotionEvent?): Boolean

  Nadpisujemy funkcje pochodzącą z klasy View, odpowiadającą za zdarzenie dotyku ekranu.

Listing 4: Funkcja onTouchEvent

```
override fun onTouchEvent(event: MotionEvent?): Boolean {
    if (!drawAbility) return false

    when(event?.action) {
        MotionEvent.ACTION_DOWN -> {
            val hitCell = getHitCell(event.x.toInt(), event.y.
                toInt())
            if (hitCell == null) {
                return false
            } else {
                notifyCellSelected(hitCell)
            }
        }

        MotionEvent.ACTION_MOVE -> handleActionMove(event)
        MotionEvent.ACTION_UP -> onFinish()
```

```
16              MotionEvent.ACTION_CANCEL -> reset()
17
18              else -> return false
19          }
20          return true
21      }
```

- override dispatchDraw(canvas: Canvas?)

  Nadpisujemy funkcje pochodzącą z klasy View, odpowiadającą za rysowanie po ekranie.

Listing 5: Funkcja dispatchDraw

```
1       override fun dispatchDraw(canvas: Canvas?) {
2           super.dispatchDraw(canvas)
3           if (invisibleDrawing) return
4           canvas?.drawPath(patternPath, patternPaint)
5
6           if (selectedCells.size > 0 && lastPointX > 0 && lastPointY
                > 0) {
7               if (!showIndicator){
8                   val center = selectedCells[selectedCells.size - 1].
                        getCenter()
9                   canvas?.drawLine(center.x.toFloat(), center.y.
                        toFloat(), lastPointX, lastPointY, patternPaint)
10              } else{
11                  val lastCell = selectedCells[selectedCells.size -
                        1]
12                  val lastCellCenter = lastCell.getCenter()
13                  val radius = lastCell.getRadius()
14
15                  if (!(lastPointX >= lastCellCenter.x - radius &&
16                                  lastPointX <= lastCellCenter.x +
                                        radius &&
17                                  lastPointY >= lastCellCenter.y -
                                        radius &&
18                                  lastPointY <= lastCellCenter.y +
                                        radius)) {
19                      val diffX = lastPointX - lastCellCenter.x
20                      val diffY = lastPointY - lastCellCenter.y
21                      val length = sqrt((diffX * diffX + diffY *
                            diffY).toDouble())
22                      canvas?.drawLine((lastCellCenter.x + radius *
                            diffX / length).toFloat(),
23                              (lastCellCenter.y + radius * diffY /
                                    length).toFloat(),
24                              lastPointX, lastPointY, patternPaint)
25                  }
26              }
27          }
28      }
```

- override removeAllViews()

Nadpisujemy funkcje pochodzącą z klasy View, odpowiadającą za wyczyszczenie wszystkich widoków, które są „dziećmi” PatternLockView.

Listing 6: Funkcja removeAllViews

```
1     override fun removeAllViews() {
2         super.removeAllViews()
3         cells.clear()
4     }
```

- handleActionMove()

Funkcja, która po ruchu uruchamia funkcje odpowiadającą za sprawdzenie czy któraś komórka została trafiona, jeśli tak jest podaje ją do funkcji, która zajmuje się obsłużeniem takiego przypadku.

Listing 7: Funkcja handleActionMove

```
1     private fun handleActionMove(event: MotionEvent) {
2         val hitCell = getHitCell(event.x.toInt(), event.y.toInt())
3         if (hitCell != null) {
4             if (!selectedCells.contains(hitCell)) {
5                 notifyCellSelected(hitCell)
6             }
7         }
8
9         lastPointX = event.x
10        lastPointY = event.y
11
12        invalidate()
13    }
```

- notifyCellSelected()

Funkcja, która odpowiada za reakcje na zaznaczenie komórki. Odpowiada ona również za narysowanie linii pomiędzy komórkami(kropkami, które się w nich znajdują).

Listing 8: Funkcja notifyCellSelected

```
1     private fun notifyCellSelected(cell: CellView) {
2         selectedCells.add(cell)
3
4         if (invisibleDrawing) return
5
6         cell.setState(DotState.SELECTED)
7         val center = cell.getCenter()
8         if (selectedCells.size == 1) {
9             patternPath.moveTo(center.x.toFloat(), center.y.toFloat
                  ())
10        } else {
11            if(!showIndicator){
12                patternPath.lineTo(center.x.toFloat(), center.y.
                      toFloat())
13            }else{
```

```
14                  val previousCell = selectedCells[selectedCells.size
                        - 2]
15                  val previousCellCenter = previousCell.getCenter()
16                  val diffX = center.x - previousCellCenter.x
17                  val diffY = center.y - previousCellCenter.y
18                  val radius = cell.getRadius()
19                  val length = sqrt((diffX * diffX + diffY * diffY).
                        toDouble())
20
21                  patternPath.moveTo((previousCellCenter.x + radius *
                        diffX / length).toFloat(), (previousCellCenter.
                        y + radius * diffY / length).toFloat())
22                  patternPath.lineTo((center.x - radius * diffX /
                        length).toFloat(), (center.y - radius * diffY /
                        length).toFloat())
23
24                  val degree = Math.toDegrees(atan2(diffY.toDouble(),
                        diffX.toDouble())) + 90
25                  previousCell.setDegree(degree.toFloat())
26                  previousCell.invalidate()
27              }
28          }
29      }
```

- reset()

  Funkcja służąca do czyszczenia planszy po akcji rysowania.

  Listing 9: Funkcja reset

```
1       fun reset() {
2           for(cell in selectedCells) {
3               cell.reset()
4           }
5
6           selectedCells.clear()
7           patternPaint.color = selectedColor
8           patternPath.reset()
9
10  //          lastPointX = 0f
11  //          lastPointY = 0f
12
13          drawAbility = true
14          invalidate()
15      }
```

- initDots()

  Funkcja odpowiadająca za zainicjowanie komórek, czyli narysowanie widoków CellView na planszy.

  Listing 10: Funkcja initDots

```
1       fun initDots() {
2           var numbering = 1
```

```
        for(i in 0 until patternRowCount) {
            for(j in 0 until patternColCount) {
                val cell =
                        CellView(context,
                                numbering,
                                patternColCount, patternRowCount,
                                sleepColor, selectedColor,
                                showCellBackground, showBorder,
                                   showIndicator,
                                border)
                val cellPadding = 72 / columnCount
                cell.setPadding(cellPadding, cellPadding,
                   cellPadding, cellPadding)
                addView(cell)
                cells.add(cell)
                numbering++
            }
        }
    }
```

- initPathPaint()

  Funkcja inicjująca obiekt linii, która łączy kropki.

Listing 11: Funkcja initPathPaint

```
    private fun initPathPaint() {
        patternPaint.isAntiAlias = true
        patternPaint.isDither = true
        patternPaint.style = Paint.Style.STROKE
        patternPaint.strokeJoin = Paint.Join.ROUND
        patternPaint.strokeCap = Paint.Cap.ROUND
        patternPaint.strokeWidth = 6f
        patternPaint.color = selectedColor
    }
```

- getHitCell(x: Int, y: Int): CellView?

  Funkcja, która zwraca zaznaczony widok(komórkę/kropkę), jeśli tak się zdało. Jeżeli miejsce, które podajemy(x, y) nie zawiera się w żadnym widoku - otrzymujemy null.

Listing 12: Funkcja getHitCell

```
    private fun getHitCell(x: Int, y: Int) : CellView? {
        for(cell in cells) {
            if (isSelected(cell, x, y)) {
                return cell
            }
        }
        return null
    }
```

- isSelected(view: View, x: Int, y: Int): Boolean

Funkcja, która sprawdza, czy miejsce które podajemy jako argumenty (x, y) zawiera się w widoku, który podajemy jako argument (view).

