

2023-2024 1ste semester

Mobiele Apps 2: verslag project

Wijnhuis Ronny

Calvin Liaci
3ICT

Inhoudsopgave

Verslag project: Wijnhuis Ronny	6
1. Korte toelichting app vanuit het oogpunt van de gebruiker.....	6
2. Icon en Splash Screen	7
2.1 Visuele weergave en code	7
2.1.1 Icon	7
2.1.2 Splash screen	7
2.2 Uitleg.....	7
3. Navigatie	8
3.1 Visuele weergave en code	8
3.2 Uitleg.....	8
4. Startpagina.....	9
4.1 Visuele weergave	9
4.2 HomePagePhotosRepository	10
4.2.1 Code.....	10
4.2.2 Uitleg code.....	10
4.3 HomepageFragment	11
4.3.1 Code.....	11
4.3.2 Uitleg code.....	11
4.4 HomepageFragmentViewModel.....	12
4.4.1 Code.....	12
4.4.2 Uitleg code.....	12
4.5 Fragment_homepage.xml.....	13
4.5.1 Code.....	13
4.5.2 Uitleg code.....	14
5. Wijnen pagina.....	15
5.1 Wijncategorieën pagina	15
5.1.1 Visuele weergave	15
5.1.2 WinesCategoryPhotosRepository	16
5.1.2.1 Code.....	16
5.1.2.2 Uitleg code.....	16
5.1.3 WinesCategoryFragment	17
5.1.3.1 Code.....	17
5.1.3.2 Uitleg code.....	18
5.1.4 WinesCategoryViewModel	18
5.1.4.1 Code.....	18
5.1.4.2 Uitleg code.....	18
5.1.5 Fragment_wines_category.xml.....	19
5.1.5.1 Code.....	19

5.1.5.2	Uitleg code.....	22
5.2	Wijnen pagina	23
5.2.1	Visuele weergave	23
5.2.2	Wine Model	24
5.2.2.1	Code.....	24
5.2.2.2	Uitleg code.....	24
5.2.3	WinesRepository	25
5.2.3.1	Code.....	25
5.2.3.2	Uitleg code.....	25
5.2.4	WineAdapter.....	26
5.2.4.1	Code.....	26
5.2.4.2	Uitleg code.....	26
5.2.5	WinesFragment.....	27
5.2.5.1	Code.....	27
5.2.5.2	Uitleg code.....	28
5.2.6	WinesFragmentViewModel	28
5.2.6.1	Code.....	28
5.2.6.2	Uitleg code.....	29
5.2.7	Fragment_wines.xml	29
5.2.7.1	Code.....	29
5.2.7.2	Uitleg code.....	29
5.2.8	Wines_item.xml	30
5.2.8.1	Code.....	30
5.2.8.2	Uitleg code.....	31
6.	Degustatie pagina	32
6.1	Visuele weergave	32
6.2	WineTasting Model	32
6.2.1	Code.....	32
6.2.2	Uitleg code.....	32
6.3	WineTastingsRepository	33
6.3.1	Code.....	33
6.3.2	Uitleg code.....	33
6.4	WineTastingsAdapter	34
6.4.1	Code.....	34
6.4.2	Uitleg code.....	34
6.5	WineTastingsFragment	35
6.5.1	Code.....	35
6.5.2	Uitleg code.....	36
6.6	WineTastingsViewModel	36

6.6.1	Code	36
6.6.2	Uitleg code	37
6.7	Fragment_wine_tastings.xml	37
6.7.1	Code	37
6.7.2	Uitleg code	37
6.8	Wine_tasting_item.xml	38
6.8.1	Code	38
6.8.2	Uitleg code	39
6.9	E-mailintent	40
6.9.1	Visuele weergave	40
7.	Winkelwagen	41
7.1	Winkelwagen pagina	41
7.1.1	Visuele weergave	41
7.1.2	ShoppingCartAdapter	42
7.1.2.1	Code	42
7.1.2.2	Uitleg code	42
7.1.3	ShoppingCartFragment	43
7.1.3.1	Code	43
7.1.3.2	Uitleg code	44
7.1.4	ShoppingCartViewModel	44
7.1.4.1	Code	44
7.1.4.2	Uitleg code	45
7.1.5	Fragment_shopping_cart.xml	46
7.1.5.1	Code	46
7.1.5.2	Uitleg code	47
7.1.6	Shoppingcart_row.xml	48
7.1.6.1	Code	48
7.1.6.2	Uitleg code	49
7.2	Checkout pagina	50
7.2.1	Visuele weergave	50
7.2.2	CheckoutOrderFragment	51
7.2.2.1	Code	51
7.2.2.2	Uitleg code	52
7.2.3	CheckoutOrderViewModel	52
7.2.3.1	Code	52
7.2.3.2	Uitleg code	53
7.2.4	Fragment_checkout_order.xml	53
7.2.4.1	Code	53
7.2.4.2	Uitleg code	55

7.3	E-mailintent	55
7.3.1	Visuele weergave	55
8.	Database en Storage	56
8.1	Indeling Firebase Realtime Database.....	56
8.2	Indeling Firebase Storage.....	57

Verslag project: Wijnhuis Ronny

1. Korte toelichting app vanuit het oogpunt van de gebruiker

Welkom bij de Wijnhuis Ronny app, waar passie, wijn en kwaliteit samenkomen! Op onze startpagina wordt je begroet met onze krachtige slogan, die de essentie van ons wijnhuis weergeeft. Daaronder vind je waardevolle informatie over wie we zijn en wat onze passie voor wijn inhoudt.

Wanneer je de ontdekkingsreis door ons wijnassortiment begint, kom je terecht op de wijnenpagina. Hier word je verwelkomd door een uitgebreide selectie wijncategorieën, variërend van verfijnde witte wijnen tot rijke rode en verfrissende rosé wijnen. Na het maken van je keuze navigeer je naar de specifieke pagina van de geselecteerde wijnsoort en ontdek je een lijst met heerlijke wijnen die je direct aan je winkelwagen kunt toevoegen.

Bij de winkelwagen aangekomen, krijg je een overzicht van al de wijnen die je hebt geselecteerd, inclusief de hoeveelheden en prijzen. Hier heb je de mogelijkheid om je bestelling te plaatsen met een eenvoudige druk op de "Bestelling plaatsen"-knop. Dit opent een nieuwe pagina waar je persoonlijke gegevens kunt invoeren om de checkout te voltooien.

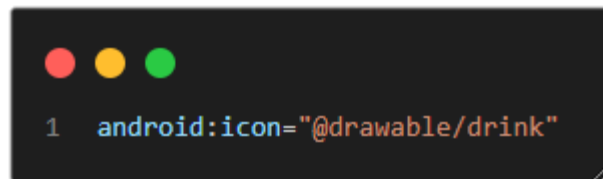
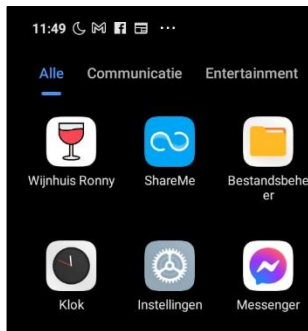
Eenmaal je gegevens zijn ingevoerd, klik je op "Bestelling afronden" en word je naadloos doorgestuurd naar je mail-app. Hier staat een automatisch gegenereerde e-mail klaar met al je gegevens en de details van je bestelde wijnen. Door op "Verzenden" te drukken, bevestig je officieel je wijnbestelling.

Naast het winkelen bieden we ook de mogelijkheid om je in te schrijven voor boeiende degustaties. Navigeer eenvoudig naar de degustatiepagina in de navigatiebalk, ontdek de komende evenementen en schrijf je in met slechts één klik. Hier ontvang je ook een automatisch opgestelde e-mail om je inschrijving te bevestigen en jezelf voor te bereiden op een smaakvolle ervaring.

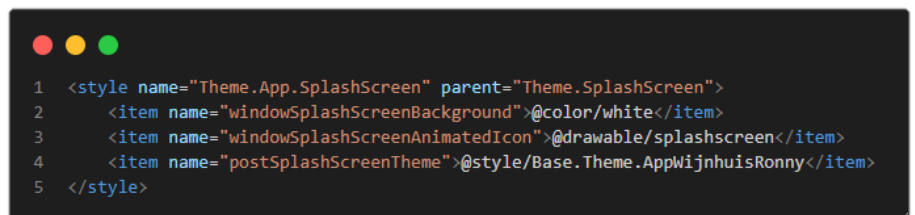
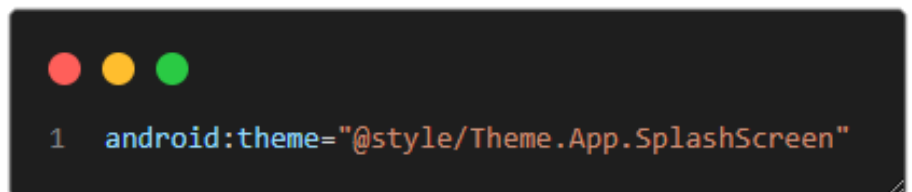
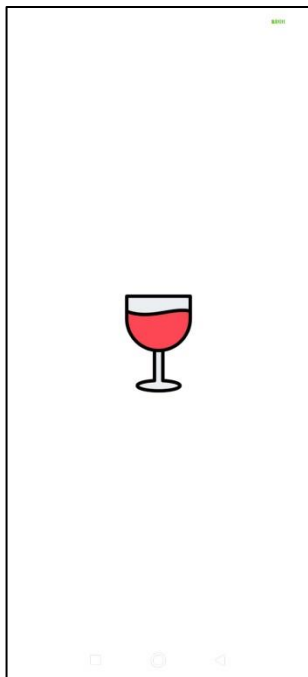
2. Icon en Splash Screen

2.1 Visuele weergave en code

2.1.1 Icon



2.1.2 Splash screen



2.2 Uitleg

Om de Icon en Splash screen aan te passen in Android Studio, moest ik een paar wijzigingen aanbrengen in het `AndroidManifest.xml`-bestand. Voor het pictogram moest ik zelf een afbeelding toevoegen via `android:icon`. Wat betreft het opstartscherm, moest ik eerst een stijl maken in het `styles.xml`-bestand, genaamd `Theme.App.SplashScreen`. Deze stijl had een witte achtergrondkleur, een afbeelding met de naam "splashscreen" en een thema voor gebruik na de Splash screen. Daarna moest ik ook nog in het `AndroidManifest`-bestand het thema instellen als waarde binnen `android:theme`.

3. Navigatie

3.1 Visuele weergave en code



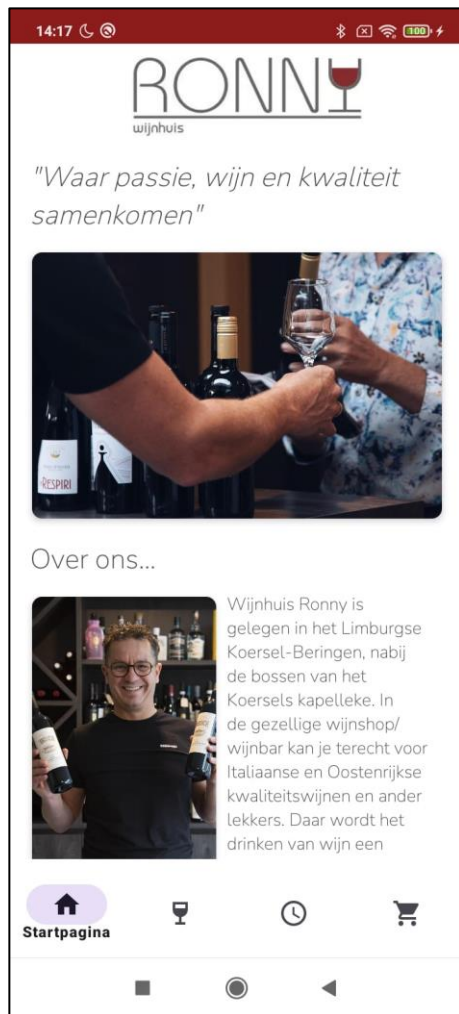
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item
4         android:id="@+id/homepageFragment"
5         android:icon="@drawable/baseline_home_24"
6         android:title="Startpagina"/>
7
8     <item
9         android:id="@+id/winesCategoryFragment"
10        android:icon="@drawable/baseline_wine_bar_24"
11        android:title="Wijnen"/>
12
13    <item
14        android:id="@+id/wineTastingsFragment"
15        android:icon="@drawable/baseline_access_time_24"
16        android:title="Degustaties"/>
17
18    <item
19        android:id="@+id/shoppingCartFragment"
20        android:icon="@drawable/baseline_shopping_cart_24"
21        android:title="Winkelmand"/>
22 </menu>
```

3.2 Uitleg

Deze `bottom_nav.xml` definieert het menu voor een Bottom Navigation View in een Android-applicatie. Het menu bevat vier items, elk geassocieerd met een specifiek fragment in de app. Deze XML-configuratie wordt gebruikt in combinatie met een Bottom Navigation View in de `activity_main.xml`, waardoor een navigatiebalk ontstaat met de genoemde items. Elk item vertegenwoordigt een specifiek deel van de app en wordt geassocieerd met een overeenkomstig fragment, waardoor gebruikers gemakkelijk kunnen schakelen tussen verschillende delen van de app. Het gebruik van pictogrammen en duidelijke titels draagt bij aan een simpele gebruikerservaring.

4. Startpagina

4.1 Visuele weergave



4.2 HomePagePhotosRepository

4.2.1 Code

```
1 class HomePagePhotosRepository private constructor() {
2
3     private val databaseReference: DatabaseReference = FirebaseDatabase.getInstance().getReference("Startpagina")
4
5     init {
6         Log.d("Repository", "Database reference initialized: $databaseReference")
7     }
8
9     companion object {
10         @Volatile
11         private var INSTANCE: HomePagePhotosRepository? = null
12
13         fun getInstance(): HomePagePhotosRepository {
14             return INSTANCE ?: synchronized(this) {
15                 INSTANCE = HomePagePhotosRepository()
16                 INSTANCE!!
17             }
18         }
19     }
20
21     // Modify the callback to explicitly specify the type
22     fun getPhotoUrls(callback: (List<String>) -> Unit) {
23         databaseReference.addListenerForSingleValueEvent(object : ValueEventListener {
24             override fun onDataChange(snapshot: DataSnapshot) {
25                 val photoUrls = mutableListOf<String>()
26                 for (childSnapshot in snapshot.children) {
27                     // Adjust the path based on your data structure
28                     val photoUrl = childSnapshot.child("Url").value as? String
29                     photoUrl?.let {
30                         photoUrls.add(it)
31                     }
32                 }
33                 callback(photoUrls)
34             }
35
36             override fun onCancelled(error: DatabaseError) {
37                 Log.e("HomePagePhotosRepository", "Failed to retrieve photo URLs", error.toException())
38                 callback(emptyList())
39             }
40         })
41     }
42 }
```

4.2.2 Uitleg code

Bovenstaande Kotlin-code vertegenwoordigt een klasse genaamd `HomePagePhotosRepository`, bedoeld voor het ophalen van foto-URL's uit een Firebase Realtime Database.

De klasse begint met het initialiseren van een `databaseReference` die verwijst naar het gedeelte "Startpagina" in de Firebase-database. Dit wordt gedaan in de constructor, waarbij ook een logboekmelding wordt gegenereerd om te melden dat de database-referentie is geïnitieerd.

Vervolgens wordt een companion object gebruikt om een Singleton-patroon te implementeren. Dit zorgt ervoor dat er slechts één exemplaar van `HomePagePhotosRepository` wordt gemaakt en gebruikt.

De `getInstance`-methode in het companion object zorgt voor het ophalen van deze enkele instantie. Als er nog geen instantie bestaat, wordt er één gemaakt.

De hoofdfunctionaliteit van de klasse is te vinden in de `getPhotoUrls`-methode. Deze methode accepteert een callback-functie die een lijst met foto-URL's als parameter heeft. Binnen de methode wordt een listener toegevoegd aan de database-referentie om eenmalig gegevens op te halen. Bij succes worden de foto-URL's uit de database gehaald en doorgegeven aan de callback. Als er zich een fout voordoet, wordt een foutmelding gegenereerd en wordt een lege lijst naar de callback gestuurd.

In essentie biedt deze klasse een gestructureerde manier om foto-URL's op te halen vanuit de database, waarbij het Singleton-patroon ervoor zorgt dat er geen onnodige instanties worden gemaakt.

4.3 HomepageFragment

4.3.1 Code

```
1 class HomepageFragment : Fragment() {
2
3     private var _binding: FragmentHomepageBinding? = null
4     private val binding get() = _binding!!
5
6     private lateinit var viewModel: HomepageFragmentViewModel
7
8     override fun onCreateView(
9         inflater: LayoutInflater, container: ViewGroup?,
10        savedInstanceState: Bundle?
11    ): View? {
12        _binding = FragmentHomepageBinding.inflate(inflater, container, false)
13        val view = binding.root
14
15        viewModel = ViewModelProvider(this).get(HompageFragmentViewModel::class.java)
16
17        binding.homepageFragmentViewModel = viewModel
18        binding.lifecycleOwner = viewLifecycleOwner
19
20        // Observe changes in photo URLs and update ImageViews
21        viewModel.photoUrlsLiveData.observe(viewLifecycleOwner, Observer { photoUrls ->
22            if (photoUrls.isNotEmpty()) {
23                Picasso.get().load(photoUrls[0]).into(binding.imageView2)
24                Picasso.get().load(photoUrls[1]).into(binding.overOnsImageView)
25            }
26        })
27
28        // Load photo URLs
29        viewModel.loadPhotoUrls()
30
31        return view
32    }
33
34    override fun onDestroyView() {
35        super.onDestroyView()
36        _binding = null
37    }
38 }
```

4.3.2 Uitleg code

De `HomepageFragment`-klasse is een subclass van `Fragment`. Binnen deze klasse wordt de binding met de bijbehorende lay-out (`FragmentHomepageBinding`) geïmplementeerd via het Android View Binding-systeem.

Het fragment heeft een `ViewModel` genaamd `HomepageFragmentViewModel`, die verantwoordelijk is voor het beheren van gegevens en logica die specifiek zijn voor dit fragment.

In de `onCreateView`-methode, die wordt aangeroepen bij het maken van de fragment, wordt de binding geïnitieerd, de ViewModel geassocieerd en wordt de observer ingesteld om wijzigingen in de lijst met foto-URL's te volgen. Wanneer er veranderingen optreden, worden de ImageViews bijgewerkt met behulp van de Picasso-bibliotheek om afbeeldingen van de ontvangen URL's weer te geven. Vervolgens wordt de methode `loadPhotoUrls` van de ViewModel aangeroepen om de foto-URL's te laden.

De `onDestroyView`-methode wordt gebruikt om het `_binding`-object naar `null` te zetten, waardoor geheugenlekken worden voorkomen.

Kort samengevat biedt dit fragment een gestructureerde manier om de lay-out te initialiseren, de ViewModel te associëren en dynamisch afbeeldingen bij te werken op basis van wijzigingen in de lijst met foto-URL's. De `ViewModel`-laag is verantwoordelijk voor het beheren van de gegevenslogica, terwijl het fragment zich richt op het verzorgen van de gebruikersinterface-interactie.

4.4 HomepageFragmentViewModel

4.4.1 Code

A screenshot of a code editor with a dark background and light-colored text. The code is written in Kotlin and defines a class named `HomepageFragmentViewModel` that inherits from `ViewModel`. Inside the class, there is a private val `whiteWinesRepository` assigned to `HomePagePhotosRepository.getInstance()`. There is also a val `photoUrlsLiveData` of type `MutableLiveData<List<String>>` assigned to `MutableLiveData()`. A function `loadPhotoUrls()` is defined, which calls `whiteWinesRepository.getPhotoUrls` with a lambda callback that updates `photoUrlsLiveData` using `postValue`. The code is numbered from 1 to 12.

