# Healthy+

Съдържание

# Какво е Healthy+

Healthy+ е приложение за проверяване съставките на лекарствата. Приложението е създадено за информиране на потребителя за всички потенциални опасни съставки, които могат да се съдържат в лекарството, което те приемат. Как става това?

1. Потребителя инсталира мобилното приложение на телефона си
2. Потребителя създава акаунт в приложението
3. В началното меню на потребителския акаунт присъства бутон за сканиране на баркода на лекарствата
4. След активирането на тази функция потребителя бива пренасочен към скенера на телефона
5. След успешно сканиране на баркода, ако лекарството присъства в базата данни се извлича цялата информация за него като име и съставки, всяка съставка се проверява и се изобраяват на екран всички съставки и дали те са здравословни, потенциални алергени или са опасни

Приложението е предназначено за всеки потребител, който се интересува за здравето си и иска да бъде по информиран какво приема.

Приложението е с интуитивен интерфейс, което позволява лесно ориентиране и разбиране логиката на работа.

# Структура на Healthy+

## Начало на приложението

Приложението е изградено на python използвайки фрейморка kivymd. Началото на приложението започва от main.py файла който подканва потребителите да продължът по напред.


В началото на приложението съдържа заглавие, което е логото на приложението, информация относно дейността на приложението и бутон, който подканва потребителя да продължи по-напред.

За създаване на тази страница се използва клас за заден фон, който се намира в файла color_overlay.py,

## ColorOverlay

```
from kivy.uix.widget import Widget
from kivy.graphics import Color, Rectangle


class ColorOverlay(Widget):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        with self.canvas:
            self.color_instruction = Color(0.058, 0.227, 0.212, 1)
            self.rect = Rectangle(pos=self.pos, size=self.size)

        self.bind(pos=self.update_rect, size=self.update_rect)

    def update_rect(self, *args):
        self.rect.pos = self.pos
        self.rect.size = self.size

    def set_color(self, r, g, b, a=1):
        self.color_instruction.r = r
        self.color_instruction.g = g
        self.color_instruction.b = b
        self.color_instruction.a = a
```

Класът работи на следния принцип наследява Widget(), за да можем да създадем елемента. В конструктора на класа задаваме цвета и правоъгълника неговата позиция и размери.

```
def __init__(self, **kwargs):
        super().__init__(**kwargs)
        with self.canvas:
            self.color_instruction = Color(0.058, 0.227, 0.212, 1)
            self.rect = Rectangle(pos=self.pos, size=self.size)
```

Методът bind() е от ключово значение за автоматичното преоразмеряване като свързва свойствата pos и size с функцията update_react() всеки път когато се направи промяна по позицията или размера на елемента той извиква метода update_rect(), който преоразмерява правоъгълника спрямо екрана.
Метода set_color() се използва за задаване на нов цвят на правоъгълника.

За изграждане на логото се използва класът за заглавие който е от фреймуърка kivymd, а именно Label(), и класа за създаване плюсът plus_logo.py.

```
self.textLogo = Label(
    text="Healthy",
    font_name="PoppinsBold",
    font_size="42",
    color=(1, 1, 1, 1),
    pos_hint={"center_x": 0.47, "center_y": 0.66},
)
```

Променливата textLogo присвоява класа Label() на kivymd в класа са променени няколко параметри като text, който да бъде изобразен на екрана font_name типът на шрифта, font_size размерът на шрифта, color цвета на текста и pos_hint позицията на текста.

```
plus = DoublePlus(
    size_hint=(None, None),
    size=(43, 43),
    pos_hint={"center_x": 0.77, "center_y": 0.66},

    # Голям плюс
    big_width=41,
    big_height=41,
    big_thickness=8,
    big_radius=2.5,
    big_color=(1, 1, 1, 1),

    # Малък плюс
    small_width=32,
    small_height=32,
    small_thickness=3,
    small_radius=1,
    small_color=(0.058, 0.227, 0.212, 1),
)
```

Класът за плюсът в логото е създаден и работи по следния начин задават се размери от size свойството и позиция от pos_hint свойството. Плюс логото е създадено от два плюса един по-малък и един по-голям, за да се получи този естетичен вид. И двата плюса имат еднакви свойства с единствената отговарящи за големината, визочината, дебелината, заоблянето на ъглите, цвета (*_width, *_height, *_thickness, *_radius, *_color).

DoublePus

```
class DoublePlus(Widget):
    def __init__(
```

...

```
with self.canvas:
    # Голям плюс
    Color(*self.big_color)
    self.h_rect_big = RoundedRectangle()
    self.v_rect_big = RoundedRectangle()

    # Малък плюс
    Color(*self.small_color)
    self.h_rect_small = RoundedRectangle()
    self.v_rect_small = RoundedRectangle()
```

Класът наследява класът Widget и двата плюса са създадени с помощта на класа от kivymd RoundedRectangle() като се поставят на кръст.

```
self.bind(pos=self.update_rects, size=self.update_rects)

def update_rects(self, *args):
    # Център на widget-а от pos и size
    cx = self.x + self.width / 2
    cy = self.y + self.height / 2

    # Голям плюс
    bw, bh = self.big_width, self.big_height
    bt = self.big_thickness
    br = self.big_radius

    self.h_rect_big.pos = (cx - bw / 2, cy - bt / 2)
    self.h_rect_big.size = (bw, bt)
    self.h_rect_big.radius = [(br, br)] * 4

    self.v_rect_big.pos = (cx - bt / 2, cy - bh / 2)
    self.v_rect_big.size = (bt, bh)
    self.v_rect_big.radius = [(br, br)] * 4
```

```
    # Малък плюс - центриран спрямо големия
    sw = self.small_width
    sh = self.small_height
    st = self.small_thickness
    sr = self.small_radius

    self.h_rect_small.pos = (cx - sw / 2, cy - st / 2)
    self.h_rect_small.size = (sw, st)
    self.h_rect_small.radius = [(sr, sr)] * 4

    self.v_rect_small.pos = (cx - st / 2, cy - sh / 2)
    self.v_rect_small.size = (st, sh)
    self.v_rect_small.radius = [(sr, sr)] * 4
```