Listing 13: Funkcja isSelected

```
1     private fun isSelected(view: View, x: Int, y: Int) : Boolean {
2         val innerPadding = view.width * 0.2f
3
4         return  x >= view.left + innerPadding &&
5                 x <= view.right - innerPadding &&
6                 y >= view.top + innerPadding &&
7                 y <= view.bottom - innerPadding
8     }
```

- onFinish()

Funkcja uruchamiana po zakończeniu rysowania. Wywołuje ona konieczne funkcje, a po wyznaczonym czasie, wywołuje funkcje reset.

Listing 14: Funkcja onFinish

```
1     private fun onFinish() {
2         lastPointX = 0f
3         lastPointY = 0f
4
5         if (selectedCells.size < 3){
6             val toast = Toast.makeText(context, "The␣pattern␣length
                  ␣is␣at␣least␣3", Toast.LENGTH_SHORT)
7             toast.show()
8             if (!invisibleDrawing){
9                 setColorAfterDrawing(veryWeakPatternColor)
10            }
11        }
12        else{
13            val strength = getPatternStrength()
14            if (!invisibleDrawing){
15                setColorAfterDrawing(getColorByPatternStrength(
                      strength))
16            }
17        }
18
19        drawAbility = false
20        invalidate()
21
22        postDelayed({
23            reset()
24        }, previewTimeAfterDrawing.toLong())
25    }
```

- getColorByPatternStrength(strength: PatternStrength): Int

Funkcja, która zwraca odpowiedni kolor dla podanej jako argument siły hasła.

Listing 15: Funkcja getColorByPatternStrength

```
1     private fun getColorByPatternStrength(strength: PatternStrength
          ):Int {
2         return when(strength){
3             PatternStrength.VERY_STRONG -> veryStrongPatternColor
4             PatternStrength.STRONG -> strongPatternColor
5             PatternStrength.MEDIUM -> mediumPatternColor
6             PatternStrength.WEAK -> weakPatternColor
7             else -> veryWeakPatternColor
8         }
9     }
```

- setColorAfterDrawing(color: Int)

  Funkcja odpowiadająca za ustawienie koloru kropek i linii na taki, który odpowiada sile hasła.

Listing 16: Funkcja setColorAfterDrawing

```
1     private fun setColorAfterDrawing(color: Int){
2         for (cell in selectedCells) {
3             cell.setPatternStrengthColor(color)
4             cell.setState(DotState.AFTER)
5         }
6         patternPaint.color = color
7     }
```

- getPatternStrength()

  Funkcja, która korzystając z algorytmu pobiera siłe hasła, a następnie wyświetla ją na ekranie, oraz dodaje do listy aktualną próbę.

Listing 17: Funkcja getPatternStrength

```
1     private fun getPatternStrength(): PatternStrength{
2         val arrayOfSelectedDotsNumbers: ArrayList<Int> = ArrayList
              ()
3
4         for (cell in selectedCells) {
5             arrayOfSelectedDotsNumbers.add(cell.dotNumber)
6         }
7
8         val array = arrayOfSelectedDotsNumbers.toTypedArray()
9
10        val res = Counter(patternRowCount, patternColCount, array)
11        val resPrint = res.printer()
12        val strength = res.verbalScaleResult(resPrint)
13
14
15        val toastStrength = res.verbalScaleResult(res.printer())
16        val toast = Toast.makeText(context, "Pattern␣strength:␣
              $resPrint%␣(${getStrengthInString(toastStrength)})",
              Toast.LENGTH_SHORT)
17        toast.show()
```

```
18
19          attemptList.add(Attempt(res.toStringConverter(array),
                getStrengthInString(toastStrength),patternRowCount,
                patternColCount))
20
21          return strength
22      }
```

2.1.3 CellView

Widok ten dziedziczy po podstawowej klasie View.
Klasa odpowiada za jedną komórkę w całej siatce, która ma za zadanie przestawienie danej kropki. Jest wielorazowo wykorzystywana, wszystko w zależności od wybranego rozmiaru siatki.

Listing 18: Klasa CellView dziedziczy po View

```
1     class CellView(context: Context,
2                 var dotNumber: Int, var columnCount: Int,var rowCount:Int
                      ,
3                 var sleepColor: Int,var selectedColor: Int,
4                 var showCellBackground: Boolean,var showBorder: Boolean,
                      var showIndicator: Boolean,
5                 var border: Drawable?): View(context)
```

Opisy funkcji zastosowanych w klasie:

- override onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int)

  Nadpisujemy funkcje pochodzącą z klasy View, odpowiadająca za ustalenie miary komórki w przypadku podanej jako argument ilości kolumn i wierszy. W zależności czy więcej jest kolumn, czy wierszy, nadaje im takie miary(a w niektórych przypadkach też marginesy), aby wszystkie były widoczne i wyśrodkowane.

Listing 19: Funkcja onMeasure

```
1     override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec
          : Int) {
2         super.onMeasure(widthMeasureSpec, heightMeasureSpec)
3
4         if (columnCount + 1 >= rowCount){
5             val cellWidth = MeasureSpec.getSize(widthMeasureSpec) /
                  columnCount
6             setMeasuredDimension(cellWidth, cellWidth)
7
8         } else {
9             val difference  = rowCount - columnCount
10            val ratio: Double
11            ratio = when {
12                difference < 3 -> {
13                    0.7
14                }
15                difference == 3 -> {
16                    0.8
```

```
                }
                else -> {
                    0.85
                }
            }
            val cellHeight = ((MeasureSpec.getSize(
                heightMeasureSpec) * ratio) / rowCount).toInt()
            val freeSpaceToSetOnLeft = (MeasureSpec.getSize(
                widthMeasureSpec) - (columnCount * cellHeight)) / 2

            if (dotNumber % columnCount == 1){
                val t = this.layoutParams as ViewGroup.
                    MarginLayoutParams
                t.setMargins(freeSpaceToSetOnLeft,0,0,0)
            }

            setMeasuredDimension(cellHeight, cellHeight)
        }
    }
```

- override onDraw(canvas: Canvas?)

  Nadpisujemy funkcje pochodzącą z klasy View służącą do rysowania, w naszym wypadku kropek za pomocą funkcji drawDot.

Listing 20: Funkcja onDraw

```
    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)

        when(state){
            DotState.SLEEP -> drawDot(canvas,null,null, sleepColor)
            DotState.SELECTED -> drawDot(canvas, cellBackground,
                border, selectedColor)
            DotState.AFTER -> drawDot(canvas, cellBackground,
                border, patternStrengthColor)
        }
    }
```

- drawDot(canvas: Canvas?, background: Drawable?, borderCell: Drawable?, dotColor: Int, radiusRation: Float = 0.3f)

  Funkcja odpowiadająca za narysowanie kropki o odpowiednich parametrach.

Listing 21: Funkcja drawDot

```
    private fun drawDot(canvas: Canvas?, background: Drawable?,
        borderCell: Drawable?, dotColor: Int, radiusRation: Float =
        0.3f){
        var radius = getRadius()
        var centerX = width / 2
        var centerY = height / 2

        if (showCellBackground){
```

```
7               if (background is ColorDrawable) {
8                   paint.color = background.color
9                   paint.style = Paint.Style.FILL
10                  canvas?.drawCircle(centerX.toFloat(), centerY.
                        toFloat(), radius.toFloat(), paint)
11              }
12          }
13
14          if (showBorder){
15              borderCell?.setBounds(paddingLeft, paddingTop, width -
                    paddingRight, height - paddingBottom)
16              borderCell?.draw(canvas!!)
17          }
18
19          paint.color = dotColor
20          paint.style = Paint.Style.FILL
21          canvas?.drawCircle(centerX.toFloat(), centerY.toFloat(),
                radius * radiusRation, paint)
22
23          if (showIndicator && (state == DotState.SELECTED || state
                == DotState.AFTER)){
24              drawIndicator(canvas)
25          }
26      }
```

- drawIndicator(canvas: Canvas?)