```
1 class HomepageFragmentViewModel : ViewModel() {  
2  
3     private val whiteWinesRepository = HomePagePhotosRepository.getInstance()  
4  
5     val photoUrlsLiveData: MutableLiveData<List<String>> = MutableLiveData()  
6  
7     fun loadPhotoUrls() {  
8         whiteWinesRepository.getPhotoUrls { photoUrls ->  
9             photoUrlsLiveData.postValue(photoUrls)  
10        }  
11    }  
12 }
```

4.4.2 Uitleg code

De `HomepageFragmentViewModel`-klasse is een subclass van `ViewModel`. Binnen deze klasse wordt er een instantie van `HomePagePhotosRepository` aangemaakt via de methode `getInstance()`.

De klasse bevat een `MutableLiveData` met de naam `photoUrlsLiveData`, die fungeert als een waarnemerbare gegevensbron voor de lijst met foto-URL's

De `loadPhotoUrls`-methode in deze klasse wordt aangeroepen om foto-URL's op te halen vanuit de `HomePagePhotosRepository`. Hierbij wordt een callback gebruikt om de verkregen foto-URL's door te geven aan `photoUrlsLiveData`. Het gebruik van `postValue` zorgt ervoor dat deze bewerking veilig wordt uitgevoerd op de hoofdthread.

In wezen fungeert deze ViewModel als een tussenschakel tussen de `HomePagePhotosRepository` en het bijbehorende fragment (`HomepageFragment`). Het initieert het ophalen van foto-URL's en zorgt ervoor dat wijzigingen in deze gegevens worden gecommuniceerd naar de observerende componenten, zoals de ImageViews in het bijbehorende fragment.

4.5 Fragment_homepage.xml

4.5.1 Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:orientation="vertical"
6     tools:context=".HomepageFragment">
7
8     <data>
9         <variable
10             name="homepageFragmentViewModel"
11             type="com.example.appwijnhuisronny.HomepageFragmentViewModel" />
12     </data>
13
14     <ScrollView
15         android:layout_width="match_parent"
16         android:layout_height="match_parent">
17
18         <androidx.constraintlayout.widget.ConstraintLayout
19             android:layout_width="match_parent"
20             android:layout_height="wrap_content">
21
22             <TextView
23                 android:id="@+id/sloganTextView"
24                 android:layout_width="wrap_content"
25                 android:layout_height="wrap_content"
26                 android:layout_marginStart="8dp"
27                 android:layout_marginTop="8dp"
28                 android:fontFamily="@font/nunito_sans_extralight"
29                 android:text="@string/wijnhuisRonnySlogan"
30                 android:textSize="24sp"
31                 android:textStyle="italic"
32                 app:layout_constraintStart_toStartOf="parent"
33                 app:layout_constraintTop_toTopOf="parent" />
34
35             <androidx.cardview.widget.CardView
36                 android:id="@+id/photoCardView"
37                 android:layout_width="0dp"
38                 android:layout_height="228dp"
39                 android:layout_marginStart="3dp"
40                 android:layout_marginTop="8dp"
41                 android:layout_marginEnd="3dp"
42                 app:cardCornerRadius="8dp"
43                 app:cardElevation="4dp"
44                 app:cardUseCompatPadding="true"
45                 app:layout_constraintEnd_toEndOf="parent"
46                 app:layout_constraintStart_toStartOf="parent"
47                 app:layout_constraintTop_toBottomOf="@+id/sloganTextView">
48
49                 <ImageView
50                     android:id="@+id/imageView2"
51                     android:layout_width="wrap_content"
52                     android:layout_height="wrap_content"
53                     android:scaleType="centerCrop"
54                     android:src="@drawable/homepagephoto" />
55
56             </androidx.cardview.widget.CardView>
57
58             <TextView
59                 android:id="@+id/overOnsTextView"
60                 android:layout_width="wrap_content"
61                 android:layout_height="wrap_content"
62                 android:layout_marginTop="8dp"
63                 android:fontFamily="@font/nunito_sans_extralight"
64                 android:text="@string/overOns"
65                 android:textSize="20sp"
66                 app:layout_constraintStart_toStartOf="@+id/sloganTextView"
67                 app:layout_constraintTop_toBottomOf="@+id/photoCardView" />
68
69             <androidx.cardview.widget.CardView
70                 android:id="@+id/overOnsCardView"
71                 android:layout_width="159dp"
72                 android:layout_height="300dp"
73                 android:layout_marginStart="3dp"
74                 android:layout_marginTop="8dp"
75                 app:cardCornerRadius="8dp"
76                 app:cardElevation="4dp"
77                 app:cardUseCompatPadding="true"
78                 app:layout_constraintStart_toStartOf="parent"
79                 app:layout_constraintTop_toBottomOf="@+id/overOnsTextView">
80
```

```

82         <ImageView
83             android:id="@+id/overOnsImageView"
84             android:layout_width="match_parent"
85             android:layout_height="match_parent"
86             android:scaleType="centerCrop"
87             android:src="@drawable/homepagephoto" />
88
89     </androidx.cardview.widget.CardView>
90
91     <TextView
92         android:id="@+id/overOnsTextView2"
93         android:layout_width="171dp"
94         android:layout_height="296dp"
95         android:layout_marginStart="2dp"
96         android:layout_marginTop="4dp"
97         android:fontFamily="@font/nunito_sans_extralight"
98         android:text="@string/overOnsText"
99         android:textSize="15sp"
100         app:layout_constraintStart_toEndOf="@+id/overOnsCardView"
101         app:layout_constraintTop_toTopOf="@+id/overOnsCardView" />
102
103     <TextView
104         android:id="@+id/overOnsTextView3"
105         android:layout_width="323dp"
106         android:layout_height="31dp"
107         android:layout_marginTop="8dp"
108         android:fontFamily="@font/nunito_sans_extralight"
109         android:text="@string/overRonny"
110         android:textSize="20sp"
111         app:layout_constraintStart_toStartOf="@+id/overOnsTextView"
112         app:layout_constraintTop_toBottomOf="@+id/overOnsTextView2" />
113
114     <TextView
115         android:id="@+id/overOnsTextView4"
116         android:layout_width="332dp"
117         android:layout_height="124dp"
118         android:fontFamily="@font/nunito_sans_extralight"
119         android:text="@string/overRonnyText"
120         android:textSize="15sp"
121         app:layout_constraintStart_toStartOf="@+id/overOnsTextView3"
122         app:layout_constraintTop_toBottomOf="@+id/overOnsTextView3" />
123 </androidx.constraintlayout.widget.ConstraintLayout>
124 </ScrollView>
125 </layout>

```

4.5.2 Uitleg code

Dit XML-bestand definieert de lay-out voor het fragment met de naam `HomepageFragment`. De lay-out begint met de specificatie van de variabele `homepageFragmentViewModel` in het ``-element, waardoor gegevensuitwisseling tussen de lay-out en de ViewModel mogelijk is.

Binnen de `ScrollView` bevindt zich een `ConstraintLayout`, dat de hoofdindeling van de UI-elementen beheert. Er zijn meerdere `TextView`-elementen voor het weergeven van tekst, waaronder de slogan, een over-ons-kop, en alinea's tekst. De teksteigenschappen zoals lettergrootte en lettertype zijn aangepast voor elk tekstelement.

Daarnaast zijn er twee `CardView`-elementen, `photoCardView` en `overOnsCardView`, die fungeren als containers voor de `ImageView`s. Deze cards hebben afgeronde hoeken en een subtiele schaduw voor visuele aantrekkelijkheid.

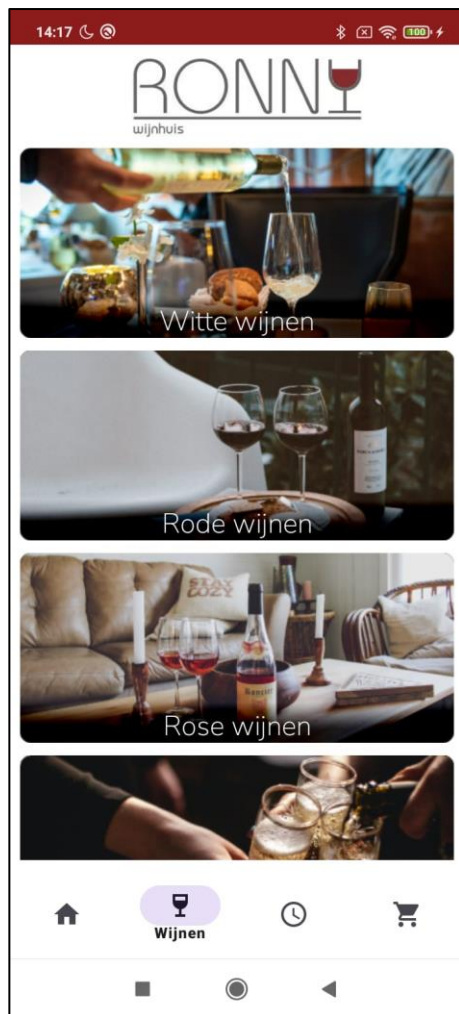
Binnen de `CardView`-elementen bevinden zich `ImageView`-elementen (`imageView2` en `overOnsImageView`) voor het weergeven van afbeeldingen. Momenteel is de bron van de afbeeldingen ingesteld op een standaardafbeelding genaamd "homepagephoto", maar deze zullen later dynamisch worden vervangen door afbeeldingen die worden opgehaald vanuit de ViewModel.

Verder zijn er `TextView`-elementen specifiek voor het gedeelte "Over Ons", waaronder een subkop, tekstalinea's en informatie over Ronny.

5. Wijnen pagina

5.1 Wijncategorieën pagina

5.1.1 Visuele weergave



5.1.2 WinesCategoryPhotosRepository

5.1.2.1 Code

```
1 class WinesCategoryPhotosRepository private constructor() {
2
3     private val databaseReference: DatabaseReference = FirebaseDatabase.getInstance().getReference("Categorieën")
4
5     init {
6         Log.d("Repository", "Database reference initialized: $databaseReference")
7     }
8
9     companion object {
10         @Volatile
11         private var INSTANCE: WinesCategoryPhotosRepository? = null
12
13         fun getInstance(): WinesCategoryPhotosRepository {
14             return INSTANCE ?: synchronized(this) {
15                 INSTANCE = WinesCategoryPhotosRepository()
16                 INSTANCE!!
17             }
18         }
19     }
20
21     fun getPhotoUrls(callback: (List<String>) -> Unit) {
22         databaseReference.addListenerForSingleValueEvent(object : ValueEventListener {
23             override fun onDataChange(snapshot: DataSnapshot) {
24                 val photoUrls = mutableListOf<String>()
25                 for (childSnapshot in snapshot.children) {
26                     // Adjust the path based on your data structure
27                     val photoUrl = childSnapshot.child("Url").value as? String
28                     photoUrl?.let {
29                         photoUrls.add(it)
30                     }
31                 }
32                 callback(photoUrls)
33             }
34
35             override fun onCancelled(error: DatabaseError) {
36                 Log.e("HomePagePhotosRepository", "Failed to retrieve photo URLs", error.toException())
37                 callback(emptyList())
38             }
39         })
40     }
41 }
```

5.1.2.2 Uitleg code

Deze Kotlin-code vertegenwoordigt een repositoryklasse genaamd `WinesCategoryPhotosRepository`, ontworpen om foto-URL's op te halen vanuit een Firebase Realtime Database voor het gedeelte "Categorieën".

De klasse begint met het initialiseren van een `databaseReference` naar het "Categorieën"-gedeelte in de Firebase Realtime Database. Deze verwijzing wordt gemaakt in de constructor van de klasse, waarbij ook een logboekmelding wordt gegenereerd om aan te geven dat de database-referentie is geïnitieerd.

Daarnaast maakt de klasse gebruik van het Singleton-patroon, waardoor slechts één instantie van `WinesCategoryPhotosRepository` kan bestaan. Dit wordt bereikt door middel van het `companion object` en de `getInstance`-methode.

De `getPhotoUrls`-methode wordt gebruikt om foto-URL's op te halen vanuit de database. Hiervoor wordt een `ValueEventListener` aan de database-referentie toegevoegd om eenmalig gegevens op te halen. In het geval van een succesvolle gegevenswijziging worden de foto-URL's uit de database gehaald en doorgegeven aan een callback-functie.

De callback-functie wordt gedefinieerd als een lambda-uitdrukking met het type `(List<String>) -> Unit`. Binnen deze callback worden de foto-URL's verwerkt en doorgegeven aan de externe componenten die

deze repository gebruiken. In het geval van een fout bij het ophalen van de gegevens, wordt een foutmelding gegenereerd en wordt een lege lijst naar de callback gestuurd.

Kort samengevat biedt deze klasse een gestructureerde manier om foto-URL's op te halen vanuit een Firebase-database met de mogelijkheid van Singleton-gebruik, waardoor efficiënt beheer van deze gegevens mogelijk is.

5.1.3 WinesCategoryFragment

5.1.3.1 Code

```
1 class WinesCategoryFragment : Fragment() {
2
3     private var _binding: FragmentWinesCategoryBinding? = null
4     private val binding get() = _binding!!
5
6     private lateinit var viewModel: WinesCategoryViewModel
7
8     override fun onCreateView(
9         inflater: LayoutInflater, container: ViewGroup?,
10        savedInstanceState: Bundle?
11    ): View? {
12        _binding = FragmentWinesCategoryBinding.inflate(inflater, container, false)
13        val view = binding.root
14
15        viewModel = ViewModelProvider(this).get(WinesCategoryViewModel::class.java)
16
17        binding.winesCategoryFragmentViewModel = viewModel
18
19        binding.layoutWitteWijn.setOnClickListener {
20            val bundle = Bundle()
21            bundle.putString("category", "WitteWijnen")
22            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
23        }
24
25        binding.layoutRodeWijn.setOnClickListener {
26            val bundle = Bundle()
27            bundle.putString("category", "RodeWijnen")
28            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
29        }
30
31        binding.layoutRoseWijn.setOnClickListener {
32            val bundle = Bundle()
33            bundle.putString("category", "RoseWijnen")
34            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
35        }
36
37        binding.layoutBubbelsWijn.setOnClickListener {
38            val bundle = Bundle()
39            bundle.putString("category", "Bubbels")
40            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
41        }
42
43        binding.layoutPorto.setOnClickListener {
44            val bundle = Bundle()
45            bundle.putString("category", "Porto")
46            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
47        }
48
49        binding.layoutGins.setOnClickListener {
50            val bundle = Bundle()
51            bundle.putString("category", "Gins")
52            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
53        }
54
55        binding.layoutOverige.setOnClickListener {
56            val bundle = Bundle()
57            bundle.putString("category", "Overige")
58            view.findNavController().navigate(R.id.action_winesCategoryFragment_to_winesFragment, bundle)
59        }
60
61        // Observe changes in photo URLs and update ImageViews
62        viewModel.photoUrlsLiveData.observe(viewLifecycleOwner, Observer { photoUrls ->
63            if (photoUrls.isNotEmpty()) {
64                Picasso.get().load(photoUrls[0]).into(binding.imageWitteWijn)
65                Picasso.get().load(photoUrls[1]).into(binding.imageRodeWijn)
66                Picasso.get().load(photoUrls[2]).into(binding.imageRoseWijn)
67                Picasso.get().load(photoUrls[3]).into(binding.imageBubbels)
68                Picasso.get().load(photoUrls[4]).into(binding.imagePorto)
69                Picasso.get().load(photoUrls[5]).into(binding.imageGins)
70                Picasso.get().load(photoUrls[6]).into(binding.imageOverige)
71            }
72        })
73
74        // Load photo URLs
75        viewModel.loadPhotoUrls()
76
77        return view
78    }
79
80    override fun onDestroyView() {
81        super.onDestroyView()
82        _binding = null
83    }
84 }
```

5.1.3.2 Uitleg code

Het fragment begint met de initialisatie van het View Binding-systeem, waarbij de bijbehorende binding variabele ``_binding`` wordt gemaakt voor het bijhouden van verwijzingen naar de lay-outelementen. Dit voorkomt geheugenlekken door de binding in de ``onDestroyView``-methode naar ``null`` te zetten.