Отново за автоматично преоразмеряване и изчертаване на плюсът в логото се използва bind(), който свързва свойствата pos и size с метода update_rects().

```
self.info = Label(
    text="Your Pocket Pharmacist",
    font_name="PoppinsSemiBold",
    font_size="21",
    color=(1, 1, 1, 1),
    pos_hint={"center_x": 0.5, "center_y": 0.45},
)
self.layout.add_widget(self.info)

self.info1 = Label(
    text="Healthy+ makes medication\n"
         "  safety simple. Scan. Learn.\n"
         "               Stay protected.",
    font_name="PoppinsSemiBold",
    font_size="13",
    color=(1, 1, 1, 1),
    pos_hint={"center_x": 0.5, "center_y": 0.36},
)
self.layout.add_widget(self.info1)
```

Относно информацията се използва вече вградения клас Label със същите параметри но различни стойности, които подчертават, че става дума за информация под заглавие, а именно текстът, размерът и позицията (text, font_size, pos_hint).

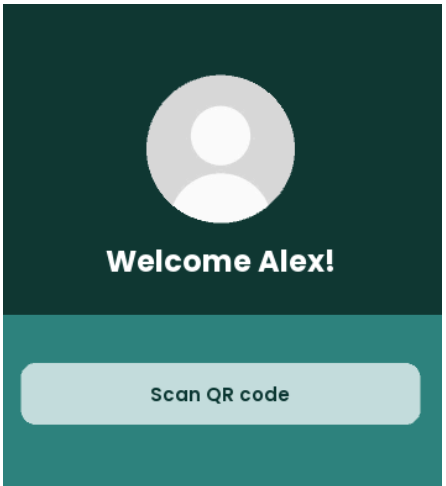
```
self.card_button = MDCard(
        size_hint=(None, None),
        size=(323, 50),
```

```
        pos_hint={"center_x": 0.5, "center_y": 0.12},
        radius=[12],
        elevation=0,
        md_bg_color=(1, 1, 1, 0.73),
    )

    # Бутон
    self.button = MDFlatButton(
        text="Get Started",
        theme_text_color="Custom",
        text_color=(0.058, 0.227, 0.212, 1),
        font_size="15",
        font_name="PoppinsSemiBold",
        size_hint=(1, 1),
        on_release=self.on_button_click
    )
    self.card_button.add_widget(self.button)
    self.layout.add_widget(self.card_button)

def on_button_click(self, instance):
    self.manager.current = "dashboard"
```

Този код показва стилизирането на бутона чрез card_button() и самият бутон. Пропъртито card_button се създава чрез класа MDCard(), който е вграден в kivymd дава размер на бутона позиция, заобляне на ъглите за по-хубав вид, премахване на сянката и бял полупрозрачен цвят (size, pos_hint, radius, elevation, md_bg_color). Чрез каса на kivymd MDFlatButton се създава бутона който препраща към следващия екран на приложението при натискане на бутона (пропъртито on_release извиква функцията on_button_click(), която се грижи за това).

## Табло за управление

Кодът за таблото на потребителя се намира във файла dashboard.py. В тази се намира профилна снимка на потребителя, персонално поздравление към него и бутонът който активира скенера на мобилното приложение.




Уиджета за профилна снимка е клас, който използва FloatLayout() класа на kivymd. Класа работи по следния начин задава се път към профилната снимка на потребителя, цвят на кръга, ширина на

рамката, и цвят на рамката. Конструктура изгражда обект със параметри по подразбиране. Всички свойства са свързани със функции, които ги автоматично преоразмеряват спрямо екрана.

```
class ProfilePictureCircle(FloatLayout):
   profile_image_source =
StringProperty('C:/Users/User/PycharmProjects/Healthy+/img/blank_profile.jpg'
)
   circle_color = ListProperty([1, 1, 1, 1])
   border_width = NumericProperty(0)
   border_color_rgba = ListProperty([0, 0, 0, 1])

   def __init__(self, **kwargs):
       super().__init__(**kwargs)

           with self.canvas:
               self.border_color_instruction = Color(*self.border_color_rgba)
               self.border_ellipse = Ellipse(pos=self.pos,
size=self.size)
               self.color_instruction = Color(*self.circle_color)
               self.circle_ellipse = Ellipse(pos=self.pos,size=self.size,
source=self.profile_image_source)
```

Конструктура на класа със свойствата по подразбиране.

…

```
self.bind(pos=self.update_circle, size=self.update_circle)
self.bind(profile_image_source=self.update_image_source)
self.bind(circle_color=self.update_circle_color)
self.bind(border_width=self.update_border,
border_color_rgba=self.update_border_color)

def update_circle(self, *args):
    self.circle_ellipse.pos = (self.pos[0] + self.border_width, self.pos[1] +
self.border_width)
   self.circle_ellipse.size = (self.size[0] - 2 * self.border_width,
self.size[1] - 2 * self.border_width)

   self.border_ellipse.pos = self.pos
   self.border_ellipse.size = self.size

def update_image_source(self, instance, value):
    self.circle_ellipse.source = value

def update_circle_color(self, instance, value):
    self.color_instruction.rgba = value

def update_border(self, instance, value):
    self.update_circle()

def update_border_color(self, instance, value):
    self.border_color_instruction.rgba = value
```

Всеки bind() свързва едно или две свойства с една или две функци свойствата pos и size са свързани с функцията отговаряща за автоматичното преоразмеряване на позицията и размера, а именно updata_circle() преоразмеряват се кръгът и рамката му. Тези редове изчисляват и задават новата позиция и размер на вътрешната кръгова част (self.circle_ellipse), като взимат предвид self.border_width. Това гарантира, че кръгът винаги остава центриран и с правилния размер спрямо границите на уиджета и ширината на рамката.
Методът update_image_source() преоразмерява снимката намираща се в кръгът.
Методите update_*_color() се грижи за автоматичното преоразмеряване на цвета на кръга и рамката.
Методът update_border_color() се използва за удобно задаване на рамката.

```
def set_profile_image(self, image_path):
    self.profile_image_source = image_path

def set_circle_color(self, rgba_list):
    self.circle_color = rgba_list

def set_border(self, width, rgba_list):
    self.border_width = width
    self.border_color_rgba = rgba_list
```