  Funkcja odpowiadająca za narysowanie wskaźnika(ang. indicator)

Listing 22: Funkcja drawIndicator

```
1       private fun drawIndicator(canvas: Canvas?) {
2           if (degree != -1f){
3               if (indicatorPath.isEmpty) {
4                   indicatorPath.fillType = Path.FillType.WINDING
5                   val radius = getRadius()
6                   val height = radius * 0.2f
7                   indicatorPath.moveTo(
8                           (width / 2).toFloat(),
9                           radius * (1 - 0.3f - 0.2f) / 2 + paddingTop
                                )
10                  indicatorPath.lineTo(
11                          (width /2).toFloat() - height,
12                          radius * (1 - 0.3f - 0.2f) / 2 + height +
                                paddingTop)
13                  indicatorPath.lineTo(
14                          (width / 2).toFloat() + height,
15                          radius * (1 - 0.3f - 0.2f) / 2 + height +
                                paddingTop)
16                  indicatorPath.close()
17              }
18
19              if (state == DotState.SELECTED) {
20                  paint.color = selectedColor
21              } else {
```

```
22                  paint.color = patternStrengthColor
23              }
24
25              paint.style = Paint.Style.FILL
26
27              canvas?.save()
28              canvas?.rotate(degree, (width / 2).toFloat(), (height /
                    2).toFloat())
29              canvas?.drawPath(indicatorPath, paint)
30              canvas?.restore()
31          }
32      }
```

- getCenter(): Point

  Funkcja, która zwraca punkt będącym środkiem danej kropki

Listing 23: Funkcja getCenter

```
1       fun getCenter() : Point {
2           var point = Point()
3           point.x = left + (right - left) / 2
4           point.y = top + (bottom - top) / 2
5           return point
6       }
```

2.2 Opis komunikacji ze zdalnym repozytorium

2.2.1 Opis

Jako zdalne repozytorium w projekcie użyto serwisu `https://jsonbin.io/`, jest to API które pozwala przechowywać dane w formacie json. Jedną z funkcji jakie daje serwis jest utworzenie kolekcji wielu jsonów w przypadku tego projektu wykorzystano dwie kolekcje:

- User - przechowuje dane o użytkownikach - każdy użytkownik ma swój obiekt json, daje to możliwość późniejszej rozbudowy o np edycję danych użytkownika.
- Sessions - zawiera zapis sesji, tutaj również każdy użytkownik ma swojego jsona.

Do komunikacji ze zdalnym repozytorium użyto biblioteki retrofit2, jest to najpopularniejsza tego typu biblioteka, aby użytkowanie jej było prostsze należało utworzyć kilka dodatkowych klas oraz interfejs zawierający endpointy. Oto ich opis:

- Kalsa ApiRoutes - zawera bazowy adres url, w przypadku tego projektu nie było konieczności jej stosowania jednak aby ewentualna rozbudowa aplikacji była łatwiejsza (np. o kolejne api) zaimplementowano ją.

Listing 24: Klasa ApiRoutes

```
class ApiRoutes {
    companion object{
        const val BASE_URL = "https://api.jsonbin.io/"
    }
}
```

- Interfejs IApiRequest - zawiera zapytania które kierujemy do Api wraz w ich typem potrzebnym do deserializacji. (np. Call<User>). W tym miejscu należy wspomnieć iż zdalne repozytorium jest niepubliczne a dostęp do niego jest możliwy tylko po podaniu „secret-key" w hederze zapytania. Watro pomyśleć o dodatkowej ochronie tego wrażliwego punktu.

Listing 25: Fragemnt Interfejsu IApiRequest

```
interface IApiRequest {
    @Headers(
        "Content-Type: application/json",
        "secret-key: $2b$10\$SKWkhv2HZAovsIicIy/61
            eeFcJGrHgoev6y5zDriR4us.vHiFmRve",
        "private: true",
        "collection-id: 5ff8c26361f92720434a5530" //Users
            collection id
    )
    @POST("b")
    fun addUser(@Body user: User): Call<Response>

     @Headers(
        "secret-key: $2b$10\$SKWkhv2HZAovsIicIy/61
            eeFcJGrHgoev6y5zDriR4us.vHiFmRve"
    )
    @GET("b/{id}")
```

```
15      fun getUser(@Path("id") id: String) : Call<User
16
17      @Headers(
18          "Content-Type: application/json",
19          "secret-key: $2b$10\$SKWkhv2HZAovsIicIy/61
                eeFcJGrHgoev6y5zDriR4us.vHiFmRve",
20          "private: true",
21          "collection-id: 60073c581c1ce6535a0f1b64", //Session
                collection id
22          "versioning: false"
23      )
24      @PUT("b/{id}")
25      fun addSession(@Path("id") id: String, @Body records: Records):
            Call<Response>
26  }
```

- Klasa Response - jej obiekt jest tworzony podczas deserializacji większości odpowiedzi Api

Listing 26: Klasa Response

```
1 data class Response( @SerializedName(value = "name", alternate =
      arrayOf("id"))  var name: String ) {}
```

2.2.2 Konto użytkownika w zdalnym repozytorium

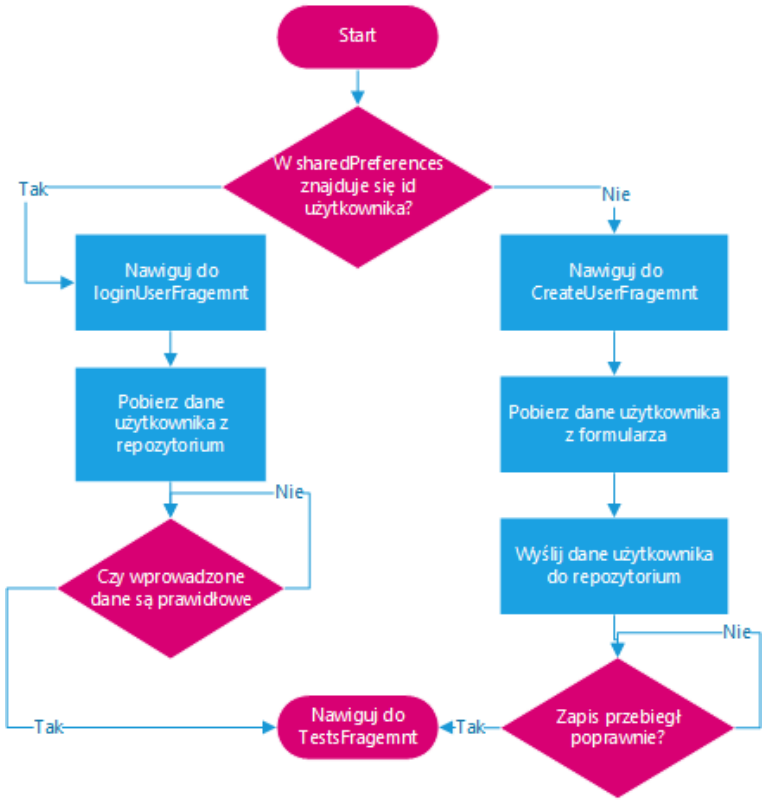
Wszystkie operacje na koncie użytkownika odbywają się w klasie UserViewModel. Sposób pobierania konta użytkownika oraz jego tworzenia został opisany w kolejnych akapitach. W tym zostaną opisane ważne fragmenty klasy UserViewModel:

- Klient retrofit (obiekt api)
- Companion object zawierający pobrany obiekt użytkownika, id jego jsona oraz id jego kolekcji sessji. Wykorzystano companion object żeby dostęp do tychże danych był prosty z każdego miejsca w aplikacji.