De ViewModel, genaamd ``WinesCategoryViewModel``, wordt geïnitieerd om de logica en gegevens voor dit fragment te beheren. De binding van de ViewModel met de lay-out wordt tot stand gebracht.

Vervolgens wordt voor elk type wijn (bijv. Witte wijnen, Rode wijnen) een zogenaamd "layout-click listener" toegevoegd. Als een bepaald wijntype wordt geselecteerd, wordt het fragment genavigeerd naar een ander fragment (``winesFragment``) met de bijbehorende wijncategorie.

Daarnaast wordt een observer ingesteld op de ``photoUrlsLiveData`` van de ViewModel. Deze observeert wijzigingen in de lijst met foto-URL's en bij wijzigingen worden de afbeeldingen dynamisch bijgewerkt met behulp van de Picasso-bibliotheek.

Ten slotte wordt de ``loadPhotoUrls``-methode aangeroepen om de foto-URL's vanuit de ViewModel op te halen en te observeren.

In essentie biedt dit fragment een gestructureerde gebruikersinterface voor het selecteren van wijncategorieën, waarbij de ViewModel zorgt voor de afhandeling van gegevens en logica, en het View Binding-systeem de interactie tussen de lay-out en de code faciliteert.

5.1.4 WinesCategoryViewModel

5.1.4.1 Code

```
1 class WinesCategoryViewModel : ViewModel() {
2
3     private val winesCategoryRepository = WinesCategoryPhotosRepository.getInstance()
4
5     private val _photoUrlsLiveData: MutableLiveData<List<String>> = MutableLiveData()
6     val photoUrlsLiveData: LiveData<List<String>> = _photoUrlsLiveData
7
8     fun loadPhotoUrls() {
9         winesCategoryRepository.getPhotoUrls { photoUrls ->
10             _photoUrlsLiveData.postValue(photoUrls)
11         }
12     }
13 }
```

5.1.4.2 Uitleg code

In deze klasse wordt een instantie van ``WinesCategoryPhotosRepository`` gecreëerd met behulp van het Singleton-patroon. Dit zorgt ervoor dat gedurende de levensduur van de applicatie slechts één exemplaar van het repository bestaat. Het repository is verantwoordelijk voor het ophalen van foto-URL's uit de Firebase Realtime Database.

De klasse maakt gebruik van het LiveData-mechanisme om wijzigingen in de lijst met foto-URL's te volgen. Er wordt een ``MutableLiveData``-instantie genaamd ``_photoUrlsLiveData`` geïnitieerd, terwijl een publieke ``LiveData``-instantie genaamd ``photoUrlsLiveData`` wordt aangeboden voor externe observatie.

De methode ``loadPhotoUrls`` binnen deze klasse initieert het proces van het ophalen van foto-URL's vanuit het repository. Het resultaat, de lijst met foto-URL's, wordt vervolgens doorgegeven aan de ``_photoUrlsLiveData`` met behulp van de methode ``postValue``. Hierdoor kunnen externe componenten, zoals de XML-laag, dynamisch reageren op wijzigingen in de beschikbare foto-URL's.

5.1.5 Fragment_wines_category.xml

5.1.5.1 Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:orientation="vertical"
6     tools:context=".WinesCategoryFragment">
7
8     <data>
9         <variable
10             name="winesCategoryFragmentViewModel"
11             type="com.example.appwijnhuisronny.WinesCategoryViewModel" />
12     </data>
13
14     <ScrollView
15         android:layout_width="match_parent"
16         android:layout_height="match_parent">
17
18         <LinearLayout
19             android:layout_width="match_parent"
20             android:layout_height="wrap_content"
21             android:orientation="vertical">
22
23
24             <androidx.constraintlayout.widget.ConstraintLayout
25                 android:id="@+id/layout_witte_wijn"
26                 android:layout_width="match_parent"
27                 android:layout_height="0dp"
28                 android:layout_weight="1"
29                 android:onClick="onLayoutClick">
30
31                 <ImageView
32                     android:id="@+id/image_witte_wijn"
33                     android:background="@drawable/witte_wijn_achtergrond"
34                     android:layout_width="0dp"
35                     android:layout_height="0dp"
36                     android:scaleType="centerCrop"
37                     app:layout_constraintBottom_toBottomOf="parent"
38                     app:layout_constraintDimensionRatio="16:7"
39                     app:layout_constraintEnd_toEndOf="parent"
40                     app:layout_constraintStart_toStartOf="parent"
41                     app:layout_constraintTop_toTopOf="parent" />
42
43                 <TextView
44                     android:layout_width="wrap_content"
45                     android:layout_height="wrap_content"
46                     android:layout_alignParentBottom="true"
47                     android:layout_centerInParent="true"
48                     android:fontFamily="@font/nunito_sans_extralight"
49                     android:text="@string/witte_wijnen_category"
50                     android:textColor="@color/white"
51                     android:textSize="24sp"
52                     app:layout_constraintBottom_toBottomOf="parent"
53                     app:layout_constraintEnd_toEndOf="parent"
54                     app:layout_constraintStart_toStartOf="parent" />
55             </androidx.constraintlayout.widget.ConstraintLayout>
56
57             <androidx.constraintlayout.widget.ConstraintLayout
58                 android:id="@+id/layout_rode_wijn"
59                 android:layout_width="match_parent"
60                 android:layout_height="0dp"
61                 android:layout_weight="1"
62                 android:layout_marginTop="10dp"
63                 android:onClick="onLayoutClick">
64
65                 <ImageView
66                     android:id="@+id/image_rode_wijn"
67                     android:background="@drawable/witte_wijn_achtergrond"
68                     android:layout_width="0dp"
69                     android:layout_height="0dp"
70                     android:scaleType="centerCrop"
71                     app:layout_constraintBottom_toBottomOf="parent"
72                     app:layout_constraintDimensionRatio="16:7"
73                     app:layout_constraintEnd_toEndOf="parent"
74                     app:layout_constraintStart_toStartOf="parent"
75                     app:layout_constraintTop_toTopOf="parent" />
76
77
```

```

78         <TextView
79             android:layout_width="wrap_content"
80             android:layout_height="wrap_content"
81             android:layout_alignParentBottom="true"
82             android:layout_centerInParent="true"
83             android:fontFamily="@font/nunito_sans_extralight"
84             android:text="@string/rode_wijnen_category"
85             android:textColor="@color/white"
86             android:textSize="24sp"
87             app:layout_constraintBottom_toBottomOf="parent"
88             app:layout_constraintEnd_toEndOf="parent"
89             app:layout_constraintStart_toStartOf="parent" />
90
91     </androidx.constraintlayout.widget.ConstraintLayout>
92
93     <androidx.constraintlayout.widget.ConstraintLayout
94         android:id="@+id/layout_rose_wijn"
95         android:layout_width="match_parent"
96         android:layout_height="0dp"
97         android:layout_weight="1"
98         android:layout_marginTop="10dp"
99         android:onClick="onLayoutClick">
100
101         <ImageView
102             android:id="@+id/image_rose_wijn"
103             android:background="@drawable/witte_wijn_achtergrond"
104             android:layout_width="0dp"
105             android:layout_height="0dp"
106             android:scaleType="centerCrop"
107             app:layout_constraintBottom_toBottomOf="parent"
108             app:layout_constraintDimensionRatio="16:7"
109             app:layout_constraintEnd_toEndOf="parent"
110             app:layout_constraintStart_toStartOf="parent"
111             app:layout_constraintTop_toTopOf="parent" />
112
113         <TextView
114             android:layout_width="wrap_content"
115             android:layout_height="wrap_content"
116             android:layout_alignParentBottom="true"
117             android:layout_centerInParent="true"
118             android:fontFamily="@font/nunito_sans_extralight"
119             android:text="@string/rose_wijn_category"
120             android:textColor="@color/white"
121             android:textSize="24sp"
122             app:layout_constraintBottom_toBottomOf="parent"
123             app:layout_constraintEnd_toEndOf="parent"
124             app:layout_constraintStart_toStartOf="parent" />
125
126     </androidx.constraintlayout.widget.ConstraintLayout>
127
128     <androidx.constraintlayout.widget.ConstraintLayout
129         android:id="@+id/layout_bubbels_wijn"
130         android:layout_width="match_parent"
131         android:layout_height="0dp"
132         android:layout_weight="1"
133         android:layout_marginTop="10dp"
134         android:onClick="onLayoutClick">
135
136         <ImageView
137             android:id="@+id/image_bubbels"
138             android:background="@drawable/witte_wijn_achtergrond"
139             android:layout_width="0dp"
140             android:layout_height="0dp"
141             android:scaleType="centerCrop"
142             app:layout_constraintBottom_toBottomOf="parent"
143             app:layout_constraintDimensionRatio="16:7"
144             app:layout_constraintEnd_toEndOf="parent"
145             app:layout_constraintStart_toStartOf="parent"
146             app:layout_constraintTop_toTopOf="parent" />
147
148         <TextView
149             android:layout_width="wrap_content"
150             android:layout_height="wrap_content"
151             android:layout_alignParentBottom="true"
152             android:layout_centerInParent="true"
153             android:fontFamily="@font/nunito_sans_extralight"
154             android:text="@string/bubbles_category"
155             android:textColor="@color/white"
156             android:textSize="24sp"
157             app:layout_constraintBottom_toBottomOf="parent"
158             app:layout_constraintEnd_toEndOf="parent"
159             app:layout_constraintStart_toStartOf="parent" />
160
161     </androidx.constraintlayout.widget.ConstraintLayout>

```

```

162
163     <androidx.constraintlayout.widget.ConstraintLayout
164         android:id="@+id/layout_porto"
165         android:layout_width="match_parent"
166         android:layout_height="0dp"
167         android:layout_weight="1"
168         android:layout_marginTop="10dp"
169         android:onClick="onLayoutClick">
170
171         <ImageView
172             android:id="@+id/image_porto"
173             android:background="@drawable/witte_wijn_achtergrond"
174             android:layout_width="0dp"
175             android:layout_height="0dp"
176             android:scaleType="centerCrop"
177             app:layout_constraintBottom_toBottomOf="parent"
178             app:layout_constraintDimensionRatio="16:7"
179             app:layout_constraintEnd_toEndOf="parent"
180             app:layout_constraintStart_toStartOf="parent"
181             app:layout_constraintTop_toTopOf="parent" />
182
183         <TextView
184             android:layout_width="wrap_content"
185             android:layout_height="wrap_content"
186             android:layout_alignParentBottom="true"
187             android:layout_centerInParent="true"
188             android:fontFamily="@font/nunito_sans_extralight"
189             android:text="@string/porto_category"
190             android:textColor="@color/white"
191             android:textSize="24sp"
192             app:layout_constraintBottom_toBottomOf="parent"
193             app:layout_constraintEnd_toEndOf="parent"
194             app:layout_constraintStart_toStartOf="parent" />
195
196     </androidx.constraintlayout.widget.ConstraintLayout>
197
198     <androidx.constraintlayout.widget.ConstraintLayout
199         android:id="@+id/layout_gins"
200         android:layout_width="match_parent"
201         android:layout_height="0dp"
202         android:layout_weight="1"
203         android:layout_marginTop="10dp"
204         android:onClick="onLayoutClick">
205
206         <ImageView
207             android:id="@+id/image_gins"
208             android:background="@drawable/witte_wijn_achtergrond"
209             android:layout_width="0dp"
210             android:layout_height="0dp"
211             android:scaleType="centerCrop"
212             app:layout_constraintBottom_toBottomOf="parent"
213             app:layout_constraintDimensionRatio="16:7"
214             app:layout_constraintEnd_toEndOf="parent"
215             app:layout_constraintStart_toStartOf="parent"
216             app:layout_constraintTop_toTopOf="parent" />
217
218         <TextView
219             android:layout_width="wrap_content"
220             android:layout_height="wrap_content"
221             android:layout_alignParentBottom="true"
222             android:layout_centerInParent="true"
223             android:fontFamily="@font/nunito_sans_extralight"
224             android:text="@string/gin_category"
225             android:textColor="@color/white"
226             android:textSize="24sp"
227             app:layout_constraintBottom_toBottomOf="parent"
228             app:layout_constraintEnd_toEndOf="parent"
229             app:layout_constraintStart_toStartOf="parent" />
230
231     </androidx.constraintlayout.widget.ConstraintLayout>
232
233     <androidx.constraintlayout.widget.ConstraintLayout
234         android:id="@+id/layout_overige"
235         android:layout_width="match_parent"
236         android:layout_height="0dp"
237         android:layout_weight="1"
238         android:layout_marginTop="10dp"
239         android:onClick="onLayoutClick">
240

```

```

241         <ImageView
242             android:id="@+id/image_overige"
243             android:background="@drawable/witte_wijn_achtergrond"
244             android:layout_width="0dp"
245             android:layout_height="0dp"
246             android:scaleType="centerCrop"
247             app:layout_constraintBottom_toBottomOf="parent"
248             app:layout_constraintDimensionRatio="16:7"
249             app:layout_constraintEnd_toEndOf="parent"
250             app:layout_constraintStart_toStartOf="parent"
251             app:layout_constraintTop_toTopOf="parent" />
252
253         <TextView
254             android:layout_width="wrap_content"
255             android:layout_height="wrap_content"
256             android:layout_alignParentBottom="true"
257             android:layout_centerInParent="true"
258             android:fontFamily="@font/nunito_sans_extralight"
259             android:text="@string/overige_category"
260             android:textColor="@color/white"
261             android:textSize="24sp"
262             app:layout_constraintBottom_toBottomOf="parent"
263             app:layout_constraintEnd_toEndOf="parent"
264             app:layout_constraintStart_toStartOf="parent" />
265
266     </androidx.constraintlayout.widget.ConstraintLayout>
267 </LinearLayout>
268 </ScrollView>
269
270 </layout>

```

5.1.5.2 Uitleg code

De bovenstaande XML-code beschrijft de opmaak van een lay-outbestand binnen een Android-applicatie, specifiek voor het fragment met de naam "WinesCategoryFragment."

De gegevensbindingen worden gespecificeerd onder het "data" element, waarbij de variabele "winesCategoryFragmentViewModel" wordt gedeclareerd als van het type "com.example.appwijnhuisronny.WinesCategoryViewModel."

Binnen de lay-out is er een `ScrollView` die de mogelijkheid biedt om door een lijst met weergave-elementen te scrollen. Binnen deze `ScrollView` is een `LinearLayout` geplaatst met een verticale oriëntatie. Deze `LinearLayout` bevat verschillende `ConstraintLayout`-elementen, elk vertegenwoordigend een wijncategorie.

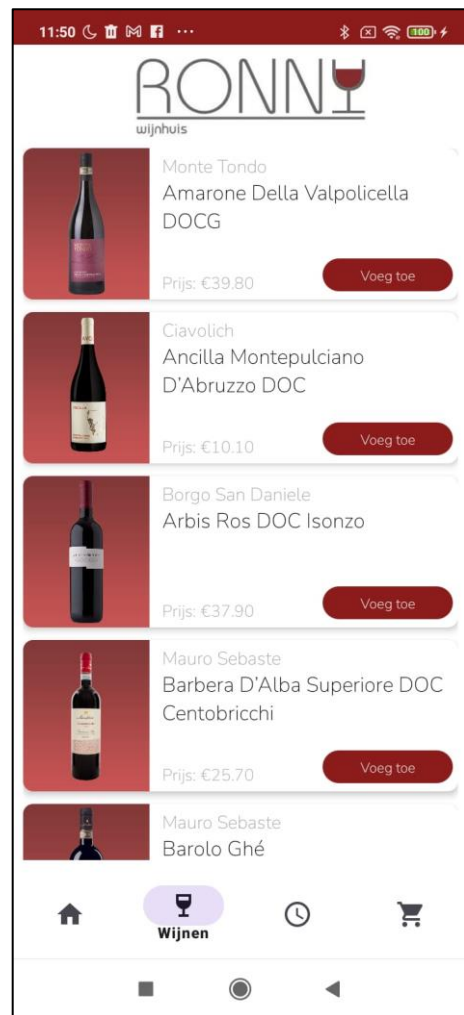
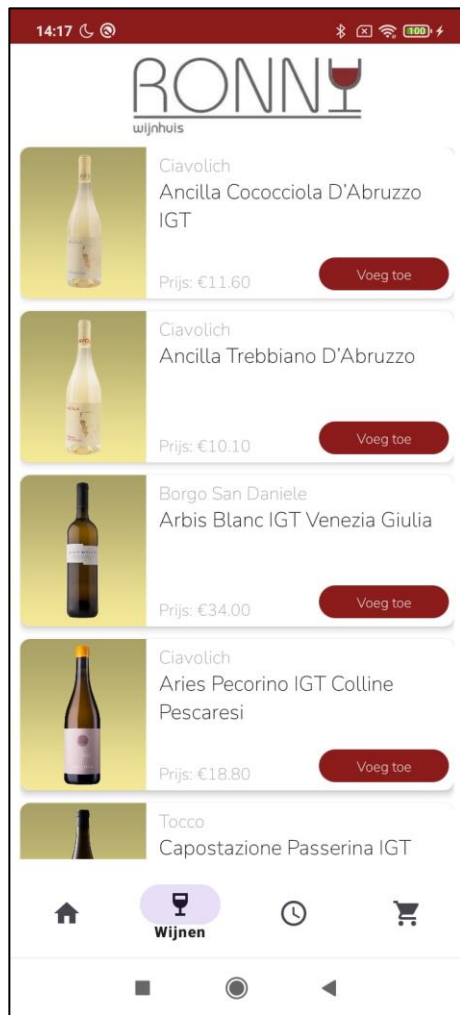
Elk `ConstraintLayout`-element bestaat uit een achtergrondaafbeelding (`ImageView`) en een tekstlabel (`TextView`). De afbeeldingen worden gespecificeerd met unieke identificatoren zoals "image_witte_wijn," "image_rode_wijn," enzovoort, elk met een bijbehorende achtergrondaafbeelding en tekstlabel voor de betreffende wijncategorie. De tekstlabels worden dynamisch geplaatst ten opzichte van de afbeeldingen met behulp van het ConstraintLayout.

Elk `ConstraintLayout`-element heeft een gewicht toegewezen, wat betekent dat ze gelijkmatig verdeeld worden in de beschikbare ruimte, waardoor ze elk een gelijk deel van het beschikbare schermgebied innemen.

Deze lay-out lijkt ontworpen te zijn voor een selectiescherm van wijncategorieën, waarbij elke categorie een achtergrondaafbeelding en een bijbehorende tekst heeft. De `onClick`-attributen zijn geconfigureerd om een gemeenschappelijke klikafhandelingsmethode genaamd "onLayoutClick" aan te roepen wanneer een gebruiker op een bepaalde categorie klikt.

5.2 Wijnen pagina

5.2.1 Visuele weergave



5.2.2 Wine Model

5.2.2.1 Code