Класът притежава публични методи за промяна на свойствата.
Методът set_profile_image() дава възможност за добавяне на нова снимка чрез промяна на пътя.
Методът set_circle_color() променя цветът на основния кръг.
Методът set_border() промяна на цвета и широчината на рамката на крага.

```
overlay = ColorOverlay(size_hint=(1, 1), pos_hint={"center_x": 0.5,
"center_y": 0.5})
overlay.set_color(0.176, 0.525, 0.498, 1)
self.layout.add_widget(overlay)
```

За екрана се използва обект на ColorOverlay() използва се метода за промяна на цвета му set_color().

```
square = SquareWidget(
    size_hint=(None, None),
    size=(360, 259)
)

top_container = MDBoxLayout(
    orientation='horizontal',
    pos_hint={"center_x": 0.5, "center_y": 1.1},
    height=259,
    padding=[0, 0, 0, 0]
)

top_container.add_widget(square)
self.layout.add_widget(top_container)
```

Класът MDBoxLayout() създава променлива, която служи за рамка на SquareWidget(). Променя позицията му височината и отместванията (orientation, pos_hint, height, padding)

…

```
from kivy.uix.widget import Widget
from kivy.graphics import Color, Rectangle


class SquareWidget(Widget):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        with self.canvas:
            Color(0.0588, 0.2275, 0.2118, 1)
            self.rect = Rectangle(pos=self.pos, size=self.size)
        self.bind(pos=self.update_rect, size=self.update_rect)

    def update_rect(self, *args):
        self.rect.pos = self.pos
        self.rect.size = self.size
```

Класът SquareWidget() създава малък квадрат с различен цвят от фона за по завършен вид и служи, за да изпъкне профилната снимка на потребителя и персоналното му поздравление.

```
label = Label(
    text="Welcome Alex!",
    font_name="PoppinsBold",
    font_size='24sp',
    halign='center',
    valign='middle',
    pos_hint={"center_x": 0.5, "center_y": 0.67},
)
```

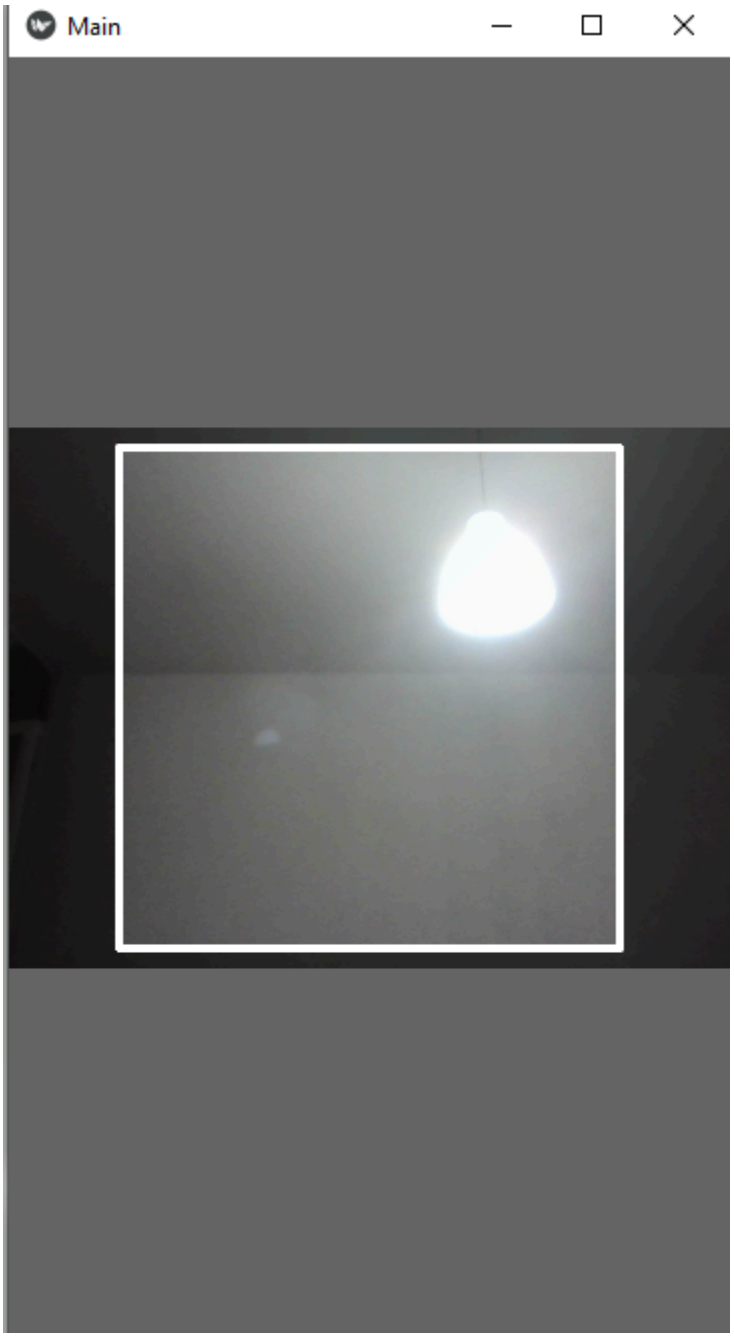
Персоналното поздравление, което е позиционирано под профилната снимка. Създава се чрез класа Label() и са зададени следните параметри съдържание на текст, тип на фонта, размер на фонта, хоризонтално и вертикално подравняване и позиция (text, font_name, font_size, halign, valign, pos_hint).

```
card_button = MDCard(
        size_hint=(None, None),
        size=(323, 50),
        pos_hint={"center_x": 0.5, "center_y": 0.5},
        radius=[12],
        elevation=0,
        md_bg_color=(1, 1, 1, 0.73),
    )

    scan_button = MDFlatButton(
        text="Scan QR code",
        theme_text_color="Custom",
        text_color=(0.058, 0.227, 0.212, 1),
        size_hint=(1, 1),
        size=(200, 50),
        pos_hint={"center_x": 0.5},
        font_size="15",
        font_name="PoppinsSemiBold",
        on_release=self.go_to_qr_scan,

    )
    card_button.add_widget(scan_button)

    self.layout.add_widget(card_button)

def go_to_qr_scan(self, instance):
    self.manager.current = 'qrread'
```