Listing 27: Klasa UserViewModel

```
1  class UserViewModel() : ViewModel() {
2
3      private val api = Retrofit.Builder().baseUrl(ApiRoutes.BASE_URL
           )
4          .addConverterFactory(GsonConverterFactory.create()).build()
5          .create(IApiRequest::class.java)
6
7          [...]
8
9          companion object{
10         lateinit var user: User
11         lateinit var binID : String
12         lateinit var collectionID: String
13     }
14 }
```

Poniżej skrócony schemat obsługi użytkownika:



Fragment kodu odpowiedzialny za sprawdzenie czy na urządzeniu zapisane są dane odnośnie użytkownika:

Listing 28: Klasa MainActivity

```
class MainActivity : AppCompatActivity() {
    private lateinit var sharedPref : SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        [..]

        sharedPref = this.getPreferences(Context.MODE_PRIVATE)

        if(!sharedPref.contains(getString(R.string.userID))){
            bottomNavigationView.visibility = View.GONE //Ukrycie
                dolnego paska nawigaci
            navController.navigate(R.id.createUserFragment)
        }
        else{
            bottomNavigationView.visibility = View.GONE
            navController.navigate(R.id.loginUserFragment)
        }

    }
}
```

Zapis
Gdy aplikacja jest uruchomiana po raz pierwszy tzn. w sharedPreferences nie znajdują się dane o użytkowniku aplikacja uruchamia fragment(widok) odpowiedzialny za uwożenie nowego użytkownika(CreateUserFragment), prosi o podanie imienia oraz ustanowienie hasła dostępu do aplikacji, po wypełnieniu tych danych tworzony jest obiekt użytkownika a jego zaserializowana postać wysyłana jest do repozytorium jako odpowiedz przychodzi unikalne id jsona zawierającego dane użytkownika, id to oraz unikalne wygenerowane id samego użytkownika (potrzebne do dodatkowej weryfikacji) zapisywane są w pamięci urządzenia za pomocą wspomnianych wcześniej sharedPreferences. Poniżej klasa User oraz jej zaserializowana postać oraz funkcja odpowiadająca za generowanie id oraz wysłanie danych o użytkowniku do Api.

- Klasa User

Listing 29: Klasa User

```
data class User (val id: String, var name: String, var
    masterPassword: String) {}
```

- User json

Listing 30: Klasa User

```
{
  "id": "173210321124123711601570174611851166 1896",
  "masterPassword": "123",
  "name": "Konrad"
}
```

- Metoda odpowiedzialna wysłanie obiektu użytkownika do zdalnego repozytorium

Listing 31: Metoda klasy UserViewModel createUser

```
fun createUser(name: String, masterPassword : String,
    saveToSPCallback:((u: User, id: String) -> Unit)){
     val userID: String = generateUserId()
     val user = User(userID, name, masterPassword)
     GlobalScope.launch(Dispatchers.IO) {
         val response  = api.addUser(user).awaitResponse()
         if (response.isSuccessful){
             val data = response.body()
             if (data != null) {
                     saveToSPCallback(user,data.name)
             }
             }
             else{
                 Log.d("api-connection","response failed")
             }
         }
     }
```

- Metoda odpowiedzialna za generowanie id użytkownika

Listing 32: Metoda kalsy UserViewModel generateUserId()

```
private fun generateUserId(): String {
    val random = SecureRandom()
    var randomString: String = ""
    random.setSeed(random.generateSeed(20))
    for (i in 1..10){
        randomString += (random.nextInt(1000 - 1 + 1) + 1000).
            toString()
    }
    return  randomString
    }
```

Odczyt

Jeżeli po uruchomieniu aplikacji w sharedPreferences znajdują się dane na temat użytkownika, aplikacja przenosi użytkownika do fragmentu odpowiedzialnego za logowanie (LoginUserFragment), następnie pobiera konto użytkownika ze zdalnego repozytorium i przypisuje do obiektu User w UserViewModel Companion Obejct. Gdy wprowadzone przez użytkownika hasło zgadza się z tym z zasobu aplikacja przenosi użytkownika do swojej głównej części. Poniżej metoda odpowiedzialna za pobranie użytkownia ze zdalnego repozytorium:

Listing 33: Metoda kalsy UserViewModel getUser()

```
    fun getUser(binID: String, doneCallback: ((d: Boolean) -> Unit)){
        GlobalScope.launch(Dispatchers.IO) {
            val response = api.getUser(binID).awaitResponse()
            if (response.isSuccessful){
                val data = response.body()
                if(data != null){
                    user = data // assignment to companion object
                    doneCallback(true)
                }
            }
            else{
                Log.d("api-connection","response failed")
            }
        }
    }
```

2.2.3 Sesje testów w zdalnym repozytorium

Aby zapisać wynik działania aplikacji w zdalnym repozytorium zaimplementowano sesje. Sesja zaczyna się gdy wpiszemy pierwszy wzór i kończy gdy opuścimy TestsFragemnt.

Tworzenie zasobu sesji

Każdy użytkownik ma swój unikalny obiekt sesji znajdujący się na zdalnym repozytorium jest on tworzony wraz z kontem użytkownika a id tego obiektu zapisywane jest w sharedPreferences. Na obiekt sesji składa się lista prób(klasa Attempt), data rozpoczęcia sesji oraz id użytkownika dla dodatkowej weryfikacji. Aby deserializacja danych była łatwiejsza konieczne było utworzenie pośredniej klasy Record zawierająca listę sesji. Pojedyncza próba(Attemp) składa się z zaznaczonego wzoru w formie listy, siły wzoru oraz wymiarów siatki.

- Zaserializowany obiekt sesji

Listing 34: Zaserializowany obiekt Records

```
{
  "records": [
    {
      "attempt": [
        {
          "columns": 3,
          "pattern": "[7,4,1,5,8,2,3,6,9]",
          "rows": 3,
          "strength": "Strong"
        },
        {
          "columns": 3,
          "pattern": "[7,4,5,8]",
          "rows": 3,
          "strength": "Weak"
        },
        {
          "columns": 3,
          "pattern": "[7,5,3]",
          "rows": 3,
          "strength": "Very weak"
        },
        {
          "columns": 3,
          "pattern": "[7,4,1,2,5,8,3,6,9]",
          "rows": 3,
          "strength": "Medium"
        }
      ],
      "startDate": "2021-01-17T11:27:25.700",
      "userId": "176816491306171811531491121114061846 1119"
    }
  ]
}
```

- Klasa Records

Listing 35: Klasa Records

```
class Records(@SerializedName("records") var records: ArrayList<
    Session>) {}
```

- Klasa Session

Listing 36: Klasa Session

```
class  Session(var startDate: String, var attempt: MutableList<
    Attempt>?, var userId: String) {}
```

- Klasa Attempt

Listing 37: Klasa Attempt

```
class Attempt (var pattern: String, var strength: String, var rows:
    Int, var columns: Int ) {
    override fun toString(): String {
        return "Pattern: $pattern Strength: $strength Size: $rows x
            $columns"
    }
}
```

- Metoda klasy UserViewModel createUserCollection() odpowiedzialna za tworzenie zbioru sesji na zdalnym repozytorium

Listing 38: Metoda createUserCollection

```
fun createUserCollection(doneCallback: ((d: Boolean) -> Unit)){
    GlobalScope.launch(Dispatchers.IO) {
        val response = api.createUserCollection(
                Records( arrayListOf(  Session("",mutableListOf(
                    Attempt("","",0,0)),""))))).awaitResponse()
            if (response.isSuccessful){
                val data = response.body()
                if(data != null){
                    collectionID = data.name
                    doneCallback(true)
                }
            }
            else{
                Log.d("api-connectionUVM","response failed")
            }
        }
    }
```

Zapisywanie sesji

Zapisywanie nowych sesji odbywa się poprzez zaktualizowanie jsona użytkownika z historią sesji(PUT), wykonywane jest ono w momencie gdy fragmenty TestsFragment zostanie opuszczony, wywoływana jest wtedy metoda onPause() która zbiera dotychczasowo wprowadzone wzory z PatterLockView oraz wywołuje metodę addSesion klasy HistoryViewModel odpowiedzialną za zapis do repozytorium.