```
1  data class Wine(  
2      var BackgroundImage: String? = null,  
3      var Image : String ?= null,  
4      var Druif : String ?= null,  
5      var Info : String ?= null,  
6      var Inhoud : String ?= null,  
7      var Naam : String ?= null,  
8      var Prijs : String ?= null,  
9      var Regio : String ?= null,  
10     var Wijndomein : String ?= null,  
11     var Aantal : Int = 1,  
12     var TotalPrice: Double = Prijs!!.toDouble()  
13 )
```

5.2.2.2 Uitleg code

De `Wine`-klasse is gemarkeerd met het sleutelwoord `data`, wat aangeeft dat deze klasse primair bedoeld is voor het vastleggen en overbrengen van gegevens. De klasse heeft verschillende eigenschappen (properties) die de kenmerken van een wijn representeren:

- `BackgroundImage`: Een optionele String die de achtergrondaafbeelding van de wijn vertegenwoordigt.
- `Image`: Een optionele String die de afbeelding van de wijn vertegenwoordigt.
- `Druif`: Een optionele String die de druivensoort van de wijn vertegenwoordigt.
- `Info`: Een optionele String die aanvullende informatie over de wijn bevat.
- `Inhoud`: Een optionele String die de inhoud van de wijn (bijv. 750 ml) vertegenwoordigt.
- `Naam`: Een optionele String die de naam van de wijn vertegenwoordigt.
- `Prijs`: Een optionele String die de prijs van de wijn vertegenwoordigt.
- `Regio`: Een optionele String die de regio van herkomst van de wijn vertegenwoordigt.
- `Wijndomein`: Een optionele String die het wijndomein van de wijn vertegenwoordigt.
- `Aantal`: Een optionele integer die het aantal flessen van de wijn vertegenwoordigt (standaard ingesteld op 1).
- `TotalPrice`: Een berekende eigenschap die de totale prijs van de wijn vertegenwoordigt, berekend als het product van de prijs en het aantal flessen.

Deze klasse biedt een gestructureerde manier om informatie over een wijn te organiseren en door te geven binnen mijn project. Het gebruik van optionele parameters (met een standaardwaarde van `null`) maakt het mogelijk om slechts een subset van eigenschappen in te vullen als dat nodig is, afhankelijk van de beschikbare informatie over een specifieke wijn.

5.2.3 WinesRepository

5.2.3.1 Code

```
1 class WinesRepository {
2     private val databaseReference: DatabaseReference = FirebaseDatabase.getInstance().getReference("Wijnen")
3     init {
4         Log.d("Repository", "Database reference initialized: $databaseReference")
5     }
6
7     @Volatile private var INSTANCE : WinesRepository? = null
8     fun getInstance(): WinesRepository {
9         Log.d("Repository", "Getting instance of WinesRepository")
10        return INSTANCE ?: synchronized(this) {
11            INSTANCE = WinesRepository()
12            INSTANCE!!
13        }
14    }
15
16    fun loadWines(category: String, winesList: MutableLiveData<List<Wine>>) {
17        val categoryReference = databaseReference.child(category)
18        categoryReference.addValueEventListener(object : ValueEventListener {
19            override fun onDataChange(snapshot: DataSnapshot) {
20                try {
21                    val _winesList: List<Wine> = snapshot.children.map { dataSnapshot ->
22                        val wineMap = dataSnapshot.value as Map<*, *>
23                        Wine(
24                            wineMap["BackgroundImage"] as? String,
25                            wineMap["Image"] as? String,
26                            wineMap["Druif"] as? String,
27                            wineMap["Info"] as? String,
28                            wineMap["Inhoud"] as? String,
29                            wineMap["Naam"] as? String,
30                            wineMap["Prijis"] as? String,
31                            wineMap["Regio"] as? String,
32                            wineMap["Wijndomein"] as? String
33                        )
34                    }
35                    winesList.postValue(_winesList)
36                } catch (e: Exception) {
37                    Log.w("Repository", "Failed to read:", e)
38                }
39            }
40            override fun onCancelled(error: DatabaseError) {
41                Log.w("Repository", "Failed to read value.", error.toException())
42            }
43        })
44    }
45 }
```

5.2.3.2 Uitleg code

De `WinesRepository`-klasse is verantwoordelijk voor de toegang tot en het ophalen van wijngerelateerde gegevens uit een Firebase Realtime Database.

De klasse heeft een private eigenschap `databaseReference` van het type `DatabaseReference`, die is geïnitieerd met de referentie naar de "Wijnen" knoop in de Firebase-database. Dit wordt gedaan in de `init`-blok, dat wordt uitgevoerd wanneer een instantie van de klasse wordt gemaakt. Er wordt een logboekmelding gegenereerd om aan te geven dat de database-referentie is geïnitieerd.

De klasse bevat een companion object met een `getInstance`-functie die verantwoordelijk is voor het ophalen van een instantie van de `WinesRepository`. De functie gebruikt een dubbele-check-vergrendeling (double-checked locking) om ervoor te zorgen dat er slechts één instantie van de repository wordt gemaakt.

De `loadWines`-functie is bedoeld voor het ophalen van wijnen uit de database op basis van een opgegeven categorie. Deze functie accepteert een categorie (bijvoorbeeld "RodeWijnen") en een `MutableLiveData`-object (`winesList`) dat de lijst met wijnen zal bevatten. Het maakt gebruik van een `ValueEventListener` om wijzigingen in de database bij te houden.

Binnen de `onDataChange`-methode wordt de snapshot van de database verwerkt. Voor elke kind-snapshot wordt een `Wine`-object gemaakt met behulp van de gegevens uit de database. Deze `Wine`-objecten worden verzameld in een lijst (`_winesList`), die vervolgens wordt gepost naar `winesList`.

5.2.4 WineAdapter

5.2.4.1 Code

```
1 class WineAdapter(private val addToCartClickListener: (Wine) -> Unit) : RecyclerView.Adapter<WineAdapter.View  
  wHolder>() {  
2  
3     private val winesList = ArrayList<Wine>()  
4     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
5         val itemView = LayoutInflater.from(parent.context).inflate(  
6             R.layout.wines_item,  
7             parent, false  
8         )  
9         return ViewHolder(itemView)  
10    }  
11  
12    override fun getItemCount(): Int {  
13        return winesList.size  
14    }  
15  
16    fun updateWinesList(winesList: List<Wine>) {  
17        this.winesList.clear()  
18        this.winesList.addAll(winesList)  
19        notifyDataSetChanged()  
20    }  
21  
22    override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
23        val currentItem = winesList[position]  
24  
25        holder.nameTextView.text = currentItem.Naam  
26        holder.priceTextView.text = currentItem.Prijs  
27        holder.wijndomeinTextView.text = currentItem.Wijndomein  
28        Picasso.get().load(currentItem.Image).into(holder.imageView)  
29        Picasso.get().load(currentItem.BackgroundImage).into(holder.backgroundImageView)  
30  
31        holder.addToCartButton.setOnClickListener {  
32            addToCartClickListener.invoke(currentItem)  
33            Log.d("WineAdapter", "Item added to cart: $currentItem")  
34        }  
35    }  
36  
37    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
38        val nameTextView: TextView = view.findViewById(R.id.textName)  
39        val priceTextView: TextView = view.findViewById(R.id.textPriceAmount)  
40        val wijndomeinTextView: TextView = view.findViewById(R.id.textWijndomein)  
41        val imageView: ImageView = view.findViewById(R.id.imageWine)  
42        val backgroundImageView: ImageView = view.findViewById(R.id.backgroundImage)  
43        val addToCartButton: Button = view.findViewById(R.id.addToCartButton)  
44    }  
45 }
```

5.2.4.2 Uitleg code

De `WineAdapter`-klasse is verantwoordelijk voor het beheren en weergeven van een lijst met wijnen in een `RecyclerView` in mijn project.

De klasse breidt `RecyclerView.Adapter<WineAdapter.ViewHolder>` uit, waarbij de generieke parameter `ViewHolder` de interne weergavehouderklasse is die is gedefinieerd binnen de `WineAdapter`.

De klasse heeft een private lijst (`winesList`) van `Wine`-objecten die in de `RecyclerView` worden weergegeven. Het bevat ook een `OnClickListener` (`addToCartClickListener`) die wordt gebruikt om te reageren wanneer een gebruiker op de knop "Toevoegen aan winkelwagen" klikt.

De `onCreateViewHolder`-functie wordt opgeroepen wanneer een nieuwe weergavehouder moet worden gemaakt. Het inflateert het lay-outbestand `wines_item.xml` om de weergavehouder te maken en retourneert deze.

De `getItemCount`-functie retourneert het aantal wijnen in de lijst.

De `updateWinesList`-functie wordt gebruikt om de lijst met wijnen bij te werken en de dataset opnieuw in te stellen. Deze functie wordt aangeroepen vanuit de bovenliggende klasse om de weergegeven wijnen dynamisch bij te werken.

De `onBindViewHolder`-functie wordt opgeroepen voor elk item in de lijst en bindt de gegevens van het huidige item aan de weergavehouder. Het gebruikt Picasso om de afbeeldingen van de wijn en de achtergrond in de respectievelijke `ImageView`-weergaven in te stellen.

De `ViewHolder`-klasse vertegenwoordigt de weergavehouder voor elk item in de `RecyclerView`. Het bevat verwijzingen naar de verschillende weergave-elementen (zoals tekstweergaven, afbeeldingsweergaven en knoppen) binnen elk item.

Deze adapter biedt een gestructureerde manier om de gegevens van `Wine`-objecten weer te geven in een `RecyclerView`. Het maakt gebruik van de `Picasso`-bibliotheek voor het laden van afbeeldingen en biedt functionaliteit voor het bijwerken van de weergegeven lijst en het omgaan met klikgebeurtenissen op de knop "Toevoegen aan winkelwagen".

5.2.5 WinesFragment

5.2.5.1 Code

```
1 class WinesFragment : Fragment() {
2
3     private var _binding: FragmentWinesBinding? = null
4     private val binding get() = _binding!!
5
6     private lateinit var viewModel: WinesFragmentViewModel
7     private lateinit var viewModel2: ShoppingCartViewModel
8
9     private lateinit var adapter: WineAdapter
10    private lateinit var winesRecyclerView: RecyclerView
11
12    override fun onCreate(savedInstanceState: Bundle?) {
13        super.onCreate(savedInstanceState)
14    }
15    override fun onCreateView(
16        inflater: LayoutInflater, container: ViewGroup?,
17        savedInstanceState: Bundle?
18    ): View? {
19        _binding = FragmentWinesBinding.inflate(inflater, container, false)
20        val view = binding.root
21
22        viewModel = ViewModelProvider(requireActivity()).get(WinesFragmentViewModel::class.java)
23        viewModel2 = ViewModelProvider(requireActivity()).get(ShoppingCartViewModel::class.java)
24
25        val category = arguments?.getString("category") ?: "default"
26        viewModel = ViewModelProvider(requireActivity()).get(WinesFragmentViewModel::class.java)
27        viewModel.init(category)
28
29        binding.whiteWinesFragmentViewModel = viewModel
30        binding.lifecycleOwner = viewLifecycleOwner
31
32        return view
33    }
34
35    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
36        super.onViewCreated(view, savedInstanceState)
37
38        winesRecyclerView = view.findViewById(R.id.wineRecyclerView)
39        winesRecyclerView.layoutManager = LinearLayoutManager(context)
40        winesRecyclerView.setHasFixedSize(true)
41
42        adapter = WineAdapter { wine ->
43            viewModel2.addToCart(wine)
44        }
45
46        winesRecyclerView.adapter = adapter
47
48        viewModel.allWines.observe(viewLifecycleOwner, Observer {
49            adapter.updateWinesList(it)
50            Log.d("WinesFragment", "Observed changes in allWines: $it")
51        })
52    }
53
54    override fun onDestroyView() {
55        super.onDestroyView()
56        _binding = null
57    }
58 }
```

5.2.5.2 Uitleg code

De klasse breidt `Fragment` uit, wat betekent dat het een fragment is dat wordt weergegeven binnen een `Activity` namelijk `activity_main`. Het bevat verschillende eigenschappen en functies die nodig zijn voor de weergave en interactie met de gebruiker.

Er is een `lateinit var viewModel: WinesFragmentViewModel` die wordt gebruikt om gegevens met betrekking tot wijnen op te halen en bij te houden. Er is ook een `lateinit var viewModel2: ShoppingCartViewModel` voor het beheren van het winkelwagentje.

Er wordt een `WineAdapter`-instantie (`adapter`) gedeclareerd om de wijnen weer te geven in een `RecyclerView`. De `RecyclerView` zelf wordt gedeclareerd als `lateinit var winesRecyclerView: RecyclerView`.

In de `onCreate`-functie wordt het fragment gemaakt. De `onCreateView`-functie wordt gebruikt om de lay-out van het fragment op te bouwen en de bijbehorende `ViewModel`s te initialiseren. Het laadt ook de wijnen voor een specifieke categorie.

In de `onViewCreated`-functie wordt verdere initialisatie uitgevoerd, waaronder het configureren van de `RecyclerView` met een `LinearLayoutManager` en het instellen van de adapter. Een observer wordt toegevoegd aan `viewModel.allWines`, zodat wijzigingen in de lijst met wijnen worden gedetecteerd en de adapter dienovereenkomstig wordt bijgewerkt.

De `onDestroyView`-functie wordt gebruikt om resources vrij te geven wanneer het fragment niet meer wordt weergegeven.

Kortom, de `WinesFragment`-klasse fungeert als een tussenliggende laag tussen de gebruikersinterface (weergegeven in de `RecyclerView` met behulp van de adapter) en de gegevensbron (vertegenwoordigd door `WinesFragmentViewModel` en `ShoppingCartViewModel`). Het initieert het ophalen van wijnen, reageert op gebruikersinteracties en zorgt voor de juiste weergave van wijnen binnen de app.

5.2.6 WinesFragmentViewModel

5.2.6.1 Code

```
1 class WinesFragmentViewModel : ViewModel() {
2
3     private val repository: WinesRepository = WinesRepository().getInstance()
4     private val _allWines = MutableLiveData<List<Wine>>()
5     val allWines: LiveData<List<Wine>> = _allWines
6
7     private val _shoppingCart = MutableLiveData<List<Wine>>()
8     val shoppingCart: LiveData<List<Wine>> = _shoppingCart
9
10    fun init(category: String) {
11        repository.loadWines(category, _allWines)
12        _shoppingCart.value = emptyList()
13        Log.d("ViewModel", "ViewModel initialized with category: $category")
14    }
15 }
```

5.2.6.2 Uitleg code

De `WinesFragmentViewModel`-klasse speelt als verbindingspunt tussen de gebruikersinterface en de gegevensbron, vertegenwoordigd door de `WinesRepository`. In deze klasse worden architecturale patronen, zoals het Singleton-patroon, toegepast.

De klasse maakt een enkele instantie van `WinesRepository` aan met behulp van het Singleton-patroon. Daarnaast bevat het twee `MutableLiveData`-objecten, `_allWines` en `_shoppingCart`, die respectievelijk de lijst met wijnen en het winkelwagentje vertegenwoordigen. Deze worden verder blootgesteld als `LiveData` om externe toegang en observatie mogelijk te maken.

De `init`-functie van de klasse is verantwoordelijk voor het initialiseren van het `ViewModel`. Het laadt de wijnen voor een specifieke categorie door een functie aan te roepen in de repository, wat resulteert in het bijwerken van `_allWines`. Tegelijkertijd wordt het winkelwagentje geïnitieerd als een lege lijst. Het geheel wordt afgerond met een logbericht dat de succesvolle initialisatie van het `ViewModel` met de opgegeven categorie aangeeft.

5.2.7 *Fragment_wines.xml*

5.2.7.1 Code

A screenshot of a code editor showing the XML layout for Fragment_wines.xml. The code is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       xmlns:app="http://schemas.android.com/apk/res-auto"
5       tools:context=".WinesFragment">
6
7     <data>
8         <variable
9             name="whiteWinesFragmentViewModel"
10            type="com.example.appwijnhuisronny.WinesFragmentViewModel" />
11     </data>
12
13     <LinearLayout
14         android:layout_width="match_parent"
15         android:layout_height="match_parent"
16         android:orientation="vertical">
17
18         <androidx.recyclerview.widget.RecyclerView
19             android:id="@+id/wineRecyclerView"
20             android:layout_width="match_parent"
21             android:layout_height="0dp"
22             android:layout_weight="1" />
23     </LinearLayout>
24
25 </layout>
```

5.2.7.2 Uitleg code

Binnen het `layout`-element wordt een `data`-element gebruikt om gegevensbinding mogelijk te maken. Het bevat een `variable`-element waarmee een variabele wordt gedefinieerd met de naam `"whiteWinesFragmentViewModel"`. Deze variabele wordt gebruikt om een referentie naar het bijbehorende `ViewModel` in de lay-out te houden.