За бутона се създава рамка са стил чрез MDCard() за създаване на стилна рамка с определен размер позиция, заобленост на ъглите без сянка и фон (size, pos_hint, radius, elevation, md_bg_color). Бутонът се създава чрез MDFlatButton() задава се текст на бутона, посочва се че темата не е по подразбиране, задава се цвят на

текста, запълва се рамката, задава се размер, подава се позиция, размер на шрифта и тип на шрифта и дава указание при натискане коя функция да изпълни(text, theme_text_color, text_color, size_hint, size, pos_hint, font_size, font_name,on_release). Функцията при натискане на бутона go_to_qr_scan() превключва към следващия екран, който е QR (баркод) четеца.

## QR четец




Файлът, в който се намира логиката и функционалността на процеса по четене на QR (Баркодове) се намира в Python файла qread_screen.py. За да се постигне визуалната естетика се използва клас, който изгражда полупрозрачна черна рамка с отвор посредата, където няма замъгляване (Това е мястото, където трябва да се постави баркода). В крайщата на отвора се очертава бяла рамка.

FocusOverlay

```
class FocusOverlay(StencilView):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.bind(pos=self.redraw, size=self.redraw)

    def redraw(self, *args):
        self.canvas.clear()
        with self.canvas:
            cx, cy = self.center
            w, h = 250, 250
            x_hole = cx - w / 2
            y_hole = cy - h / 2

            Color(0, 0, 0, 0.6)
            Rectangle(pos=(0, y_hole + h), size=(self.width, self.height -
(y_hole + h)))
            Rectangle(pos=(0, 0), size=(self.width, y_hole))
            Rectangle(pos=(0, y_hole), size=(x_hole, h))
            Rectangle(pos=(x_hole + w, y_hole), size=(self.width - (x_hole +
w), h))

            Color(1, 1, 1, 1)
            Line(rectangle=(x_hole, y_hole, w, h), width=2, cap='round')
```

Класът FocusOverlay() присвоява уиджета StencilView() този клас е вграден в kivymd и позволява изчертаването на полупрозрачния фон и отворът. Конструктура на класът присвоява базата на родителския клас и обвързваме свойствата за промяна на позицията и размера с метода redraw(). Методът работи по следния начин. Винаги изчиства платното за да не се получава наслояване на фонове с отвори при всяко преместване или преоразмеряване. Методът изчертава рамката на платното на уиджета. Променливите cx и cy взимат центъра на уиджета, а променливите w и h служат за определяне на размерите на прозорчето, което няма да е затъмнено. Променливите x_hole и y_hole определят кординатите къде да се постави този прозорец, за да е точно в центъра. Color() задава текущия цвят на рисуване. Методът на рисуване е следния, вместо да се чертае един голям полупрозрачен черен фон и след това да се изрязва по средата се създават четири правоъгълника за лява, дясна, горна и долна част, които оставят дупка по средата,

която е именно прозорецът. Color() задава бял цвят на рамката на прозореца, а Line() изчертава тази рамка.

### QRReadScreen

```
class QRReadScreen(Screen):
    _zbarcam_instance = None

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.layout = FloatLayout()
        self.add_widget(self.layout)

        self.spinner = MDSpinner(
            size_hint=(None, None),
            size=(64, 64),
            pos_hint={"center_x": 0.5, "center_y": 0.5},
            active=True
        )
        self.layout.add_widget(self.spinner)

        self.error_label = MDLabel(
            text='',
            halign='center',
            valign='center',
            font_style='H6',
            color=(1, 0, 0, 1),
            size_hint=(0.8, None),
            height='96dp',
            pos_hint={"center_x": 0.5, "center_y": 0.2},
            opacity=0
        )
        self.layout.add_widget(self.error_label)

        self.overlay = None
        self.scanning = False
```

Класът QRReadScreen() присвоява функционалностите от базовия клас Screen(), чиято основна функционалност е да използва ZBarCam външна библиотека за управление на камерата и четене на QR кодове. Логиката на класа е да управлява камерата на устройството, да чете QR кодове и да препраща прочетения код към следващия екран има и функционалност за зареждане и за отпечатване на грешки при проблем с камерата или QR четеца. Класът използва само една инстанция към камерата _zbarcam_instance=None и само към нея се обръща всеки път,

когато се активира четеца на QR кодове. Това предотвратява от грешки и колизи поради създаването всеки път на нови инстанции при стартиране на четеца. Конструктура създава шаблон, на когото да се добавят уиджети. Първият уиджет, който се изобразява е зареждащия спинер MDSpiner() от kivymd, който се върти докато се стартира камерата. Създава се MDLabel() error_label, който визуализура грешки от камерата или четеца. Променливата overlay служи за създаването на обект на FocusOverlay().Променливата scanning е флаг, който отчита дали програмата да сканира или не.

```
def on_enter(self):
    Logger.info("QRReadScreen: Entering screen. Preparing camera.")
    self.scanning = True
    self.spinner.opacity = 1
    self.spinner.active = True
    self.error_label.opacity = 0

    Clock.schedule_once(self.start_camera_after_delay, 0.5)
```