- Metoda onPause TestsFragment

Listing 39: Metoda onPause

```
@RequiresApi(Build.VERSION_CODES.O)
override fun onPause() {
    super.onPause()
    val attempts = pattern_lock_id.getAttempts()
    if (attempts.size != 0)
    {
        HistoryViewModel.dataReady.postValue(false)
        historyViewModel.getSessions {
            val session = Session(LocalDateTime.now().toString(),
                attempts, UserViewModel.user.id )
```

```
10                historyViewModel.addSession(session){
11                    HistoryViewModel.dataReady.postValue(true)
12                }
13            }
14        }
15    }
```

- Metoda addSession() klasy HistoryViewModel

Listing 40: Metoda addSession()

```
1  fun addSession(session: Session,  doneCallback: ((d: Boolean) ->
       Unit)){
2      sessionList.value!!.add(session)
3      GlobalScope.launch(Dispatchers.IO) {
4          val response = api.addSession( UserViewModel.collectionID,
               Records(sessionList.value!!)).awaitResponse()
5          if (response.isSuccessful){
6              val data = response.body()
7              if(data != null){
8                  doneCallback(true)
9              }
10         }
11         else{
12             Log.d("api-connection",response.message())
13         }
14     }
15 }
```

Odczytywanie sesji

Odczytywanie sesji zrealizowano za pomocą metody getSession() w klasie HistoryViewModel, odpowiedz z api zapisywania jest do listy sessionList która znajduje się w companion object HistoryViewModelu. Odczytane dane wyświetlane są we fragmencie HistoryFragment (zakładka History) za pomocą recyclerView oraz HistoryListAdpater.

- Metoda getSession() klasy HistoryViewModel

Listing 41: Metoda getSession()

```
1  fun getSessions( doneCallback: ((d: Boolean) -> Unit) ){
2      GlobalScope.launch(Dispatchers.IO) {
3          val response = api.getUserCollection(UserViewModel.
               collectionID).awaitResponse()
4          if (response.isSuccessful){
5              val data = response.body()
6              if(data != null){
7                  if (data.records[0].userId == "")
8                      data.records.removeAt(0)
9                      //data.records.reverse()
10                 sessionList.postValue(data.records)
11                 doneCallback(true)
12             }
13         }
14         else{
15             Log.d("api-connection","response failed")
16         }} }
```

2.3 Opis algorytmu weryfikującego jakość wzoru

2.3.1 Opis

Algorytm sprawdzający jakość wzoru jest autorski a składają się na niego następujące charakterystyki:

- czy kod rozpoczyna się od rogu planszy
- czy kod jest linią poziomą bądź pionową
- czy kod jest przekątną w przypadku plansz kwadratowych
- różnica połączeń wertykalnych oraz horyzontalnych z połączeniami na skos poszczególnych sąsiedztw
- długość kodu względem całej planszy
- czy kod jest długości krótszego bądź dłuższego boku

Na samym początku algorytmu przypisywana jest zmienna o wartości 100, jest to także zmienna wynikowa, to właśnie tak procentowo bezpieczny jest wzór użytkownika. Następnie wykonywane są funkcje weryfikujące ww. charakterystyki dobrego i złego wzoru. Każda z operacji wiąże się z utratą punktów procentowych, dając ostateczny wynik. Zakres jaki może przyjąć ostateczny wynik to od 0 do 100. Po wygenerowaniu ostatecznego wyniku następuje konwersja wartości liczbowej w wartość w postaci komentarza, opisującego jakość wzoru.

Wyróżniamy następujące rodzaje hasła:

- od 100 do 81 - VERY STRONG
- od 80 do 61 - STRONG
- od 60 do 41 - MEDIUM
- od 40 do 21 - WEAK
- od 20 do 0 - VERY WEAK

Algorytm uwzględniający ww. charakterystyki nie ma wglądu jednak w umysł użytkownika, co za tym idzie w przypadkach zmian w modzie w generowaniu wzorów nie sprawdziłby się. Aby jak najbardziej zminimalizować nieścisłości powstała niewielka baza najpopularniejszych wzorów planszy 3x3.

Jeśli wzór użytkownika należy do bazy modnych, najpopularniejszych wzorów punkty procentowe bezpieczeństwa wzoru zostają odjęte.

2.3.2 Zaimplementowane funkcjonalności

Zakładamy, że plansza jest postaci:

$$\begin{pmatrix} 1 & 2 & 3 & ... & y \\ y+1 & y+2 & y+3 & ... & 2y \\ & . & & & \\ & . & & & \\ & . & & & \\ (x-1)y+1 & (x-1)y+2 & (x-1)y+3 & ... & xy \end{pmatrix},$$

gdzie:
x - liczba wierszy
y - liczba kolumn

- cornerStart() - sprawdza czy kod rozpoczyna się od rogu planszy

  Jeśli kod rozpoczyna się od rogu planszy to od 100 procent odejmowany jest wyznaczony procent

Listing 42: Czy kod rozpoczyna się od rogu?

```
1 private fun cornerStart(rows: Int, columns: Int, code: Array<Int>):
      Int {
2     if(code[0] == 1)
3         return 15
4     else if(code[0] == columns || code[0]==columns*(rows-1)+1)
5         return 14
6     else if(code[0] == columns * rows)
7         return 13
8     else
9         return 0
10 }
```

- horizontalLines() - sprawdza czy kod jest poziomą linią

  Jeśli kod jest poziomą linią to od dotychczasowego procentu bezpieczeństwa hasła odejmowane jest kolejno 30 punktów procentowych lub 25 punktów procentowyc. Wartość odjętych punktów procentowych zależy od tego czy kod jest z lewej czy z prawej strony. Kod, który jest linią poziomą i równocześnie zaczynający się z prawej strony jest bardziej bezpieczny niż ten z lewej.

Listing 43: Czy kod jest linią poziomą?

```
1 private fun horizontalLines(columns: Int, code: Array<Int>):
    Boolean {
2     var r = 0
3     var startBok = false
4     var res = false;
5
6     for(i in 0 until columns){
7         if(code[0] == columns * i + 1){
8             startBok = true
```

```
9           }
10
11      }
12
13      if(startBok && code.size == columns){
14          for(i in 0 until columns - 1){
15              if(code[i+1] - code[i] == 1)
16                  r++
17          }
18          if(r == columns - 1)
19              res = true
20      }
21      return res
22  }
```

- verticalLines() - sprawdza czy kod jest pionową linią

  Jeśli kod jest pionową linią to od dotychczasowego punktu procentowego bezpieczeństwa hasła odejmowane jest kolejno 30 punktów procentowych lub 25 punktów procentowyc. Wartość odjętych punktów procentowych zależy od tego czy kod jest z góry czy z dołu rysowany. Kod, który jest linią pionową i równocześnie zaczynający się z dołu jest bardziej bezpieczny niż ten z góry.