Vervolgens wordt een `LinearLayout`-element gebruikt om de algemene structuur van de lay-out te beheren. Dit `LinearLayout` heeft de eigenschappen `match_parent` voor zowel de breedte als de hoogte, en heeft een verticale oriëntatie.

Binnen het `LinearLayout` bevindt zich een `RecyclerView`-element met de ID `"@+id/wineRecyclerView"`. Dit `RecyclerView`-element wordt gebruikt om de lijst met wijnen weer te geven. Het heeft `match_parent` voor de breedte, `0dp` voor de hoogte met een gewicht van 1. Hierdoor wordt de hoogte dynamisch aangepast, afhankelijk van andere weergaven in dezelfde lay-out.

5.2.8 Wines_item.xml

5.2.8.1 Code

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <layout xmlns:tools="http://schemas.android.com/tools"
3        xmlns:android="http://schemas.android.com/apk/res/android"
4        xmlns:app="http://schemas.android.com/apk/res-auto">
5
6      <data>
7          <variable
8              name="WhiteWinesFragmentViewModel"
9              type="com.example.appwijnhuisronny.WinesFragmentViewModel" />
10     </data>
11
12     <androidx.cardview.widget.CardView
13         android:id="@+id/wineCardView"
14         android:layout_height="wrap_content"
15         android:layout_width="match_parent"
16         app:cardCornerRadius="8dp"
17         app:cardElevation="4dp"
18         android:layout_marginBottom="10dp">
19         <LinearLayout
20             android:layout_width="match_parent"
21             android:layout_height="120dp"
22             android:orientation="horizontal">
23             <FrameLayout
24                 android:layout_width="100dp"
25                 android:layout_height="120dp"
26                 android:layout_marginEnd="10dp">
27                 <ImageView
28                     android:id="@+id/backgroundImage"
29                     android:layout_width="100dp"
30                     android:layout_height="120dp"
31                     android:scaleType="centerCrop"
32                     android:background="@color/white"/>
33                 <ImageView
34                     android:id="@+id/imageWine"
35                     android:layout_width="100dp"
36                     android:layout_height="120dp"
37                     android:scaleType="centerCrop"
38                     android:src="@drawable/cococciola_ciavolich_removebg_preview"/>
39             </FrameLayout>
40
41             <LinearLayout
42                 android:layout_width="match_parent"
43                 android:layout_height="wrap_content"
44                 android:layout_toEndOf="@id/imageWine"
45                 android:orientation="vertical">
46                 <TextView
47                     android:id="@+id/textWijndomein"
48                     android:layout_width="match_parent"
49                     android:layout_height="wrap_content"
50                     android:layout_marginTop="4dp"
51                     android:fontFamily="@font/nunito_sans_extralight"
52                     android:text="Ciavolich"
53                     android:textColor="@android:color/darker_gray"
54                     android:textSize="16sp" />
55
56                 <androidx.constraintlayout.widget.ConstraintLayout
57                     android:layout_width="match_parent"
58                     android:layout_height="92dp">
59
60                     <TextView
61                         android:id="@+id/textName"
62                         android:layout_width="match_parent"
63                         android:layout_height="wrap_content"
64                         android:fontFamily="@font/nunito_sans_extralight"
65                         android:text="Cocciola"
66                         android:textColor="@android:color/black"
67                         android:textSize="18sp"
68                         app:layout_constraintEnd_toEndOf="parent"
69                         app:layout_constraintStart_toStartOf="parent"
70                         app:layout_constraintTop_toTopOf="parent" />
71
72                     <TextView
73                         android:id="@+id/textPrice"
74                         android:layout_width="wrap_content"
75                         android:layout_height="wrap_content"
76                         android:fontFamily="@font/nunito_sans_extralight"
77                         android:text="Prijs:"
78                         android:textColor="@android:color/darker_gray"
79                         android:textSize="14sp"
80                         app:layout_constraintBottom_toBottomOf="parent"
81                         app:layout_constraintStart_toStartOf="parent" />
82
83                 </LinearLayout>
84             </LinearLayout>
85         </CardView>
86     </layout>
```

```

84         <TextView
85             android:id="@+id/textEuro"
86             android:layout_width="wrap_content"
87             android:layout_height="wrap_content"
88             android:layout_marginStart="4dp"
89             android:fontFamily="@font/nunito_sans_extralight"
90             android:text="@string/euroTeken"
91             android:textColor="@android:color/darker_gray"
92             android:textSize="14sp"
93             app:layout_constraintBottom_toBottomOf="parent"
94             app:layout_constraintStart_toEndOf="@+id/textPrice" />
95
96
97         <TextView
98             android:id="@+id/textPriceAmount"
99             android:layout_width="wrap_content"
100            android:layout_height="wrap_content"
101            android:fontFamily="@font/nunito_sans_extralight"
102            android:text="10"
103            android:textColor="@android:color/darker_gray"
104            android:textSize="14sp"
105            app:layout_constraintBottom_toBottomOf="parent"
106            app:layout_constraintStart_toEndOf="@+id/textEuro" />
107
108        <Button
109            android:id="@+id/addToCartButton"
110            android:layout_width="103dp"
111            android:layout_height="34dp"
112            android:layout_marginEnd="4dp"
113            android:layout_marginBottom="-2dp"
114            android:fontFamily="@font/nunito_sans_extralight"
115            android:text="@string/voegToe"
116            android:textSize="11dp"
117            app:layout_constraintBottom_toBottomOf="parent"
118            app:layout_constraintEnd_toEndOf="parent" />
119
120    </androidx.constraintlayout.widget.ConstraintLayout>
121    </LinearLayout>
122    </LinearLayout>
123    </androidx.cardview.widget.CardView>
124    </layout>

```

5.2.8.2 Uitleg code

De bovenstaande XML-code definieert de lay-out voor individuele wijnitems. De lay-out maakt gebruik van een `CardView` om een kaartachtige weergave te creëren voor elke wijn. Binnen deze kaart bevindt zich een horizontaal georiënteerd `LinearLayout`-element, waarin zowel de achtergrond- als de wijnafbelingen worden weergegeven.

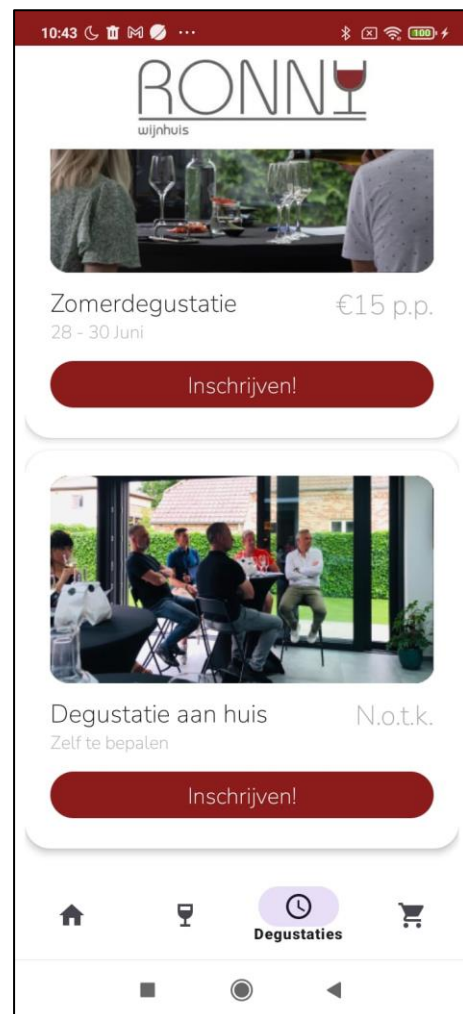
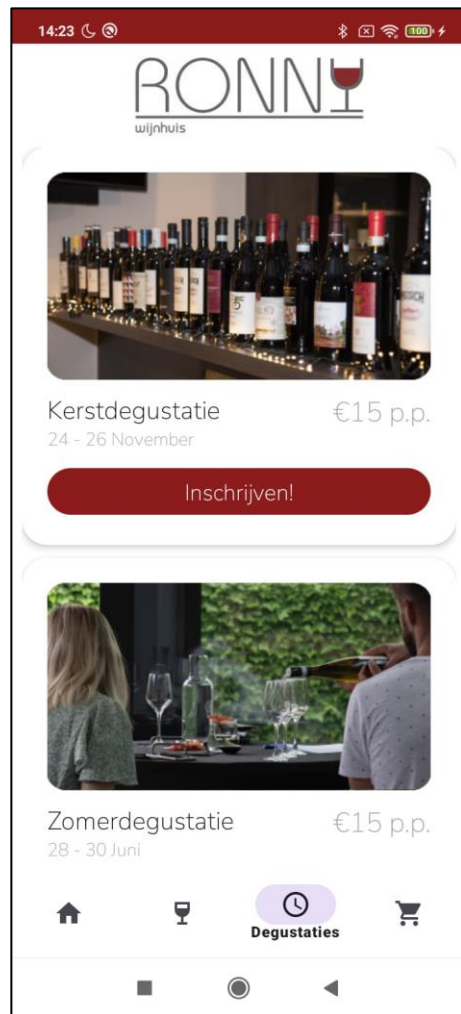
Een `FrameLayout` fungeert als container voor deze afbeeldingen, waarbij de achtergrondaafbeelding een achtergrond heeft en de wijnafbelding hier bovenop wordt geplaatst. Verschillende `TextView`s worden gebruikt om informatie over de wijn weer te geven, zoals het wijndomein, de naam en de prijs. Deze tekstweergaven zijn georganiseerd in een `ConstraintLayout` om een gestructureerde weergave te garanderen.

Het is belangrijk op te merken dat de weergegeven informatie dynamisch is en afkomstig is van de database. De koppeling tussen de gegevens en de lay-out wordt verzorgd door de `WineAdapter`. Deze adapter fungeert als een tussenliggende component die de gegevens vanuit de database, via het ViewModel (`WhiteWinesFragmentViewModel`), naar de daadwerkelijke lay-out in de `wines_item.xml` stuurt. Hierdoor wordt een naadloze weergave van de wijngegevens in de gebruikersinterface mogelijk gemaakt, en kunnen gebruikers met gemak door het assortiment bladeren en items aan hun winkelwagen toevoegen.

Bovendien bevat de lay-out een knop (`addToCartButton`) waarmee gebruikers een specifieke wijn aan hun winkelwagen kunnen toevoegen. Deze lay-out is ontworpen om een overzichtelijke en aantrekkelijke presentatie van wijninformatie mogelijk te maken, met duidelijke secties voor verschillende details zoals de naam van de wijn, het wijndomein en de prijs.

6. Degustatie pagina

6.1 Visuele weergave



6.2 WineTasting Model

6.2.1 Code

```
1 data class WineTasting(  
2     var DegustatieType: String? = null,  
3     var Datum : String ?= null,  
4     var DegustatiePrijs : String ?= null,  
5     var DegustatieImage : String ?= null  
6 )
```

6.2.2 Uitleg code

De `WineTasting`-klasse wordt gebruikt om informatie over te modelleren. Hier zijn de kenmerken:

- `DegustatieType`: Het type wijnproeverij.
- `Datum`: De datum van de wijnproeverij.
- `DegustatiePrijs`: De prijs van de wijnproeverij.
- `DegustatieImage`: De afbeelding (URL) die met de wijnproeverij is geassocieerd.

Deze eigenschappen zijn optioneel en kunnen null zijn als de bijbehorende informatie niet beschikbaar is. In essentie biedt de `WineTasting`-klasse een handige manier om gestructureerde gegevens over wijnproeverijen te representeren in de app.

6.3 WineTastingsRepository

6.3.1 Code

```
1 class WineTastingsRepository {
2     private val databaseReference: DatabaseReference = FirebaseDatabase.getInstance().getReference("Degustaties")
3
4     @Volatile private var INSTANCE : WineTastingsRepository ?= null
5
6     fun getInstance(): WineTastingsRepository {
7         return INSTANCE ?: synchronized(this) {
8             INSTANCE = WineTastingsRepository()
9             INSTANCE!!
10        }
11    }
12
13    fun loadWineTastings(wineTastingsList: MutableLiveData<List<WineTasting>>) {
14        databaseReference.addValueEventListener(object : ValueEventListener {
15            override fun onDataChange(snapshot: DataSnapshot) {
16                Log.d("Repository", "Data changed: $snapshot")
17                try {
18                    val _wineTastingsList: List<WineTasting> = snapshot.children.map { dataSnapshot ->
19                        val wineTastingsMap = dataSnapshot.value as Map<*, *>
20                        WineTasting(
21                            wineTastingsMap["DegustatieType"] as? String,
22                            wineTastingsMap["Datum"] as? String,
23                            wineTastingsMap["DegustatiePrijs"] as? String,
24                            wineTastingsMap["DegustatieImage"] as? String
25                        )
26                    }
27                    wineTastingsList.postValue(_wineTastingsList)
28                    Log.d("Repository", "Data loaded successfully")
29                } catch (e: Exception) {
30                    Log.w("Repository", "Failed to read:", e)
31                }
32            }
33            override fun onCancelled(error: DatabaseError) {
34                Log.w("Repository", "Failed to read value.", error.toException())
35            }
36        })
37    }
38 }
```

6.3.2 Uitleg code

De `WineTastingsRepository`-klasse is verantwoordelijk voor het beheren en ophalen van informatie over wijnproeverijen van een Firebase Realtime Database.

De klasse bevat een `databaseReference`, waarmee wordt verwezen naar het knooppunt "Degustaties" in de Firebase-database. Bovendien maakt het gebruik van een `Volatile`-instantievariabele genaamd `INSTANCE` om ervoor te zorgen dat slechts één instantie van de repository wordt gemaakt.

De methode `getInstance` implementeert het Singleton-ontwerppatroon, waarbij wordt gecontroleerd of er al een instantie van de repository bestaat. Als dat niet het geval is, wordt er een nieuwe aangemaakt en getourneerd.

De functie `loadWineTastings` wordt gebruikt om wijnproeverijgegevens uit de database te laden. Hiervoor wordt een `ValueEventListener` gebruikt om wijzigingen in de database bij te houden. Bij een succesvolle gegevenswijziging worden de gegevens gemapt naar een lijst van `WineTasting`-objecten en toegevoegd aan de `MutableLiveData` van `wineTastingsList`. In het geval van een fout wordt een logbericht gegenereerd.

Kort samengevat biedt deze repositoryklasse een gestructureerde manier om wijnproeverijgegevens op te halen en te beheren, waarbij gebruik wordt gemaakt van de Firebase Realtime Database.

6.4 WineTastingsAdapter

6.4.1 Code

```
1 class WineTastingsAdapter(private val clickListener: OnInschrijvenClickListener)
2 : RecyclerView.Adapter<WineTastingsAdapter.ViewHolder>() {
3     private val wineTastingsList = ArrayList<WineTasting>()
4     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
5         val itemView = LayoutInflater.from(parent.context).inflate(
6             R.layout.wine_tasting_item,
7             parent, false
8         )
9         return ViewHolder(itemView)
10    }
11
12    override fun getItemCount(): Int {
13        return wineTastingsList.size
14    }
15
16    fun updateWineTastingsList(wineTastingsList: List<WineTasting>) {
17        this.wineTastingsList.clear()
18        this.wineTastingsList.addAll(wineTastingsList)
19        notifyDataSetChanged()
20    }
21
22    interface OnInschrijvenClickListener {
23        fun onInschrijvenClick(wineTasting: WineTasting)
24    }
25    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
26        val currentItem = wineTastingsList[position]
27
28        holder.wineTastingType.text = currentItem.DegustatieType
29        holder.wineTastingPrice.text = currentItem.DegustatiePrijs
30        holder.wineTastingDate.text = currentItem.Datum
31        Picasso.get().load(currentItem.DegustatieImage).into(holder.imageView)
32
33        holder.wineTastingButton.setOnClickListener {
34            clickListener.onInschrijvenClick(currentItem)
35        }
36    }
37
38    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
39        val wineTastingType: TextView = view.findViewById(R.id.wineTastingType)
40        val wineTastingPrice: TextView = view.findViewById(R.id.wineTastingPrice)
41        val wineTastingDate: TextView = view.findViewById(R.id.wineTastingDate)
42        val imageView: ImageView = view.findViewById(R.id.wineTastingImage)
43        val wineTastingButton: Button = view.findViewById(R.id.wineTastingbutton)
44    }
45 }
```

6.4.2 Uitleg code

De adapter breidt de `RecyclerView.Adapter`-klasse uit en heeft een `ViewHolder`-klasse binnenin. Het bevat een lijst van `WineTasting`-objecten en biedt functionaliteit om deze gegevens aan de RecyclerView te binden.

In de `onCreateViewHolder`-methode wordt de weergave van een enkel wijnproeverij-item geïnfleteerd vanuit het XML-bestand `wine_tasting_item.xml` en toegevoegd aan een nieuwe `ViewHolder`.

De `getItemCount`-methode geeft het aantal wijnproeverijen in de lijst weer.

De `updateWineTastingsList`-methode wordt gebruikt om de lijst van wijnproeverijen bij te werken wanneer er wijzigingen zijn in de gegevens.

Er is een `OnInschrijvenClickListener`-interface gedefinieerd in de adapter, dat fungeert als een callback-mechanisme. Het vereist de implementatie van de `onInschrijvenClick`-methode, waarmee wordt gereageerd op klikgebeurtenissen op de inschrijvingsknop van een wijnproeverij.

In de `onBindViewHolder`-methode worden de gegevens van een specifieke wijnproeverij aan een `ViewHolder` toegewezen, en de bijbehorende weergave-elementen worden ingevuld met informatie zoals het type, de prijs, de datum en een afbeelding.

De `ViewHolder`-klasse bevat verwijzingen naar de weergave-elementen in een enkel wijnproeverij-item, zoals tekstvelden (`TextView`), een afbeelding (`ImageView`), en een knop (`Button`), die allemaal worden geïnitieerd in de `onCreateViewHolder`-methode.

6.5 WineTastingsFragment

6.5.1 Code