Методът on_enter() се извиква всеки път, когато се стартира qreader.py казано по друг начин когато потребителя натисне бутона в таблото за управление за сканиране на QR код. В този метод се отпечатва съобщение, което показва в какъв етап се намира зареждането на четеца. Флагът за сканиране се вдига (става True), което сигнализира, че приложението започва да сканира. За да се изчертае движещия спинер той се открива self.spnner.opacity = 1 и се активира self.spinner.active = True. Етикета за грешки се скрива self.error_label.opacity = 0. Класът Clock(), който се импортира от kivy.clock служи за отложен старт на камерата, а именно да се извика метода start_camera_after_delay() след 0.5 секунди.

```
def start_camera_after_delay(self, dt):
    if not self.scanning:
        Logger.info("QRReadScreen: Camera startup interrupted (screen left).")
        return

    if QRReadScreen._zbarcam_instance is None:
        Logger.info("QRReadScreen: Creating new ZBarCam instance (first
start).")
        try:
            QRReadScreen._zbarcam_instance = ZBarCam(size_hint=(1, 1), pos=(0,
0),)
            self.ids['xcamera'] = QRReadScreen._zbarcam_instance
        except Exception as e:
            Logger.exception("QRReadScreen: Failed to create ZBarCam instance:
%s", e)
            self.error_label.text = ("Failed to start camera.\n"
                                     "Please check if the camera is in use by
another app,\n"
                                     "privacy settings, or drivers.")
            self.error_label.opacity = 1
            self.spinner.active = False
            self.spinner.opacity = 0
            self.scanning = False
            return
    else:
        Logger.info("QRReadScreen: Using existing ZBarCam instance.")
        self.ids['xcamera'] = QRReadScreen._zbarcam_instance
```

…

Методът start_camera_after_delay() започва с проверка на флага за сканиране. Ако флагът за сканиране е вдигнат то кодът минава към създаването на единична връзка с камерата. Ако не съществува връзка с камерата (приложението е стартирано за първи път) чрез try кодът се опитва да създаде връзка, която да се използва през целия живот на приложението, докато не се изключи. Променливата на класа _zbarcam_instance служи за съхранение на тази връзка, а self.ids[‘xcamera’] е уиджета на kivy, който отговаря за управлението на камерата (създаване, четене на QR чрез OpenCV, освобождаване на ресурсите и свойства, които позволяват да се настрои според предпочитанията). При поява на грешка except улавя грешгите и и ги логва на конзола както и error_label присвоява

текст, който да се изобрази на потребителя при грешка. При използване на четеца всеки следващ път се присвоява вече създадената инстанция към уиджета self.ids[‘xcamera’] това гарантира че няма да създаваме повече от една инстанция, което да доведе до грешки и бъгове.

```
QRReadScreen._zbarcam_instance.unbind(symbols=self.on_symbols)
QRReadScreen._zbarcam_instance.bind(symbols=self.on_symbols)
```

Логиката за свързване на прочетеното от камерата със обработването му става чрез unbind() и bind() в случая unbind() гарантира, че ще има само един слушател, който да обработи прочетеното, а bind() му задава задачата.

```
if QRReadScreen._zbarcam_instance.parent is None:
    self.layout.add_widget(QRReadScreen._zbarcam_instance, index=0)
    Logger.info("QRReadScreen: ZBarCam added to layout.")
else:
    Logger.info("QRReadScreen: ZBarCam is already in the layout.")

if self.overlay is None:
    self.overlay = FocusOverlay(size_hint=(1, 1), pos=(0, 0))
    self.layout.add_widget(self.overlay)
    Logger.info("QRReadScreen: FocusOverlay successfully added.")
self.overlay.opacity = 0

Animation(opacity=1, duration=0.5).start(QRReadScreen._zbarcam_instance)
Animation(opacity=1, duration=0.5).start(self.overlay)

spinner_anim = Animation(opacity=0, duration=0.5)
spinner_anim.bind(on_complete=lambda animation, widget: setattr(widget,
'active', False))
spinner_anim.start(self.spinner)
```

Първият if проверява дали потокът на камерата е добавен към layout (тоест потребителя да види потокът на устройството си от камерата в реално време). След поставянето на потока в layout се поставя и рамката, която показва къде трябва да се постави QR кодът и замъгляващия ефект покраищата на просорчето. Добавя се анимация на потока и на рамката като придава по завършен вид на стартиране на процеса. След което спинерът се изключва от екрана на потребителя.

```
def on_symbols(self, instance, symbols):
    if not self.scanning:
        return

    if symbols:
        try:
            qr_text = symbols[0].data.decode("utf-8")
            Logger.info(f"QRReadScreen: Scanned QR code: {qr_text}")
        except Exception as e:
            Logger.error(f"QRReadScreen: Error decoding QR: {e}")
            self.error_label.text = "Error decoding QR code."
            self.error_label.opacity = 1
            return

        self.scanning = False
        self.hide_camera_for_reuse()

        # --- Логика за подаване на кода ---
        # Изпраща сканирания код до ResultScreen
        if self.manager and self.manager.has_screen("result"):  # Таргитира се
'result' екрана
            result_screen = self.manager.get_screen("result")
            if hasattr(result_screen, 'show_result'):
                result_screen.show_result(qr_text)
            else:
                Logger.warning("QRReadScreen: ResultScreen does not have a
show_result method.")
            self.manager.current = "result"  # Навигира до 'result' екрана
        else:
            Logger.error("QRReadScreen: ResultScreen not found or manager not
set.")
        # --- Край на логиката за подаване ---
```