Listing 44: Czy kod jest linią pionową?

```
1   private fun verticalLines(rows: Int, columns: Int, code: Array<
        Int>): Boolean {
2       var r = 0
3       var startGD = false
4       var res = false
5
6       for(i in 0 until columns)
7           if(code[0] == i + 1)
8               startGD = true
9
10      if(startGD && code.size == rows){
11          for(i in 0 until rows-1){
12              if(code[i+1] - code[i] == columns)
13                  r++
14          }
15          if(r == rows - 1)
16              res = true
17      }
18      return res
19  }
```

- diagonal1() - sprawdza czy kod jest przekątną, prowadzoną z lewego górnego rogu do prawego dolnego bądź odwrotnie

  Jeśli kod jest ww. przekątną to od dotychczasowego punktu procentowego bezpieczeństwa hasła odejmowane jest kolejno 80 punktów procentowych.

Listing 45: Czy kod jest przekątną LG-PD lub PD-LG?

```
1 private fun diagonal1(rows: Int, columns: Int, code: Array<Int>):
      Boolean {
```

```
    var res = false
    var ile = 0
    if(rows == columns && code.size == columns){
        for(i in 0 until rows){
            if(code[i] == (rows + 1) * i + 1)
                ile += 1
        }
        if(ile == rows)
            res = true
    }
    return res
}
```

- diagonal2() - sprawdza czy kod jest przekątną, prowadzoną z lewego dolnego rogu do prawego górnego bądź odwrotnie

  Jeśli kod jest ww. przekątną to od dotychczasowego punktu procentowego bezpieczeństwa hasła odejmowane jest kolejno 80 punktów procentowych.

Listing 46: Czy kod jest przekątną LD-PG lub PG-LD?

```
private fun diagonal2(rows: Int, columns: Int, code: Array<Int>):
    Boolean{
    var ile = 0
    var res = false
    if(rows == columns && code.size == columns){
        for(i in 0 until rows){
            if(code[i] == rows * (1 + i) - i)
                ile += 1
        }
        if(ile == rows)
            res = true
    }
    return res
}
```

- neighborHorizontallyVertically() - zwraca ile występuje połączeń poziomo lub pionowo kolejnych po sobie elementów kodu

  Przykład
  Mamy planszę 3x3 :

  $$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$, natomiast wzór podany przez użytkownika ma postać [4,5,8,6]

  Wyżej wymieniona funkcja, zwróci nam liczbę 2, ponieważ 4 jest względem 5 sąsiadem poziomo oraz 5 jest względem 8 sąsiadem pionowo.

Listing 47: Ile występuje połączeń horyzontalych i wertykalnych?

```
private fun neighborHorizontallyVertically(columns: Int, code:
    Array<Int>): Int{
     var ile = 0
     for(i in 1 until code.size)
         if ((code[i] == code[i - 1] - 1 && code[i]%columns != 0) ||
         (code[i] == code[i - 1] + 1 && code[i]%columns != 1) ||
         code[i] == code[i - 1] - columns ||
         code[i] == code[i - 1] + columns)
             ile++
     return ile
}
```

- neighborDiagonally() - zwraca ile występuje połączeń na skos kolejnych po sobie elementów kodu

  Przykład
  Mamy planszę 3x3 :

  $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, natomiast wzór podany przez użytkownika ma postać [1,5,3,6,8]

  Wyżej wymieniona funkcja, zwróci nam liczbę 3, ponieważ 1 jest względem 5 sąsiadem na skos, 5 jest względem 3 sąsiadem oraz 6 względem 8.

Listing 48: Ile występuje połączeń na skos?

```
private fun neighborDiagonally(columns: Int, code: Array<Int>): Int
    {
     var ile = 0
    for(i in 1 until code.size)
         if ((code[i] == code[i - 1] + columns + 1 && code[i]%
            columns != 1) ||
         (code[i] == code[i - 1] - columns + 1 && code[i]%columns !=
             1) ||
         (code[i] == code[i - 1] - columns - 1 && code[i]%columns !=
             0) ||
         (code[i] == code[i - 1] + columns - 1 && code[i]%columns !=
             0))
             ile++
     return ile
}
```

- lengthRelative() - zwraca liczbę punktów procentowych do odjęcia

  Funkcja sprawdza długość kodu względem planszy i na tej podstawie zwraca liczbę punktów procentowych do odjęcia. Im kod jest krótszy tym liczba punktów zwiększa się. Jest to spowodowane faktem iż np. liczba kombinacji kodu trzyelementowego jest równa 504 w planszy 3x3, natomiast kodu sześcioelementowego jest równa 60480.

Wzór przyjmuje postać:

$$p = \frac{150}{(x+y)/2}(1 - \frac{l}{x*y}) \tag{1}$$

,gdzie:
p - liczba punktów procentowych do odjęcia
x - liczba wierszy
y - liczba kolumn
l - długość kodu

Listing 49: Liczba punktów procentowych do odjęcia za długość kodu względem wielkości planszy.

```
private fun lengthRelative(rows: Int, columns: Int, code: Array<Int
    >): Int{
     val m:Double = (rows * columns).toDouble()
     val a:Double = (code.size / m)
     val x:Double = ((columns + rows)/2).toDouble()
     val b:Double = (150/x)
     return (b * (1-a)).toInt()
}
```

- shorterSide() - sprawdza czy kod jest długości krótszego boku planszy

  Jeśli kod jest długości krótszego boku planszy to od dotychczasowego punktu procentowego bezpieczeństwa hasła odejmowane jest kolejno 25 punktów procentowych.

Listing 50: Czy kod jest długości krótszego boku?

```
private fun shorterSide(rows: Int, columns: Int, code: Array<Int>):
     Boolean{
     val shorter: Int = if(rows > columns){
         columns
     } else{
         rows
     }
     return code.size == shorter
}
```

- longerSide() - sprawdza czy kod jest długości dłuższego boku planszy

  Jeśli kod jest długości krótszego boku planszy to od dotychczasowego punktu procentowego bezpieczeństwa hasła odejmowane jest kolejno 20 punktów procentowych.

Listing 51: Czy kod jest długości krótszego boku?

```
private fun longerSide(rows: Int, columns: Int, code: Array<Int>):
    Boolean{
     val short: Int = if(rows > columns){
         rows
     } else{
         columns
```

```
    }
    return code.size == short
}
```

- neighborsDifference() - zwraca liczbę punktów procentowych do odjęcia

  Funkcja na podstawie różnicy w ilości punktów, które są sąsiadami ze sobą wertykalnie, horyzontalnie lub na skos i na tej podstawie zwraca liczbę punktów proentowych do odjęcia. Im różnica w ilości obu tych zależności jest większa tym liczba punktów zwiększa się.

  Wzór przyjmuje postać:

$$p = \frac{30 * |wh - s|}{x * y} \tag{2}$$

  ,gdzie:
  p - liczba punktów procentowych do odjęcia
  x - liczba wierszy
  y - liczba kolumn
  wh - liczba połączeń wertykalnych i horyzontalnych
  s - liczba połączeń na skos

  Listing 52: Liczba punktów procentowych do odjęcia ze względu na liczbę połączeń sąsiadów.

```
  private fun neighborsDifference(wh: Int, s: Int, rows: Int,
      columns: Int): Int{
    return if(wh == 0 && s != 0 || wh != 0 && s == 0)
        (20 + abs(wh - s) * 30 / (rows * columns))
    else
        (abs(wh - s) * 30 / (rows * columns))
}
```

  Uwaga!
  Jeśli liczba połączeń jednego rodzaju jest równa 0 a drugiego rodzaju jest większa niż 0 to dodatkowo odejmuje się 20 punktów procentowych.