```
1 class WineTastingsFragment : Fragment(), WineTastingsAdapter.OnInschrijvenClickListener {
2
3     private var _binding: FragmentWineTastingsBinding? = null
4     private val binding get() = _binding!!
5     private lateinit var viewModel: WineTastingsViewModel
6
7     private lateinit var adapter: WineTastingsAdapter
8     private lateinit var wineTastingsRecyclerView: RecyclerView
9
10    override fun onCreateView(
11        inflater: LayoutInflater, container: ViewGroup?,
12        savedInstanceState: Bundle?
13    ): View? {
14        _binding = FragmentWineTastingsBinding.inflate(inflater, container, false)
15        val view = binding.root
16
17        viewModel = ViewModelProvider(this).get(WineTastingsViewModel::class.java)
18
19        binding.wineTastingsViewModel = viewModel
20        binding.lifecycleOwner = viewLifecycleOwner
21
22        return view
23    }
24
25    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
26        super.onViewCreated(view, savedInstanceState)
27
28        wineTastingsRecyclerView = binding.wineTastingsRecyclerView
29        wineTastingsRecyclerView.layoutManager = LinearLayoutManager(context)
30        wineTastingsRecyclerView.setHasFixedSize(true)
31
32        adapter = WineTastingsAdapter(this)
33        wineTastingsRecyclerView.adapter = adapter
34
35        viewModel.allWineTastings.observe(viewLifecycleOwner, Observer {
36            adapter.updateWineTastingsList(it)
37            Log.d("WinesFragment", "Observed changes in allWines: $it")
38        })
39    }
40
41    override fun onInschrijvenClick(wineTasting: WineTasting) {
42        try {
43            val subject = "Inschrijving degustatie"
44            val emailSub = "info@wijnhuisronny.be"
45
46            val intent = Intent(Intent.ACTION_SENDTO).apply {
47                data = Uri.parse("mailto:")
48                putExtra(Intent.EXTRA_EMAIL, arrayOf(emailSub))
49                putExtra(Intent.EXTRA_SUBJECT, subject)
50                putExtra(Intent.EXTRA_TEXT, viewModel.getBodyText(wineTasting))
51            }
52            startActivity(intent)
53        } catch (e: Exception) {
54            Log.e("CheckoutOrderFragment", "Error during checkout: ${e.message}")
55        }
56        view?.findNavController()?.navigate(R.id.action_wineTastingsFragment_to_homepageFragment)
57    }
58 }
59 }
```

6.5.2 Uitleg code

Deze klasse breidt de `Fragment`-klasse uit en implementeert ook het `OnInschrijvenClickListener`-interface van de bijbehorende adapter. Hierdoor kan het fragment reageren op klikgebeurtenissen op de inschrijvingsknop van een wijnproeverij.

In de `onCreateView`-methode wordt de lay-out van het fragment geïnflateerd vanuit het bijbehorende XML-bestand (`FragmentWineTastingsBinding`). De bijbehorende `ViewModel` (`WineTastingsViewModel`) wordt geïnitieerd en aan de lay-out gebonden voor databinding.

In de `onViewCreated`-methode worden verdere initialisaties uitgevoerd. De RecyclerView voor wijnproeverijen wordt ingesteld met een lineaire lay-outmanager en een aangepaste adapter (`WineTastingsAdapter`). De adapter wordt aan de RecyclerView toegewezen, en de observer wordt ingesteld om wijzigingen in de lijst met wijnproeverijen op te merken.

De methode `onInschrijvenClick` wordt geactiveerd wanneer er wordt geklikt op de inschrijvingsknop van een wijnproeverij. Deze methode probeert een e-mailintent te starten voor het inschrijven van een gebruiker voor de wijnproeverij. Het bevat informatie zoals het onderwerp, het e-mailadres van de ontvanger, en de inhoud van het bericht gegenereerd door de ViewModel. Als er een fout optreedt, wordt dit vastgelegd in de log.

Ten slotte wordt na de inschrijving genavigeerd naar het startscherm van de app (`homepageFragment`) met behulp van de navigatiecontroller.

In het algemeen biedt de klasse een gestructureerde weergave van wijnproeverijen met de mogelijkheid voor gebruikers om zich in te schrijven via e-mail.

6.6 WineTastingsViewModel

6.6.1 Code

```
1 class WineTastingsViewModel : ViewModel() {
2     fun getBodyText(wineTasting: WineTasting): String {
3         val stringBuilder = StringBuilder()
4
5         // Thank you message
6         stringBuilder.append("Ik/Wij zouden graag komen naar de: ${wineTasting.DegustatieType}!\n\n")
7         stringBuilder.append("Wij zijn in totaal met: ZELF IN TE VULLEN!\n")
8         stringBuilder.append("\n")
9         stringBuilder.append("Graag alle namen invullen: ZELF IN TE VULLEN!\n")
10        stringBuilder.append("\n")
11        stringBuilder.append("Prijs bedraagt €15 per persoon\n")
12        stringBuilder.append("Gelieve te betalen bij afhaal! (Cash/Payconiq)\n")
13
14        return stringBuilder.toString()
15    }
16
17    private val repository: WineTastingsRepository = WineTastingsRepository().getInstance()
18    private val _allWineTastings = MutableLiveData<List<WineTasting>>()
19    val allWineTastings: LiveData<List<WineTasting>> = _allWineTastings
20
21    init {
22        repository.loadWineTastings(_allWineTastings)
23    }
24 }
```

6.6.2 Uitleg code

Deze ViewModel bevat een methode genaamd `getBodyText`, die wordt gebruikt om de tekst voor de e-mailinhoud te genereren wanneer een gebruiker zich inschrijft voor een wijnproeverij. De methode maakt gebruik van een `StringBuilder` om de verschillende tekstuele elementen samen te voegen, waaronder een bedankbericht, het type wijnproeverij, ruimte voor het aantal deelnemers en namen, evenals informatie over de prijs en de betalingsmethode.

Daarnaast beheert de ViewModel ook de communicatie met de gegevensbron, in dit geval, de `WineTastingsRepository`. In de `init`-blok wordt bij het instantiëren van de klasse de repository gebruikt om de lijst met wijnproeverijen op te halen en te observeren. Eventuele wijzigingen in deze lijst worden automatisch doorgegeven.

6.7 Fragment_wine_tastings.xml

6.7.1 Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:orientation="vertical"
5       tools:context=".WineTastingsFragment">
6
7     <data>
8       <variable
9         name="wineTastingsViewModel"
10        type="com.example.appwijnhuisronny.WineTastingsViewModel" />
11     </data>
12     <ScrollView
13       android:layout_width="match_parent"
14       android:layout_height="match_parent">
15
16       <LinearLayout
17         android:layout_width="match_parent"
18         android:layout_height="match_parent"
19         android:orientation="vertical">
20
21         <androidx.recyclerview.widget.RecyclerView
22           android:id="@+id/wineTastingsRecyclerView"
23           android:layout_width="match_parent"
24           android:layout_height="0dp"
25           android:layout_weight="1"
26           tools:listitem="@layout/wine_tasting_item"
27           tools:itemCount="3"/>
28       </LinearLayout>
29
30     </ScrollView>
31 </layout>
```

6.7.2 Uitleg code

De bovenstaande XML-code beschrijft de lay-out van het fragment dat wordt gebruikt om een lijst met wijnproeverijen weer te geven. De lay-out is opgebouwd met behulp van het Android Data Binding-framework, wat betekent dat het fragment kan communiceren met de bijbehorende `WineTastingsViewModel`. Dit wordt bereikt door de variabele `wineTastingsViewModel` te declareren in het `data`-blok en te associëren met het ViewModel-type.

De belangrijkste container van de lay-out is een `ScrollView` om ervoor te zorgen dat gebruikers door de lijst met wijnproeverijen kunnen scrollen als deze de beschikbare schermruimte overschrijdt. Binnen de `ScrollView` bevindt zich een `LinearLayout` met een verticale oriëntatie, dat fungeert als de hoofdcontainer voor de inhoud van het fragment.

De kern van de lay-out is de `RecyclerView`, geïdentificeerd met de ID `wineTastingsRecyclerView`. Dit is een krachtige UI-component die wordt gebruikt om dynamische lijsten weer te geven. De `RecyclerView` haalt zijn gegevens uit de bijbehorende ViewModel en gebruikt een aangepaste lay-out (`wine_tasting_item.xml`) om elk afzonderlijk item in de lijst weer te geven. In de voorbeeldcode wordt gebruik gemaakt van tools voor het ontwerpen, waarbij drie dummy-items worden weergegeven.

Samengevat biedt deze lay-out een flexibele en schaalbare structuur om een lijst met wijnproeverijen weer te geven, met de mogelijkheid om gemakkelijk nieuwe items toe te voegen of te wijzigen door de ViewModel.

6.8 Wine_tasting_item.xml

6.8.1 Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:tools="http://schemas.android.com/tools"
3       xmlns:android="http://schemas.android.com/apk/res/android"
4       xmlns:app="http://schemas.android.com/apk/res-auto">
5
6     <data>
7         <variable
8             name="wineTastingViewModel"
9             type="com.example.appwijnhuisronny.WineTastingsViewModel" />
10    </data>
11
12    <androidx.cardview.widget.CardView
13        android:id="@+id/wineTastingCardView"
14        android:layout_height="wrap_content"
15        android:layout_width="match_parent"
16        app:cardCornerRadius="20dp"
17        app:cardElevation="4dp"
18        android:layout_marginBottom="10dp">
19
20        <androidx.constraintlayout.widget.ConstraintLayout
21            android:layout_width="match_parent"
22            android:layout_height="match_parent"
23            android:orientation="vertical">
24
25            <ImageView
26                android:id="@+id/wineTastingImage"
27                android:layout_width="match_parent"
28                android:layout_height="0dp"
29                android:layout_marginStart="20dp"
30                android:layout_marginTop="20dp"
31                android:layout_marginEnd="20dp"
32                android:scaleType="centerCrop"
33                android:src="@drawable/winterdegustatie"
34                app:layout_constraintDimensionRatio="24:13"
35                app:layout_constraintEnd_toEndOf="parent"
36                app:layout_constraintStart_toStartOf="parent"
37                app:layout_constraintTop_toTopOf="parent" />
38
39
40            <TextView
41                android:id="@+id/wineTastingType"
42                android:layout_width="wrap_content"
43                android:layout_height="wrap_content"
44                android:layout_marginStart="20dp"
45                android:layout_marginTop="10dp"
46                android:fontFamily="@font/nunito_sans_extralight"
47                android:text="Winterdegustatie"
48                android:textColor="@android:color/black"
49                android:textSize="22sp"
50                app:layout_constraintStart_toStartOf="parent"
51                app:layout_constraintTop_toBottomOf="@+id/wineTastingImage" />
52
53            <TextView
54                android:id="@+id/wineTastingPrice"
55                android:layout_width="wrap_content"
56                android:layout_height="wrap_content"
57                android:layout_marginTop="10dp"
58                android:layout_marginEnd="20dp"
59                android:fontFamily="@font/nunito_sans_extralight"
60                android:text="$15"
61                android:textColor="@android:color/darker_gray"
62                android:textSize="22dp"
63                app:layout_constraintEnd_toEndOf="parent"
64                app:layout_constraintTop_toBottomOf="@+id/wineTastingImage" />
65
```

```

66         <TextView
67             android:id="@+id/wineTastingDate"
68             android:layout_width="wrap_content"
69             android:layout_height="wrap_content"
70             android:layout_marginBottom="10dp"
71             android:fontFamily="@font/nunito_sans_extralight"
72             android:text="24 - 16 November"
73             android:textColor="@android:color/darker_gray"
74             android:textSize="16sp"
75             app:layout_constraintBottom_toTopOf="@+id/wineTastingbutton"
76             app:layout_constraintStart_toStartOf="@+id/wineTastingType"
77             app:layout_constraintTop_toBottomOf="@+id/wineTastingType" />
78
79         <Button
80             android:id="@+id/wineTastingbutton"
81             android:layout_width="match_parent"
82             android:layout_height="45dp"
83             android:layout_marginStart="20dp"
84             android:layout_marginEnd="20dp"
85             android:layout_marginBottom="20dp"
86             android:fontFamily="@font/nunito_sans_extralight"
87             android:text="Inschrijven!"
88             android:textSize="18dp"
89             android:onClick="onInschrijvenButtonClick"
90             app:layout_constraintBottom_toBottomOf="parent"
91             app:layout_constraintEnd_toEndOf="@+id/wineTastingImage"
92             app:layout_constraintStart_toStartOf="@+id/wineTastingImage" />
93
94     </androidx.constraintlayout.widget.ConstraintLayout>
95
96 </androidx.cardview.widget.CardView>
97 </layout>

```

6.8.2 Uitleg code

De XML-lay-out begint met de declaratie van de XML-versie en het `layout`-element, waarin de XML- en app-namespaces worden gedefinieerd. Binnen het `data`-element wordt een variabele genaamd `wineTastingViewModel` gedeclareerd, die is gekoppeld aan het type `WineTastingsViewModel`.

Het hoofdelement van de lay-out is een `CardView` met een unieke ID `wineTastingCardView`. Deze `CardView` fungeert als de container voor de visuele weergave van een wijnproeverij. De `CardView` heeft afgeronde hoeken (`cardCornerRadius`), een zachte schaduw (`cardElevation`), en een marginaal onderaan om enige ruimte te creëren tussen opeenvolgende items.

Binnen de `CardView` bevindt zich een `ConstraintLayout` die verschillende UI-elementen bevat, gepositioneerd met behulp van constraints. Een `ImageView` met de ID `wineTastingImage` toont de afbeelding van de wijnproeverij met schaaltype "centerCrop" en aangepaste dimensieratio.

Verschillende `TextView`-elementen worden gebruikt om informatie weer te geven, zoals de naam van de wijnproeverij (`wineTastingType`), de prijs (`wineTastingPrice`), en de datum (`wineTastingDate`). Deze elementen hebben aangepaste opmaak, zoals lettergrootte, kleur, en marges, om een aantrekkelijke visuele weergave te garanderen.

Tot slot is er een `Button` met de ID `wineTastingbutton` die is ontworpen voor het inschrijven voor de wijnproeverij. De knop heeft aangepaste opmaak en een `onClick`-attribuut dat is ingesteld op een methode genaamd "onInschrijvenButtonClick".

6.9 E-mailintent

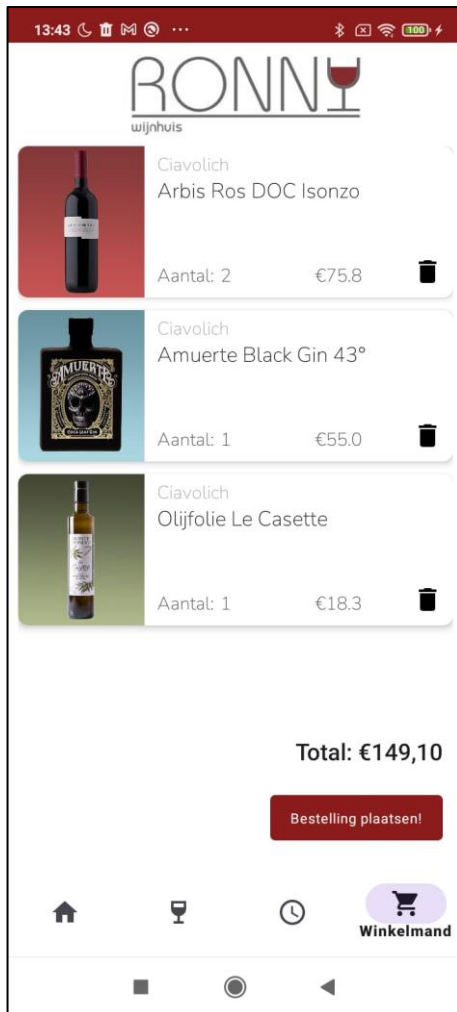
6.9.1 Visuele weergave



7. Winkelwagen

7.1 Winkelwagen pagina

7.1.1 Visuele weergave



7.1.2 ShoppingCartAdapter

7.1.2.1 Code