Функцията се грижи за правилното прочитане на QR кода спиране на сканирането self.scaning=False скриване на камерата за повторно използване hide_camera_for_reuse() и подаване на прочетения код към функция на ResultScreen show_result() и навигирането към result екрана.

```
def hide_camera_for_reuse(self):
    Logger.info("QRReadScreen: Hiding camera for reuse.")

    if QRReadScreen._zbarcam_instance:
        if QRReadScreen._zbarcam_instance.parent:

QRReadScreen._zbarcam_instance.parent.remove_widget(QRReadScreen._zbarcam_ins
tance)
            Logger.info("QRReadScreen: ZBarCam removed from layout (hidden).")

        try:
            Logger.info("QRReadScreen: Attempting to call _real_stop() on
ZBarCam (hidden).")
            if hasattr(QRReadScreen._zbarcam_instance, '_real_stop') and
callable(
                    QRReadScreen._zbarcam_instance._real_stop):
                QRReadScreen._zbarcam_instance._real_stop()
                Logger.info("QRReadScreen: _real_stop() successfully called
(hidden).")

            Logger.info("QRReadScreen: Attempting to call _camera.release() on
ZBarCam (hidden).")
            if hasattr(QRReadScreen._zbarcam_instance, '_camera') and \
                    hasattr(QRReadScreen._zbarcam_instance._camera, 'release')
and \
                    callable(QRReadScreen._zbarcam_instance._camera.release):
                QRReadScreen._zbarcam_instance._camera.release()
                Logger.info("QRReadScreen: _camera.release() successfully
called (hidden).")

        except Exception as e:
            Logger.error(f"QRReadScreen: Error explicitly stopping camera when
hiding: {e}")

        QRReadScreen._zbarcam_instance.unbind(symbols=self.on_symbols)
        Logger.info("QRReadScreen: ZBarCam unbinded.")

    if self.overlay:
        if self.overlay.parent:
            self.layout.remove_widget(self.overlay)
        self.overlay = None
        Logger.info("QRReadScreen: FocusOverlay removed.")

    self.spinner.opacity = 0
    self.spinner.active = False
    self.error_label.opacity = 0
    Logger.info("QRReadScreen: Spinner and error hidden.")
    Logger.info("QRReadScreen: Camera hiding complete (for reuse).")
```

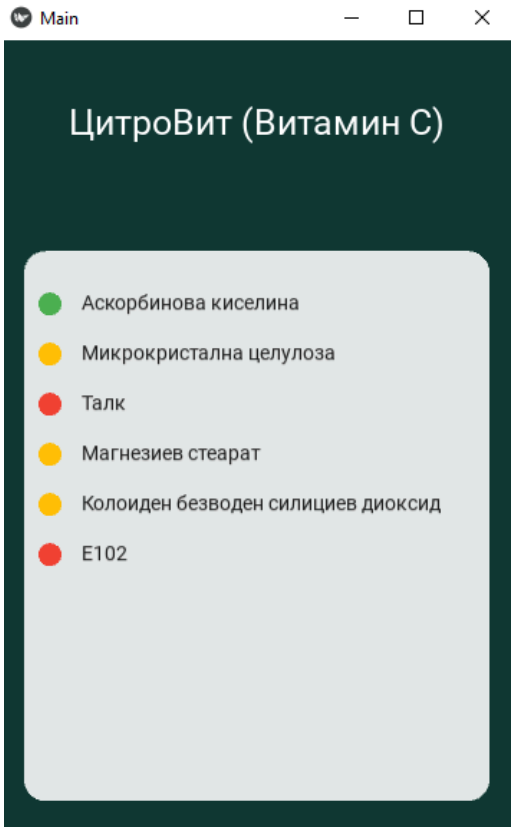
Методът премахва камерата от лейаута чрез remove_widget() след което се опитва да освободи хардуерните ресурси на камерата чрез вградените методи на ZBarCam _real_stop() и realease() при

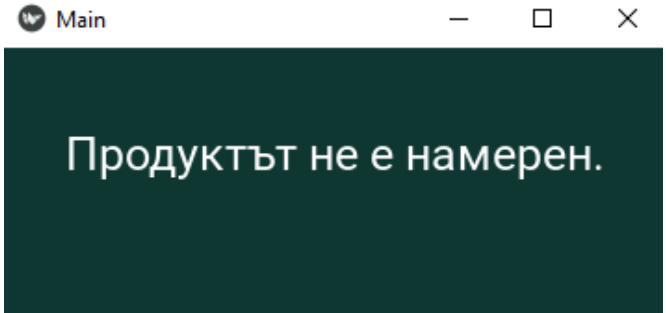
неуспешно освобождаване грешките се хващат от except и се сигнализира за тях. Програмата развързва методът за четене на символи (това предотвратява четене на символи докато камерата е неактивна) чрез unbind(). Изчистват се от лейаута също и рамката self.overlay и спинера self.spinner.

```
def on_leave(self):
    Logger.info("QRReadScreen: Leaving screen. Hiding camera.")
    self.scanning = False
    self.hide_camera_for_reuse()
```

Този метод on_leave() се извиква винаги, когато се напусне екрана и се премине към друг. Той сигнализира на да се спре сканирането чрез self.scaning=False и да се освободят ресурсите на камерата чрез self.hide_camera_for_reuse() гарантира освобождаване дори, когато QR код не е сканиран.

## Показване на резултат




Кодът, който извежда ресултатът се намира във файла result_screen.py. Логиката и функционалността на кодът е следната.

1. След успешно сканиране от QRReadScreen() кодът се подава на метода на ResultScreen show_result().
2. Методът show_result() използва подадения код като търси в база данни ключ който съответства на поредицата числа.
    a. Ако намери на екран се показват името на продукта и съставките с тяхните тикер символи за здравословни, несдравословни, опасни и незнаен (зелено, жълто, червеното и сиво кръгче)

b. Ако продуктът не е намерен на екран се сигнализира, че няма такъв продукт в базата данни.