- verbalScaleResult() - funkcja na podstawie liczby punktów procentowych bezpieczeństwa hasła wyznacza jego nazwę

  Listing 53: Konwersja punktów procentowych na informacje zwrotną słowną.

```
fun verbalScaleResult(res: Int): PatternStrength{
    return when (res) {
        in 81..100 -> PatternStrength.VERY_STRONG
        in 61..80 -> PatternStrength.STRONG
        in 41..60 -> PatternStrength.MEDIUM
        in 21..40 -> PatternStrength.WEAK
        else -> PatternStrength.VERY_WEAK
```

```
8       }
9   }
```

- toStringConverter() - funkcja konwertuje kod podany jako tablicę int'ów w string

  Funkcja ta jest bardzo przydatna, ponieważ posiadając bazę najpopularniejszych prostych wzorów możemy w łatwy sposób porównać jej elementy do wzoru utworzonego przez użytkownika.

Listing 54: Konwersja Array<Int> -> string.

```
1  fun toStringConverter(code: Array<Int>): String{
2      var res ="["
3      for (element in code) {
4          res += element.toString()
5          res += ","
6      }
7      res = res.dropLast(1)
8      res += "]"
9      return res
10 }
```

2.3.3 Rodzaje siły wzoru (enum)

Aplikacja Bezpiecznik podaje użytkownikowi informacje jak bardzo jest jego hasło bezpieczne w dwóch wersjach. Pierwszą z nich jest wartość procentowa w skali 0-100, natomiast druga to wersja w postaci opisu jednowyrazowego. Obie te wersje są ze sobą w pełni kompatybilne a ich konwersję przedstawiono w poprzednim podrozdziale.

Kod inicjujący wersję słowną ma postać:

Listing 55: Inicjalizacja rodzajów siły wzoru.

```
1 enum class PatternStrength {
2     VERY_WEAK,
3     WEAK,
4     MEDIUM,
5     STRONG,
6     VERY_STRONG,
7 }
```

2.3.4 Kompatybilność z widokiem aplikacji

Aby przekazać użytkownikowi jak bezpieczne jest jego hasło w pliku PatternLockView.kt został umiejscowiony toast, który odpowiada za tę funkcjonalność. Przyjmuje on postać:

Listing 56: Wizualizacja wartości bezpieczeństwa hasła

```
1 val toast = Toast.makeText(context, "Pattern␣strength:␣$resPrint%␣(${
      getStrengthInString(toastStrength)})", Toast.LENGTH_SHORT)
2 toast.show()
```

,gdzie:
resPrint - wartość procentowa bezpieczeństwa hasła
toastStrength - informacja w postaci słowa

2.4 Ustawienia aplikacji

Ustawienia aplikacji to wbudowana w Androidzie funkcjonalność, którą postanowiliśmy wykorzystać, ponieważ uważamy że idealnie spełnia swą rolę w naszej aplikacji. Ustawienia pozwalają modyfikować siatkę, a także jej wygląd. Znajduje się tam również przycisk otwierający widok strony informacyjnej.
Android pozwala tu wykorzystać kilka możliwych funkcjonalności. My wykorzystaliśmy następujące:

- SeekBarPreference - suwak wyznaczający wartość całkowitą, idealnie nadał się do wyznaczania wymiarów siatki
- SwitchPreference - przycisk zmieniający wartość pomiędzy true/false, który przydał się w przypadku kilku funkcji wizualnych

Listing 57: rootpreferences.xml

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory
    app:title="Pattern settings">
    <SeekBarPreference
        android:key="col_number"
        app:title="Number of columns"
        app:showSeekBarValue="true"
        app:defaultValue="3"
        app:min="3"
        android:max="10"/>
    <SeekBarPreference
        android:key="row_number"
        app:title="Number of rows"
        app:showSeekBarValue="true"
        app:defaultValue="3"
        app:min="3"
        android:max="10"/>
</PreferenceCategory>
<PreferenceCategory
    app:title="Appearance">
    <SwitchPreference
        android:key="background"
        app:title="Fill selected cell backgrounds"
        app:defaultValue="false"/>
    <SwitchPreference
        android:key="border"
        app:title="Display border of selected cells"
        app:defaultValue="false"/>
    <SwitchPreference
        android:key="indicator"
        app:title="Display indicator of direction"
        app:defaultValue="false"/>
</PreferenceCategory>
<PreferenceCategory
    app:title="Security">
    <SwitchPreference
```

```
38              android:key="invisible_drawing"
39              app:title="Invisible drawing"
40              app:defaultValue="false"/>
41      </PreferenceCategory>
42      <PreferenceCategory
43          app:title="Info">
44          <Preference
45              android:title="About app"
46              android:key="about_btn"
47              android:summary="open about fragment"/>
48      </PreferenceCategory>
49  </PreferenceScreen>
```

Z ustawień wartości pobieraliśmy w momencie otwierania widoku testów, czyli właściwie tuż przed wygenerowaniem siatki. Pozwalało to dynamicznie zmieniać ustawienia, a siatka zawsze posiada taki wygląd, jaki nadaliśmy jej w ustawieniach.

Listing 58: funkcja onViewCreated wykonująca się po otwarciu danego fragmenu

```
1     override fun onViewCreated(view: View, savedInstanceState: Bundle?)
          {
2         super.onViewCreated(view, savedInstanceState)
3
4         val sp = PreferenceManager.getDefaultSharedPreferences(context)
5
6         val col = sp.getInt("col_number",3)
7         val row = sp.getInt("row_number",3)
8         val background = sp.getBoolean("background",false)
9         val border = sp.getBoolean("border",false)
10        val indicator = sp.getBoolean("indicator",false)
11        val invisibleDrawing = sp.getBoolean("invisible_drawing", false)
12
13        pattern_lock_id.columnCount = col
14        pattern_lock_id.rowCount = row
15        pattern_lock_id.patternColCount = col
16        pattern_lock_id.patternRowCount = row
17        pattern_lock_id.showCellBackground = background
18        pattern_lock_id.showBorder = border
19        pattern_lock_id.showIndicator = indicator
20        pattern_lock_id.invisibleDrawing = invisibleDrawing
21
22        pattern_lock_id.reset()
23        pattern_lock_id.removeAllViews()
24        pattern_lock_id.initDots()
25    }
```

2.5 Nawigacja w aplikacji

Dolne menu zdefiniowane jest w pliku xml, przypisujemy tam odpowiednie fragmenty oraz ich ikony i podpisy. Jest go główny sposób nawigacji po aplikacji. Z racji na niewielką ilość widoków, w pełni wystarcza, a przy tym zapewnia schludny wygląd oraz ogromną wygodę.

Listing 59: menu.xml

```
1  <menu xmlns:tools="http://schemas.android.com/tools"
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    tools:ignore="ExtraText">
    <item
        android:id="@+id/testsFragment"
        android:icon="@drawable/ic_baseline_apps_24"
        android:title="@string/tests"/>
    <item
        android:id="@+id/historyFragment"
        android:icon="@drawable/ic_baseline_history_24"
        android:title="@string/history" />
    <item
        android:id="@+id/settings2Fragment"
        android:icon="@drawable/ic_baseline_settings_24"
        android:title="@string/settings" />
</menu>
```

Menu podpinamy w głównym pliku xml odpowiadającym za layout aplikacji.

Listing 60: fragment pliku activitymain.xml