```
1 class ShoppingCartAdapter(private val deleteItemClickListener: (Wine) -> Unit)
2 : RecyclerView.Adapter<ShoppingCartAdapter.ViewHolder>() {
3
4     private val cartItems = mutableListOf<Wine>()
5
6     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
7         val itemView = LayoutInflater.from(parent.context).inflate(
8             R.layout.shoppingcart_row,
9             parent, false
10        )
11        return ViewHolder(itemView)
12    }
13
14    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
15        val currentItem = cartItems[position]
16
17        holder.productNameTextView.text = currentItem.Naam
18        holder.productTotalPriceTextView.text = currentItem.TotalPrice.toString()
19        holder.quantityTextView.text = currentItem.Aantal.toString()
20        Picasso.get().load(currentItem.Image).into(holder.productImageView)
21        Picasso.get().load(currentItem.BackgroundImage).into(holder.backgroundImageView)
22        holder.deleteItemButton.setOnClickListener {
23            deleteItemClickListener.invoke(currentItem)
24        }
25    }
26
27    override fun getItemCount(): Int {
28        return cartItems.size
29    }
30
31    fun updateCartItems(newItems: List<Wine>) {
32        cartItems.clear()
33        cartItems.addAll(newItems)
34        notifyDataSetChanged()
35        Log.d("ShoppingCartAdapter", "Updated cart items: $newItems")
36    }
37
38    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
39        val productNameTextView: TextView = view.findViewById(R.id.productNameTextView)
40        val productTotalPriceTextView: TextView = view.findViewById(R.id.productTotalPriceTextView)
41        val quantityTextView: TextView = view.findViewById(R.id.aantalTextView)
42        val productImageView: ImageView = view.findViewById(R.id.imageWine)
43        val backgroundImageView: ImageView = view.findViewById(R.id.backgroundImage)
44        val deleteItemButton: ImageButton = view.findViewById(R.id.deleteProductButton)
45    }
46 }
```

7.1.2.2 Uitleg code

De `ShoppingCartAdapter` breidt de `RecyclerView.Adapter` klasse uit en heeft een binnenklasse genaamd `ViewHolder`.

In de `ViewHolder`-klasse worden views gedefinieerd voor verschillende attributen van een winkelwagenitem, zoals de productnaam, totale prijs, hoeveelheid, productafbeelding en knop om een item te verwijderen.

De `ShoppingCartAdapter` heeft methoden om de winkelwagenitems bij te werken en om de juiste views in de `ViewHolder` in te vullen op basis van de gegeven dataset. De `deleteItemButton` heeft een klikgebeurtenis die wordt afgehandeld door de externe functie `deleteItemClickListener`, waardoor een item uit het winkelwagentje kan worden verwijderd.

De adapter is ontworpen om te worden gebruikt met een RecyclerView waarin elk item wordt weergegeven volgens de opgegeven lay-out gedefinieerd in `R.layout.shoppingcart_row`.

De methode `updateCartItems` wordt gebruikt om de dataset van de adapter bij te werken wanneer er wijzigingen zijn in de inhoud van het winkelwagentje. De dataset wordt bijgewerkt met nieuwe items en vervolgens wordt `notifyDataSetChanged` aangeroepen om de RecyclerView op de hoogte te stellen van de wijzigingen.

Deze `ShoppingCartAdapter` biedt een gestandaardiseerde manier om het winkelwagentje van de gebruiker weer te geven en maakt het gemakkelijk om interactie toe te voegen, zoals het verwijderen van items.

7.1.3 ShoppingCartFragment

7.1.3.1 Code

```
1 class ShoppingCartFragment : Fragment() {
2
3     private var _binding: FragmentShoppingCartBinding? = null
4     private val binding get() = _binding!!
5     private lateinit var viewModel: ShoppingCartViewModel
6     private lateinit var adapter: ShoppingCartAdapter
7
8     override fun onCreateView(
9         inflater: LayoutInflater, container: ViewGroup?,
10        savedInstanceState: Bundle?
11    ): View? {
12        _binding = FragmentShoppingCartBinding.inflate(inflater, container, false)
13        val view = binding.root
14
15        viewModel = ViewModelProvider(requireActivity()).get(ShoppingCartViewModel::class.java)
16
17        binding.placeOrderButton.setOnClickListener {
18            view.findNavController().navigate(R.id.action_shoppingCartFragment_to_checkoutOrderFragment)
19        }
20
21        binding.shoppingCartFragmentViewModel = viewModel
22        binding.lifecycleOwner = viewLifecycleOwner
23
24        return view
25    }
26
27    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
28        super.onViewCreated(view, savedInstanceState)
29        viewModel.shoppingCart.observe(viewLifecycleOwner) { cartItems ->
30            adapter.updateCartItems(cartItems)
31        }
32
33        viewModel.totalAmount.observe(viewLifecycleOwner) { totalAmount ->
34            val formattedTotal = String.format("Total: €%.2f", totalAmount)
35            binding.orderTotalTextView.text = formattedTotal
36        }
37
38        adapter = ShoppingCartAdapter { wine ->
39            viewModel.deleteItem(wine)
40        }
41        binding.shoppingCartRecyclerView.adapter = adapter
42        binding.shoppingCartRecyclerView.layoutManager = LinearLayoutManager(context)
43    }
44 }
```

7.1.3.2 Uitleg code

De `ShoppingCartFragment` breidt de `Fragment` klasse uit en maakt gebruik van View Binding via de `FragmentShoppingCartBinding` klasse om toegang te krijgen tot de views in de bijbehorende lay-out.

In de `onCreateView`-methode wordt de databinding opgezet, de ViewModel geïnitieerd en een klikgebeurtenis toegevoegd aan de knop om de bestelling te plaatsen (`placeOrderButton`). Bij het klikken op deze knop wordt de navigatie naar het afrekeningsfragment geactiveerd.

De `onViewCreated`-methode komt na `onCreateView` en wordt gebruikt om logica uit te voeren nadat de bijbehorende lay-out is gemaakt. Hier wordt de RecyclerView geconfigureerd met een `ShoppingCartAdapter` en een `LinearLayoutManager`. De adapter wordt geïnitieerd met een lambda-functie die wordt aangeroepen wanneer een item uit het winkelwagentje moet worden verwijderd.

De `ViewModel` van het fragment wordt geobserveerd voor wijzigingen in het winkelwagentje en het totaalbedrag. Bij elke wijziging worden de bijbehorende views bijgewerkt via de adapter.

Deze klasse biedt een gestandaardiseerde manier om het winkelwagentje van de gebruiker weer te geven, interacties toe te voegen, en maakt gebruik van het Android Navigation Component voor fragmentnavigatie.

7.1.4 ShoppingCartViewModel

7.1.4.1 Code

```
1 class ShoppingCartViewModel : ViewModel() {
2     private val _shoppingCart = MutableLiveData<List<Wine>>()
3     val shoppingCart: LiveData<List<Wine>> = _shoppingCart
4     private val _totalAmount = MutableLiveData(0.00)
5     val totalAmount: LiveData<Double> = _totalAmount
6
7     fun addToCart(item: Wine) {
8         val currentList = _shoppingCart.value ?: emptyList()
9         val updatedList = currentList.toMutableList()
10
11         val existingItem = updatedList.find { it == item }
12
13         if (existingItem == null) {
14             updatedList.add(item)
15         } else {
16             existingItem.Aantal++
17             existingItem.TotalPrice = existingItem.Prijs!!.toDouble() * existingItem.Aantal
18         }
19
20         _shoppingCart.value = updatedList
21
22         updateTotalAmount()
23     }
24
25     fun deleteItem(item: Wine) {
26         val currentList = _shoppingCart.value ?: emptyList()
27         val updatedList = currentList.toMutableList()
28         val existingItem = updatedList.find { it == item }
29
30         if (existingItem != null) {
31             if (existingItem.Aantal > 1) {
32                 existingItem.Aantal--
33                 existingItem.TotalPrice = existingItem.Prijs!!.toDouble() * existingItem.Aantal
34             } else {
35                 updatedList.remove(existingItem)
36             }
37             _shoppingCart.value = updatedList
38
39             updateTotalAmount()
40         }
41     }
42 }
```

```

43     private fun updateTotalAmount() {
44         val currentList = _shoppingCart.value ?: emptyList()
45         val total = currentList.sumByDouble { it.TotalPrice ?: 0.00 }
46         _totalAmount.postValue(total)
47     }
48
49     fun getShoppingCart(): List<Wine> {
50         val wines = _shoppingCart.value ?: emptyList()
51         return wines
52     }
53
54     fun getTotalAmount(): Double {
55         val total = _totalAmount.value ?: 0.00
56         return total
57     }
58 }

```

7.1.4.2 Uitleg code

De `ShoppingCartViewModel` klasse is verantwoordelijk voor het beheren van gegevens en logica met betrekking tot het winkelwagentje van de gebruiker.

De klasse erft van `ViewModel` en maakt gebruik van LiveData om wijzigingen in het winkelwagentje en het totaalbedrag bij te houden. In de klasse worden verschillende functies geïmplementeerd om items toe te voegen of te verwijderen uit het winkelwagentje, en om de totale kosten bij te werken.

De `addToCart`-functie wordt gebruikt om een item aan het winkelwagentje toe te voegen. Hierbij wordt eerst de huidige lijst met winkelwagenitems opgehaald, en indien het toegevoegde item al in het winkelwagentje zit, wordt het aantal verhoogd en de totale prijs bijgewerkt. Als het item nog niet in het winkelwagentje zit, wordt het toegevoegd aan de lijst. Vervolgens wordt het winkelwagentje bijgewerkt en wordt de totale prijs bijgewerkt via de functie `updateTotalAmount`.

De `deleteItem`-functie wordt gebruikt om een item uit het winkelwagentje te verwijderen. Het aantal van het te verwijderen item wordt verlaagd, en als het aantal nul is geworden, wordt het item volledig uit het winkelwagentje verwijderd. Ook hier wordt het winkelwagentje bijgewerkt en de totale prijs wordt opnieuw berekend via `updateTotalAmount`.

De `updateTotalAmount`-functie berekent het totaalbedrag van het winkelwagentje door de totale prijs van elk item op te tellen. Dit totaalbedrag wordt vervolgens bijgewerkt in de LiveData `_totalAmount`.

Tot slot zijn er twee hulpprogrammafuncties, `getShoppingCart` en `getTotalAmount`, die eenvoudige toegang bieden tot de huidige lijst met winkelwagenitems en het totaalbedrag.

7.1.5 Fragment_shopping_cart.xml

7.1.5.1 Code

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <layout xmlns:android="http://schemas.android.com/apk/res/android"
3        xmlns:tools="http://schemas.android.com/tools"
4        xmlns:app="http://schemas.android.com/apk/res-auto"
5        tools:context=".ShoppingCartFragment">
6
7      <data>
8        <variable
9          name="shoppingCartFragmentViewModel"
10         type="com.example.appwijnhuisronny.ShoppingCartViewModel" />
11      </data>
12
13      <LinearLayout
14        android:layout_width="match_parent"
15        android:layout_height="match_parent"
16        android:orientation="vertical">
17
18        <ScrollView
19          android:layout_width="match_parent"
20          android:layout_height="0dp"
21          android:layout_weight="1">
22
23          <LinearLayout
24            android:layout_width="match_parent"
25            android:layout_height="wrap_content"
26            android:orientation="vertical">
27
28            <androidx.recyclerview.widget.RecyclerView
29              android:id="@+id/shoppingCartRecyclerView"
30              android:layout_width="match_parent"
31              android:layout_height="wrap_content"
32              app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
33              tools:listitem="@layout/shoppingcart_row"
34              tools:itemCount="2" />
35
36            <Space
37              android:layout_width="match_parent"
38              android:layout_height="16dp" />
39
40          </LinearLayout>
41        </ScrollView>
42
43        <TextView
44          android:id="@+id/orderTotalTextView"
45          android:layout_width="wrap_content"
46          android:layout_height="wrap_content"
47          android:layout_gravity="end"
48          android:layout_margin="8dp"
49          android:text="@string/totalValue"
50          android:textAppearance="@style/TextAppearance.MaterialComponents.Headline6" />
51
52        <Button
53          android:id="@+id/placeOrderButton"
54          style="@style/Widget.MaterialComponents.Button.UnelevatedButton"
55          android:layout_width="wrap_content"
56          android:layout_height="wrap_content"
57          android:layout_gravity="end"
58          android:layout_margin="8dp"
59          android:text="@string/bestellingPlaatsen"
60          android:textAppearance="@style/TextAppearance.MaterialComponents.Caption" />
61      </LinearLayout>
62    </layout>
```

7.1.5.2 Uitleg code

Binnen het `LinearLayout`-element, dat de volledige weergave van het fragment omvat, wordt een `ScrollView` toegevoegd om het scherm scrollbaar te maken als er te veel items zijn om op één scherm weer te geven. Hierbinnen bevindt zich een ander `LinearLayout` met een verticale oriëntatie, waarin de hoofdinhoud van het winkelwagentje wordt geplaatst.

Een `RecyclerView` met de id `shoppingCartRecyclerView` wordt gebruikt om de daadwerkelijke items in het winkelwagentje weer te geven. Het heeft een aangepaste lay-outmanager voor verticale weergave (`LinearLayoutManager`). De lay-out voor elk item wordt bepaald door het XML-bestand `shoppingcart_row`. De `tools:itemCount` attributen worden gebruikt voor het voorbeeld van ontwerptijd om enkele items weer te geven.

Tussen de `RecyclerView` en de eindmarge van het scherm bevindt zich een lege ruimte (`Space`), wat bijdraagt aan de visuele scheiding tussen de items in het winkelwagentje en de onderkant van het scherm.

Onderaan het scherm bevinden zich twee extra views: een `TextView` met de id `orderTotalTextView` en een `Button` met de id `placeOrderButton`. De `TextView` wordt gebruikt om het totaalbedrag van de winkelwagen weer te geven, met de tekst die wordt ingesteld op een stringbron genaamd `totalValue`. De `Button` dient als een knop om de bestelling te plaatsen, waarbij de tekst wordt ingesteld op een stringbron genaamd `bestellingPlaatsen`.

Beide views zijn uitgelijnd aan de rechterkant van het scherm en hebben marges voor de juiste positionering. De `Button` heeft ook een stijl toegepast voor een ongeaccentueerde knop (`Widget.MaterialComponents.Button.UnelevatedButton`), en de tekstweergave wordt aangepast met een specifieke stijl (`TextAppearance.MaterialComponents.Caption`). Hierdoor wordt een gestructureerde en gebruiksvriendelijke lay-out gecreëerd voor het weergeven van de winkelwageninhoud en het uitvoeren van bestelacties.

7.1.6 Shoppingcart_row.xml

7.1.6.1 Code

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <layout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools">
5
6     <data>
7         <variable
8             name="shoppingCartFragmentViewModel"
9             type="com.example.appwijnhuisronny.ShoppingCartViewModel" />
10    </data>
11
12    <androidx.cardview.widget.CardView
13        android:id="@+id/wineCardView"
14        android:layout_height="wrap_content"
15        android:layout_width="match_parent"
16        app:cardCornerRadius="8dp"
17        app:cardElevation="4dp"
18        android:layout_marginBottom="10dp">
19        <LinearLayout
20            android:layout_width="match_parent"
21            android:layout_height="120dp"
22            android:orientation="horizontal">
23            <FrameLayout
24                android:layout_width="100dp"
25                android:layout_height="120dp"
26                android:layout_marginEnd="10dp">
27                <ImageView
28                    android:id="@+id/backgroundImage"
29                    android:layout_width="100dp"
30                    android:layout_height="120dp"
31                    android:scaleType="centerCrop"
32                    android:background="@color/white"/>
33                <ImageView
34                    android:id="@+id/imageWine"
35                    android:layout_width="100dp"
36                    android:layout_height="120dp"
37                    android:scaleType="centerCrop"
38                    android:src="@drawable/cocciola_ciavolich_removebg_preview"/>
39            </FrameLayout>
40
41            <LinearLayout
42                android:layout_width="match_parent"
43                android:layout_height="wrap_content"
44                android:layout_toEndOf="@id/imageWine"
45                android:orientation="vertical">
46                <TextView
47                    android:id="@+id/textWijndomein"
48                    android:layout_width="match_parent"
49                    android:layout_height="wrap_content"
50                    android:layout_marginTop="4dp"
51                    android:fontFamily="@font/nunito_sans_extralight"
52                    android:text="Ciavolich"
53                    android:textColor="@android:color/darker_gray"
54                    android:textSize="16sp" />
55
56                <androidx.constraintlayout.widget.ConstraintLayout
57                    android:layout_width="match_parent"
58                    android:layout_height="92dp">
59
60                    <TextView
61                        android:id="@+id/productNameTextView"
62                        android:layout_width="match_parent"
63                        android:layout_height="wrap_content"
64                        android:fontFamily="@font/nunito_sans_extralight"
65                        android:text="Cocciola"
66                        android:textColor="@android:color/black"
67                        android:textSize="18sp"
68                        app:layout_constraintEnd_toEndOf="parent"
69                        app:layout_constraintStart_toStartOf="parent"
70                        app:layout_constraintTop_toTopOf="parent" />
71
72                    <TextView
73                        android:id="@+id/QtyTextView"
74                        android:layout_width="wrap_content"
75                        android:layout_height="wrap_content"
76                        android:layout_marginBottom="4dp"
77                        android:fontFamily="@font/nunito_sans_extralight"
78                        android:text="@string/aantal"
79                        android:textSize="16sp"
80                        app:layout_constraintBottom_toBottomOf="parent"
81                        app:layout_constraintStart_toStartOf="parent" />
82
83                </LinearLayout>
84            </LinearLayout>
85        </CardView>
86    </layout>
```