3. Бутонът Back препраща потребителя към dashboard.py

```
# --- Дефиниране на списъци със съставки (Примери) ---
HEALTHY_INGREDIENTS = [
    "аскорбинова киселина",
]

UNHEALTHY_INGREDIENTS = [
    "микрокристална целулоза", "магнезиев стеарат", "колоиден безводен
силициев диоксид",
]

DANGEROUS_INGREDIENTS = [
    "талк", "е102",
]
```

Листове, които съхраняват информация за съставките и към коя група те спадат HEALTHY_INGREDIENTS, UNHEALTHY_INGREDIENTS, DANGEROUS_INGREDIENTS (здравословни, несдравословни, опасни).

```
# --- Дефиниране на цветове за категориите ---
COLOR_HEALTHY = get_color_from_hex("#4CAF50")  # Зелен (Healthy)
```

```
COLOR_UNHEALTHY = get_color_from_hex("#FFC107")  # Жълт (Unhealthy)
COLOR_DANGEROUS = get_color_from_hex("#F44336")  # Червен (Dangerous)
COLOR_UNKNOWN = get_color_from_hex("#9E9E9E")  # Сив (Unknown)
```

Този код оказва цветовете на кръгчетата на съответните съставки.

### ColoredCircle

```
class ColoredCircle(Widget):
    def __init__(self, color_rgba, **kwargs):
        super().__init__(**kwargs)
        self.color_rgba = color_rgba
        self.size_hint = (None, None)
        self.size = (dp(16), dp(16))

        with self.canvas:
            self.color_instruction = Color(*self.color_rgba)
            self.circle = Ellipse(pos=self.pos, size=self.size)

        self.bind(pos=self.update_circle, size=self.update_circle)

    def update_circle(self, *args):
        self.circle.pos = self.pos
        self.circle.size = self.size
```

Класът ColoredCircle() рисува кръгчетата до съответните съставки той има свойство за цвят color_rgba, което задава цвета спрямо класификацията на съставката кръгчетата за в размер 16x16 и имат автоматично преоразмеряване посредством свързване на своиствата pos и size с update_circle().

### ResultScreen

#### Конструктор

```
class ResultScreen(MDScreen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        # Основен layout на екрана е FloatLayout
        self.layout = FloatLayout()
        self.add_widget(self.layout)  # Добавяме го към ResultScreen
```

```
        overlay = ColorOverlay(size_hint=(1, 1), pos_hint={"center_x": 0.5,
"center_y": 0.5})
        self.layout.add_widget(overlay)
```

Създава контейнер layout в който да се поставят уиджетите по прецизно. Използва се клас ColorOverlay(), който се използва за задаване на фон на контейнера.

```
self.name_label = Label(
    text="Име на продукта:",
    halign="center",
    color=(1, 1, 1, 1),
    font_size="25",
    size_hint=(0.9, None),
    height=dp(50),
    pos_hint={"center_x": 0.5, "top": 0.95},
)
self.layout.add_widget(self.name_label)
```

Лейбълът задава начина на показва името на продукта, което е извлечено от базата данни има фиксирана височина с height=dp(50) и е центрирано хоризонтално намира се в горния ъгъл на приложението чрез pos_hint.

```
self.ingredients_card = MDCard(
    md_bg_color=(0.96, 0.96, 0.96, 0.93),
    radius=[dp(15)],
    elevation=0,
    orientation="vertical",  # MDCard е BoxLayout, така че това е валидно
    padding=dp(10),
    spacing=dp(5),
    size_hint=(0.9, 0.6),  # Заема 90% ширина, 60% височина
    pos_hint={"center_x": 0.5, "center_y": 0.47},
    # Позициониран леко под центъра, за да остави място за бутона отдолу
)
self.layout.add_widget(self.ingredients_card)
```

За да се изобразят съставките се създава контейнер за тях, който е с вертикално подравняване crientation=”vertical” и се позиционира по средата на приложението чрез pos_hint.

```
self.scroll_view = MDScrollView()
self.ingredients_card.add_widget(self.scroll_view)
```

За да се видят всички съставки без да се променя размера на контейнера най-удачно е да контейнера да има скролбар затова се

инициализира такъв self.scrol_view и се добавя към вече създадения контейнер за съставките self.ingredients_card този елемент ще позволи на потребителя да скролва нагоре и надолу и ще може да види всички съставки.

```
self.ingredients_list_layout = MDBoxLayout(
    orientation="vertical",
    size_hint_y=None,
    height=dp(1),
    spacing=dp(5),
)
self.scroll_view.add_widget(self.ingredients_list_layout)
```

Кодът инициализира уиджет кутия с вертикално подреждане важен параметър тук е size_hint_y=None това ще позволи потребителя да скролва списъка надолу и нагоре. Листа със съставките се добавя към self.scroll_view не към self.ingredients_card.

```
self.card_button = MDCard(
    size_hint=(None, None),
    size=(323, 50),
    pos_hint={"center_x": 0.5, "center_y": 0.1},
    radius=[12],
    elevation=0,
    md_bg_color=(1, 1, 1, 0.73),
)

self.back_button = MDFlatButton(
    text="Back",
    theme_text_color="Custom",
    text_color=(0.058, 0.227, 0.212, 1),
    font_size="15",
    font_name="PoppinsSemiBold",
    size_hint=(1, 1),
    height="48dp",
    on_release=self.go_back,
)
self.card_button.add_widget(self.back_button)
self.layout.add_widget(self.card_button)  # Добавяме към новия FloatLayout
```

…

```
def go_back(self, *args):
    self.manager.current = "dashboard"
```

Бутонът който при натискане връща потребителя към табло за управление посредством метода go_back(). Използва персонализиран стил на изобразяване с помощта на self.card_button

и активирането на параметъра за използване на персонализиран стил theme_text_color="Custom".