```
    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottomNavigationView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:textAlignment="center"
        app:menu="@menu/menu" />
```

2.6 Zastosowane narzędzia

- Android Studio - implementacja programu
- Android Emulator - debugowanie oraz testowanie programu
- Telefon z androidem - testowanie programu w rzeczywistych warunkach
- Postman - testowanie zdalnego repozytorium
- Google Docs - tworzenie notatek, prezentacji
- Microsoft Excel - prowadzenie zestawienia godzinowego pracy
- www.clickup.com - narzędzie do harmonogramowania
- www.overleaf.com - tworzenie dokumentacji

2.7 Lista używanych bibliotek

- kotlin.math - biblioteka zawierająca funkcje matematyczne takie jak: trygonometrię, hiperbolikę lub logarytmikę.
  Nasza aplikacja wykorzystuje takie funkcje jak:
  - atan2 - zwraca kąt (w radianach) utworzony przez oś OX i prostą przechodzącą przez punkt o podanych współrzędnych.

  - sqrt - zwraca wynik pierwiastkowania
  - abs - zwraca wartość bezwzględną
  - min - zwraca minimalną wartość z podanych

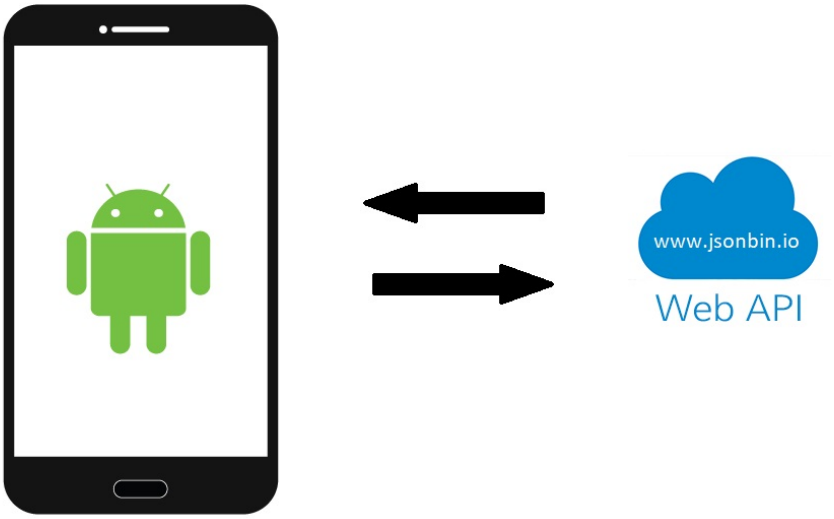
- Java.lang.Math - również biblioteka zawierająca funkcje matematyczne.
  Nasza aplikacja w jej przypadku korzysta tylko z funkcji toDegrees

- retrofit2 - najpopularniejszy klient http na androida, jest bardzo elastyczny i oferuje szeroki wachlarz funkcji, takich jak np rożne Json Parsery, Gson, Jackson. Utworzona przez firmę Square Inc. Link: https://github.com/square/retrofit
  Nasza aplikacja wykorzystuję podstawowego kilenta http do komunikacji z api oraz Gson do odczytwyania odpowiedzi.

- java.security.SecureRandom - służy do generowania silnych kryptograficznie liczb losowych

2.8 Dobór wzorców architektonicznych oprogramowania

2.8.1 Schemat graficzny struktury systemu



2.9 Testowanie oprogramowania

2.9.1 Raport pisemny z przeprowadzonych testów ze statystyką znalezionych błędów

Testowanie aplikacji zostało przeprowadzone za pomocą testowania manualnego. Zgromadzono grupę 15 osób złożoną z twórców aplikacji, ich rodzin oraz znajomych. Warto również wspomnieć, że testy te zaczęliśmy wykonywać już jakiś czas temu, aby dać sobie bufor czasu na naprawę znalezionych błędów oraz poprawę jakości działania aplikacji.
Konkretny system jaki obraliśmy - to dostarczanie naszym testerom co nowszych wersji aplikacji, które tworzyliśmy pozbywając się wszystkich błędów, które zostały zgłoszone, jak i tych które sami zauważyliśmy.

- Pierwsza sesja testów
  Znalezione błędy:
  - Problem z widokiem, którego nie zauważyliśmy, a zgłosiła jedna z testujących osób. Widok komórek w momencie, gdy liczba kolumn była mniejsza niż liczba wierszy nie mieścił się na ekranie. Było to spowodowane skalowaniem tylko ze względu na kolumny, przez co aplikacja nie brała pod uwagę takiego przypadku. Zostało to uzupełnione, w momencie gdy liczba kolumn jest mniejsza niż liczba wierszy, skalowanie odbywa się ze względu na wiersze i wysokość, a następnie zostaje dodany margines z lewej strony dla komórek przy lewej krawędzi, aby wycentrować komórki na ekranie.
  - Problem z asynchronicznym pobieraniem danych. Dane które chcieliśmy wykorzystać nie były jeszcze w pełni załadowane z zasobu sieciowego co wymusiło używanie tzw „callbacków".
- Druga sesja testów
  - Funkcje zdalnego repozytorium które miały zostać wykorzystane w aplikacji nie działały zadowalająco, czas odpowiedzi był bardzo długi a czasami jako odpowiedź przychodził nieznany błąd. Wymusiło to zmiany w komunikacji ze zdalnym repozytorium, na szczęście problem udało się rozwiązać.
- Trzecia sesja testów
  - Problem, który zgłosiła jedna z osób testujących - brak komunikatu o sile hasła po narysowaniu wzoru. Na szczęście okazało się że problem nie leżał w kwestii aplikacji, lecz osoba ta miała wyłączone powiadomienia dla tej aplikacji
  - Problem z komunikacją z API(aktualizacja statystyk, odczyt statystyk) zgłoszony przez jedną z testujących osób. Powodem okazała się zbyt stara wersja Androida, która już nie jest przez nas obsługiwana.

Sesja	Ilość błędów	Naprawione	Nie do rozwiązania
1	2	2	0
2	1	1	0
3	2	0	2
Suma	5	3	2

Po tych przeprowadzonych testach, wszystkie znalezione błędy, które posiadały możliwe rozwiązanie, zostały wyeliminowane.
Bardzo dziękujemy osobom biorącym udział w testach. Uważamy, że ich wkład bardzo przyczynił się do poprawnego działania naszej aplikacji oraz jej niezawodności.

3 Podsumowanie

Aplikacja Bezpiecznik spełnia podstawowe założenia. Dzięki użyciu architektury MVVM kod aplikacji jest przejrzysty i łatwo się w nim odnaleźć. Mnogość narzędzi realizujących podobne założenia co nasza aplikacja pozwoliło nam spojrzeć na konkretne elementy z kilku stron.

W przyszłości aplikacja mogłaby się rozwinąć, np. o następujące funkcjonalności:

- Zmiana języka aplikacji
- Dodatkowe funkcje weryfikujące bezpieczeństwo wzoru
- Generowanie raportów w wielu formatach oraz wysyłanie ich np. za pomocą maila
- Podgląd ścieżki w statystykach
- Możliwość edycji konta użytkownika
- Przenoszenie konta użytkownika pomiędzy urządzeniami
- Rozwinięcie konta użytkownika poprzez dodanie większej ilości jego charakterystyk
- Automatyczne generowanie wybranej jakości wzoru
- Własne API

4 Odnośniki

- Github - repozytorium, zawiera również zestawienie godzinowe
  `https://github.com/ketiovv/Bezpiecznik`
- ClickUp - harmonogramowanie
  `https://share.clickup.com/b/h/5-16669515-2/e1f4172f52ff824`