Функция show_result

```
def show_result(self, qrcode):
    conn = sqlite3.connect("qr_data.db")
    cursor = conn.cursor()
    cursor.execute("SELECT name, ingredients FROM qr_items WHERE qrcode = ?",
(qrcode,))
    result = cursor.fetchone()
    conn.close()
```

Методът, който се извиква от QRReadScreen() и му се подава QR кодът. Първо методът прави връзка с базата данни и подава заявка за извличане името на продукта и съставките му базирани на qrcode чрез cursor.execute() след което извлича резултата и затваря връзката с conn.close().

```
self.ingredients_list_layout.clear_widgets()
self.ingredients_list_layout.height = 0
```

Код, който изчиства предишните резултати и рестартира височината ако е имало преди това по-дълъг списък.

```
if result:
    product_name = result[0]
    ingredients_raw = result[1]

    self.name_label.text = f"{product_name}"

    # Парсваме съставките, разделени с ", "
    ingredients_list = [
        s.strip().lower() for s in ingredients_raw.split(",") if s.strip()
    ]
```

Първата стъпка при намерен продукт извличат се данните в променливи. Лейбълът който служи за показване на името на продукта се актуализира self.name_label.text съставките се извличат една по една от списъка ingredients_raw като се предполага че всички съставки са разделени с ", " всички ненужни празни пространства се премахват за по-бързо сравняване по нататък на съставките кодът който служи за цялата тази логика е ingredients_list.

```
# >>> НАСТРОЙКИ ЗА ФИНО ПОЗИЦИОНИРАНЕ НА ТЕКСТА (в dp) <<<
# Променяйте тези стойности, за да измествате текста спрямо кръгчето.
#
# Положителна стойност за text_offset_x_dp измества текста НАДЯСНО.
# Отрицателна стойност за text_offset_x_dp измества текста НАЛЯВО.
#
# Положителна стойност за text_offset_y_dp измества текста НАДОЛУ.
# Отрицателна стойност за text_offset_y_dp измества текста НАГОРЕ.
#
# Пример: dp(0) за без отместване. dp(1) за отместване с 1 пиксел.
# Експериментирайте с малки стойности (напр. от -5 до 5)
text_offset_x_dp = dp(-12)  # Хоризонтално изместване на текста
(наляво/надясно)
text_offset_y_dp = dp(12)  # Вертикално изместване на текста (нагоре/надолу)
# >>> КРАЙ НА НАСТРОЙКИТЕ <<<
```

Тези две променливи служат за прецизно подравняване на съставките спрямо кръгчето така че да се разбере ясно кое кръгче за коя съставка се отнася. Тези променливи се използват по-нататък в кода, когато се създава визуализацията на съставките.

```
for ingredient in ingredients_list:
    category_color = COLOR_UNKNOWN  # По подразбиране

    if ingredient in HEALTHY_INGREDIENTS:
        category_color = COLOR_HEALTHY
    elif ingredient in UNHEALTHY_INGREDIENTS:
        category_color = COLOR_UNHEALTHY
    elif ingredient in DANGEROUS_INGREDIENTS:
        category_color = COLOR_DANGEROUS
```

Цикъл който проверява всяка една съставка от списъка към коя категория спада и създава променлива category_color, която ще се използва по-нататък за определяне цвета на кръгчето при неговото създаване.

```
# Създаваме ред за всяка съставка
ingredient_row = MDBoxLayout(
    orientation="horizontal",
    size_hint_y=None,
    height=dp(30),  # Фиксирана височина за всеки ред
)
```

За всяка съставка се създава ред (контейнер), в който да поставим съответния тикер на съставката и нейното име.

```
# Цветно кръгче
circle_widget = ColoredCircle(color_rgba=category_color)

# Обвиваме кръгчето в MDBoxLayout, за да го центрираме вертикално
circle_container = MDBoxLayout(
    orientation="vertical",
    size_hint=(None, 1),  # Заема пълната височина на реда
    width=dp(30),  # Фиксирана ширина за контейнера на кръгчето
)
circle_container.add_widget(circle_widget)
```

Създава се кръгче с класа ColoredCircle() със съответния цвят, който индикира класа на съставката. За кръгчето се създава контейнер който отговаря на изискванията на реда за големина width кръгчето заема пълната височина на реда size_hint=(None, 1).

```
# Label за текста на съставката
ingredient_label = MDLabel(
    text=ingredient.capitalize(),
    theme_text_color="Primary",
    font_style="Body2",
    halign="left",  # Хоризонтално подравняване на текста в неговото
пространство
    valign="middle",  # Вертикално подравняване на текста в неговото
пространство
    size_hint=(1, 1),  # Заема цялото налично пространство
    # Прилагаме padding директно към MDLabel за фино отместване
    # padding: [ляво, горе, дясно, долу]
    padding=[
        text_offset_x_dp if text_offset_x_dp > 0 else 0,  # Ляв padding (движи
текста надясно)
        text_offset_y_dp,  # Горен padding (движи текста надолу)
        -text_offset_x_dp if text_offset_x_dp < 0 else 0,  # Десен padding
(движи текста наляво)
        0  # Долен padding
    ]
)
```

Променливата ingredient_laber отговаря за стилът на съставката (theme_text_color, font_style) нейното подравняване (halign, valing), размер и фината настройка с padding и променливите text_offset_x_dp и text_offset_y_dp.

```
# Ред на добавяне: кръгче отляво, текст отдясно
ingredient_row.add_widget(circle_container)  # Първо кръгчето (отляво)
ingredient_row.add_widget(ingredient_label)  # След това лейбъла
```

Контейнера на кръгчето и съставката се добавят в реда ingredient_row.

```
self.ingredients_list_layout.add_widget(ingredient_row)
```

Редът на с тикера и името на съставката се добавя в контейнера.

```
self.ingredients_list_layout.height += dp(30) +
self.ingredients_list_layout.spacing
```

Автоматично увеличаване височината на контейнера със съставки, за да се разбере колко съдържание има за скролване

```
else:
    self.name_label.text = "Продуктът не е намерен."
    self.ingredients_list_layout.clear_widgets()
    self.ingredients_list_layout.height = 0
```

Ако резултат не е намерен то само се актуализира лейбълът за името на продукта на “Продуктът не е намерен.” и се изчиства контейнерът за съставки, ако евентуално преди това потребителят е сканирал продукт и той е бил запълнен.

## Заключение

Приложението е интуитивно, което позволява на потребителя от пръв поглед да разбере идеята му. На приложението липсват някои функционалности като създаване на акаунт и логване с вече съществуващ акаунт. Преди предприемане на каквито и да е действия потребителя трябва да се консултира с личен лекар или фармацевт.

## Използвани технологии

Python
KivyMD
SQL/sqlite