```

84         <TextView
85             android:id="@id/aantalTextView"
86             android:layout_width="wrap_content"
87             android:layout_height="wrap_content"
88             android:layout_marginStart="5dp"
89             android:fontFamily="@font/nunito_sans_extralight"
90             android:text="6"
91             android:textSize="16sp"
92             app:layout_constraintBottom_toBottomOf="@id/QtyTextView"
93             app:layout_constraintStart_toEndOf="@id/QtyTextView"
94             app:layout_constraintTop_toTopOf="@id/QtyTextView" />
95         <ImageButton
96             android:id="@id/deleteProductButton"
97             android:layout_width="wrap_content"
98             android:layout_height="wrap_content"
99             android:background="?android:attr/selectableItemBackground"
100            android:paddingTop="8dp"
101            android:paddingBottom="4dp"
102            android:paddingLeft="8dp"
103            android:paddingRight="8dp"
104            app:layout_constraintBottom_toBottomOf="parent"
105            app:layout_constraintEnd_toEndOf="parent"
106            app:srcCompat="@drawable/baseline_delete_24" />
107         <TextView
108             android:layout_width="wrap_content"
109             android:layout_height="wrap_content"
110             android:fontFamily="@font/nunito_sans_extralight"
111             android:text="@string/euroTeken"
112             android:textSize="16sp"
113             app:layout_constraintBottom_toBottomOf="@id/productTotalPriceTextView"
114             app:layout_constraintEnd_toStartOf="@id/productTotalPriceTextView" />
115         <TextView
116             android:id="@id/productTotalPriceTextView"
117             android:layout_width="wrap_content"
118             android:layout_height="wrap_content"
119             android:layout_marginEnd="32dp"
120             android:fontFamily="@font/nunito_sans_extralight"
121             android:text="12.55"
122             android:textSize="16sp"
123             app:layout_constraintBottom_toBottomOf="@id/aantalTextView"
124             app:layout_constraintEnd_toStartOf="@id/deleteProductButton"
125             app:layout_constraintTop_toTopOf="@id/aantalTextView" />
126
127     </androidx.constraintlayout.widget.ConstraintLayout>
128 </LinearLayout>
129 </LinearLayout>
130 </androidx.cardview.widget.CardView>
131 </layout>

```

7.1.6.2 Uitleg code

De bovenstaande XML-code beschrijft de lay-out van een individueel item in het winkelwagentje van de app. De lay-out maakt gebruik van een `CardView` met afgeronde hoeken en subtiele elevatie om een visueel aantrekkelijke kaartachtige weergave te realiseren. Binnen deze kaart zijn twee belangrijke secties: een `FrameLayout` voor afbeeldingen en een verticale `LinearLayout` voor tekstuele details.

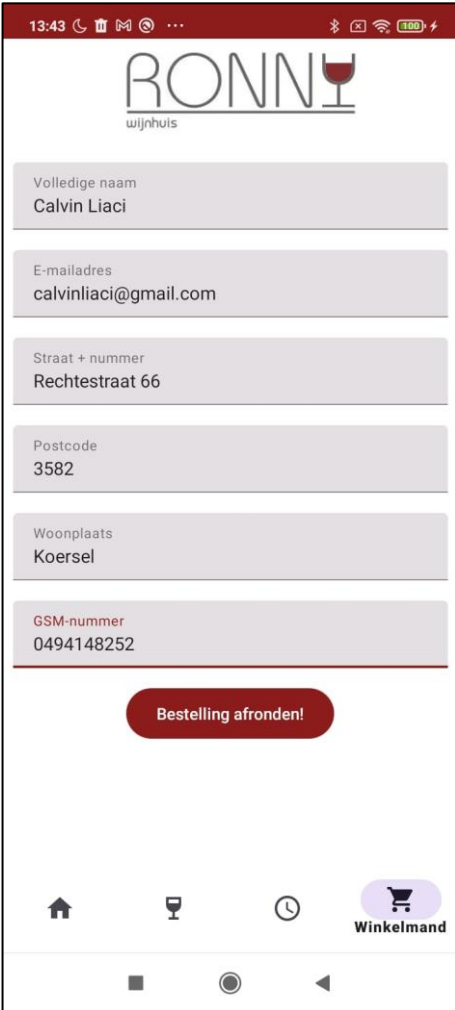
In het `FrameLayout` worden twee `ImageView`-elementen geplaatst. De eerste, met de id `backgroundImage`, fungeert als een achtergrondafbeelding met een witte kleur als fallback. De tweede, met de id `imageWine`, bevat de hoofdafbeelding van de wijn.

De verticale `LinearLayout` bevat tekstuele details van het wijnittem, waaronder het wijndomein, weergegeven in een aparte `TextView` met aangepaste tekstgrootte en kleur. Daarna volgt een `ConstraintLayout`, waarin verschillende tekstweergaven zijn opgenomen, zoals de naam van het product, het label 'Aantal', het daadwerkelijke aantal, het prijssymbool ('€'), de totaalprijs en een verwijderknop in de vorm van een `ImageButton`.

Elk tekstueel element is opgemaakt met aangepaste lettertypes, tekstgroottes en kleuren om een consistente en aantrekkelijke weergave te garanderen. De afbeeldingen worden geladen met behulp van `ImageView`-elementen, waarbij de `Picasso`-bibliotheek wordt gebruikt. Over het geheel genomen is de lay-out ontworpen om een overzichtelijke en aantrekkelijke presentatie van winkelwagenitems mogelijk te maken, met duidelijke secties voor afbeeldingen en tekstuele details.

7.2 Checkout pagina

7.2.1 Visuele weergave



The screenshot shows a mobile application interface for 'RONNY wijnhuis'. The status bar at the top indicates the time is 13:43 and shows various system icons. The app's logo, 'RONNY' with a wine glass icon and 'wijnhuis' below it, is at the top. The checkout form consists of several input fields with labels and pre-filled values: 'Volledige naam' (Calvin Liaci), 'E-mailadres' (calvinliaci@gmail.com), 'Straat + nummer' (Rechtestraat 66), 'Postcode' (3582), 'Woonplaats' (Koersel), and 'GSM-nummer' (0494148252). A red button labeled 'Bestelling afronden!' is positioned below the form. At the bottom, there is a navigation bar with icons for home, a wine glass, a clock, and a shopping cart labeled 'Winkelmand'. The Android navigation bar is visible at the very bottom.

13:43

RONNY
wijnhuis

Volledige naam
Calvin Liaci

E-mailadres
calvinliaci@gmail.com

Straat + nummer
Rechtestraat 66

Postcode
3582

Woonplaats
Koersel

GSM-nummer
0494148252

Bestelling afronden!

Winkelmand

7.2.2 CheckoutOrderFragment

7.2.2.1 Code

```
1 class CheckoutOrderFragment : Fragment() {
2     private var _binding: FragmentCheckoutOrderBinding? = null
3     private val binding get() = _binding!!
4
5     private lateinit var viewModel: CheckoutOrderViewModel
6
7     lateinit var shoppingCartViewModel: ShoppingCartViewModel
8
9
10    override fun onCreateView(
11        inflater: LayoutInflater, container: ViewGroup?,
12        savedInstanceState: Bundle?
13    ): View? {
14        _binding = FragmentCheckoutOrderBinding.inflate(inflater, container, false)
15        val view = binding.root
16
17        viewModel = ViewModelProvider(this).get(CheckoutOrderViewModel::class.java)
18        shoppingCartViewModel = ViewModelProvider(requireActivity()).get(ShoppingCartViewModel::class.java)
19
20        binding.orderDetailsViewModel = viewModel
21        binding.lifecycleOwner = viewLifecycleOwner
22
23        binding.checkoutButton.setOnClickListener {
24            try {
25                val subject = "Bestelbevestiging"
26                val emailSub = "info@wijnhuisronny.be"
27
28                val name = binding.nameEditText.text.toString().trim()
29                val email = binding.mailEditText.text.toString().trim()
30                val address = binding.addressEditText.text.toString().trim()
31                val zipCode = binding.zipCodeEditText.text.toString().trim()
32                val country = binding.countryEditText.text.toString().trim()
33                val phoneNumber = binding.phoneEditText.text.toString().trim()
34
35                val orderDetails = OrderDetails(
36                    name = name,
37                    email = email,
38                    address = address,
39                    zipCode = zipCode,
40                    country = country,
41                    phoneNumber = phoneNumber,
42                )
43
44                val cartItems = shoppingCartViewModel.getShoppingCart()
45                val totalAmount = shoppingCartViewModel.getTotalAmount()
46
47                val intent = Intent(Intent.ACTION_SENDTO).apply {
48                    data = Uri.parse("mailto:")
49                    putExtra(Intent.EXTRA_EMAIL, arrayOf(emailSub))
50                    putExtra(Intent.EXTRA_SUBJECT, subject)
51                    putExtra(Intent.EXTRA_TEXT, viewModel.getBodyText(orderDetails, cartItems, totalAmount))
52                }
53                startActivity(intent)
54            } catch (e: Exception) {
55                Log.e("CheckoutOrderFragment", "Error during checkout: ${e.message}")
56            }
57            view?.findNavController()?.navigate(R.id.action_checkoutOrderFragment_to_homepageFragment)
58        }
59        return view
60    }
61 }
```

7.2.2.2 Uitleg code

Het fragment, genaamd `CheckoutOrderFragment`, maakt gebruik van data binding via de `FragmentCheckoutOrderBinding` klasse en heeft een bijbehorende `ViewModel` genaamd `CheckoutOrderViewModel`. Daarnaast wordt er ook gebruik gemaakt van de `ShoppingCartViewModel` om toegang te krijgen tot gegevens over het winkelwagentje.

Het lay-outbestand, dat vermoedelijk is gekoppeld aan dit fragment, bevat waarschijnlijk verschillende `EditText`-velden voor het invoeren van klantgegevens, zoals naam, e-mailadres, adres, postcode, land en telefoonnummer. Daarnaast omvat het lay-outbestand ook een knop met de id `checkoutButton`.

Wanneer de gebruiker op de knop klikt, wordt een reeks klantgegevens verzameld vanuit de invoervelden, en deze gegevens worden vervolgens gebruikt om een `OrderDetails` object te maken. Dit object bevat informatie zoals de naam, e-mailadres, adres, postcode, land en telefoonnummer van de klant.

De winkelwagengegevens, inclusief de lijst met items en het totale bedrag van de bestelling, worden opgehaald via de `ShoppingCartViewModel`. Vervolgens wordt er een intent gemaakt om een e-mailapplicatie te openen met vooraf ingevulde gegevens, zoals het onderwerp, het e-mailadres van de ontvanger, het berichtonderwerp en de inhoud van het bericht. Het bericht bevat de tekst die is gegenereerd door de `getBodyText`-functie van de `CheckoutOrderViewModel`, waarin de details van de bestelling, klantgegevens en de totale kosten zijn opgenomen.

Eventuele fouten tijdens het afrekenproces worden afgevangen en gelogd. Na het succesvol verzenden van de e-mail wordt de gebruiker genavigeerd naar het startscherm van de app (`homepageFragment`).

7.2.3 CheckoutOrderViewModel

7.2.3.1 Code

```
1 class CheckoutOrderViewModel() : ViewModel() {
2     fun getBodyText(orderDetails: OrderDetails, cartItems: List<Wine>, totalAmount: Double): String {
3         val stringBuilder = StringBuilder()
4
5         // Thank you message
6         stringBuilder.append("Bestelling geplaatst door: ${orderDetails.name}!\n\n")
7
8         // Order details
9         stringBuilder.append("Bestel details:\n")
10        stringBuilder.append("Naam: ${orderDetails.name}\n")
11        stringBuilder.append("E-mailadres: ${orderDetails.email}\n")
12        stringBuilder.append("Adres: ${orderDetails.address}\n")
13        stringBuilder.append("Postcode: ${orderDetails.zipCode}\n")
14        stringBuilder.append("Woonplaats: ${orderDetails.country}\n")
15        stringBuilder.append("GSM-nummer: ${orderDetails.phoneNumber}\n\n")
16
17        // Bought wines
18        stringBuilder.append("Bestelde wijnen:\n")
19        for (wine in cartItems) {
20            stringBuilder.append("${wine.Naam} - Hoeveelheid: ${wine.Aantal} - Totaalprijs per wijn: €${wine.TotalPrice}\n")
21        }
22        stringBuilder.append("\n")
23
24        // Total price
25        stringBuilder.append("Totaalprijs: €$totalAmount\n")
26        stringBuilder.append("Gelieve te betalen bij afhaal! (Cash/Payconiq)\n")
27
28        return stringBuilder.toString()
29    }
30 }
```

7.2.3.2 Uitleg code

De ViewModel bevat een functie genaamd `getBodyText`, die wordt gebruikt om een opgemaakte tekst te genereren op basis van de ingevoerde gegevens van een `OrderDetails` object, een lijst van bestelde wijnen (`cartItems`), en het totale bedrag van de bestelling.

In deze functie begint een `StringBuilder` met het samenstellen van een e-mail, waarin de naam van de klant wordt vermeld. Vervolgens worden de details van de bestelling toegevoegd, zoals de naam, het e-mailadres, het adres, de postcode, de woonplaats en het GSM-nummer van de klant.

Daarna wordt een sectie toegevoegd met de bestelde wijnen, waarbij voor elke wijn in de lijst de naam, hoeveelheid en totale prijs per wijn worden vermeld. Deze informatie wordt ontleend aan de `cartItems`-lijst die is doorgegeven aan de functie.

Tot slot bevat de gegenereerde tekst ook de totale prijs van de bestelling, evenals een instructie om bij afhaling te betalen, waarbij zowel contant als via de Payconiq-betaalmethode wordt genoemd.

De functie retourneert de uiteindelijke opgemaakte tekst als een `String`. Deze tekst kan vervolgens worden gebruikt in de e-mail die wordt verzonden bij het afronden van de bestelling, zoals geïmplementeerd in het `CheckoutOrderFragment`.

7.2.4 Fragment_checkout_order.xml

7.2.4.1 Code

```
1  <layout xmlns:android="http://schemas.android.com/apk/res/android"
2        xmlns:app="http://schemas.android.com/apk/res-auto"
3        xmlns:tools="http://schemas.android.com/tools">
4
5      <data>
6        <variable
7          name="orderDetailsViewModel"
8          type="com.example.appwijnhuisronny.CheckoutOrderViewModel" />
9      </data>
10
11     <LinearLayout
12       android:layout_width="match_parent"
13       android:layout_height="match_parent"
14       android:orientation="vertical"
15       android:padding="8dp">
16
17       <com.google.android.material.textfield.TextInputLayout
18         android:id="@+id/nameInputLayout"
19         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
20         android:layout_width="match_parent"
21         android:layout_height="wrap_content"
22         android:layout_marginTop="16dp">
23
24         <com.google.android.material.textfield.TextInputEditText
25           android:id="@+id/nameEditText"
26           android:layout_width="match_parent"
27           android:layout_height="wrap_content"
28           android:hint="@string/voornaamEnAchternaam" />
29       </com.google.android.material.textfield.TextInputLayout>
30
31       <com.google.android.material.textfield.TextInputLayout
32         android:id="@+id/mailInputLayout"
33         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
34         android:layout_width="match_parent"
35         android:layout_height="wrap_content"
36         android:layout_marginTop="16dp">
37
38         <com.google.android.material.textfield.TextInputEditText
39           android:id="@+id/mailEditText"
40           android:layout_width="match_parent"
41           android:layout_height="wrap_content"
42           android:hint="@string/email" />
43       </com.google.android.material.textfield.TextInputLayout>
44
```

```

45     <com.google.android.material.textfield.TextInputLayout
46         android:id="@+id/addressInputLayout"
47         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
48         android:layout_width="match_parent"
49         android:layout_height="wrap_content"
50         android:layout_marginTop="16dp">
51
52         <com.google.android.material.textfield.TextInputEditText
53             android:id="@+id/addressEditText"
54             android:layout_width="match_parent"
55             android:layout_height="wrap_content"
56             android:hint="@string/straatEnNummer" />
57     </com.google.android.material.textfield.TextInputLayout>
58
59     <com.google.android.material.textfield.TextInputLayout
60         android:id="@+id/zipCodeInputLayout"
61         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
62         android:layout_width="match_parent"
63         android:layout_height="wrap_content"
64         android:layout_marginTop="16dp">
65
66         <com.google.android.material.textfield.TextInputEditText
67             android:id="@+id/zipCodeEditText"
68             android:layout_width="match_parent"
69             android:layout_height="wrap_content"
70             android:hint="@string/postcode" />
71     </com.google.android.material.textfield.TextInputLayout>
72
73     <com.google.android.material.textfield.TextInputLayout
74         android:id="@+id/countryInputLayout"
75         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
76         android:layout_width="match_parent"
77         android:layout_height="wrap_content"
78         android:layout_marginTop="16dp">
79
80         <com.google.android.material.textfield.TextInputEditText
81             android:id="@+id/countryEditText"
82             android:layout_width="match_parent"
83             android:layout_height="wrap_content"
84             android:hint="@string/woonplaats" />
85     </com.google.android.material.textfield.TextInputLayout>
86
87     <com.google.android.material.textfield.TextInputLayout
88         android:id="@+id/phoneInputLayout"
89         style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
90         android:layout_width="match_parent"
91         android:layout_height="wrap_content"
92         android:layout_marginTop="16dp">
93
94         <com.google.android.material.textfield.TextInputEditText
95             android:id="@+id/phoneEditText"
96             android:layout_width="match_parent"
97             android:layout_height="wrap_content"
98             android:hint="@string/gsmNummer" />
99     </com.google.android.material.textfield.TextInputLayout>
100
101     <Button
102         android:id="@+id/checkoutButton"
103         android:layout_width="wrap_content"
104         android:layout_height="wrap_content"
105         android:layout_gravity="center"
106         android:layout_marginTop="12dp"
107         android:text="@string/bestellingAfronden" />
108
109 </LinearLayout>
110 </layout>

```

7.2.4.2 Uitleg code

De bovenstaande XML-code vertegenwoordigt een lay-outbestand voor een scherm in een Android-applicatie, waarschijnlijk bedoeld voor het afrekenen van een bestelling. In dit lay-outbestand worden verschillende visuele elementen gedefinieerd, zoals tekstinvoervelden en een knop, die worden gestyled met Material Design-componenten.

Het `LinearLayout`-element wordt gebruikt als de hoofdcontainer, en het bevat verschillende `TextInputLayout`-elementen, elk gevolgd door een `TextInputEditText`. Deze elementen vormen invoervelden voor de naam, het e-mailadres, het adres, de postcode, de woonplaats en het GSM-nummer van de klant. Elke `TextInputLayout` wordt gestyled met Material Design en bevat een hint om de gebruiker te begeleiden bij het invoeren van de juiste informatie.

Tot slot is er een knop (`Button`) met de ID `checkoutButton`, waarschijnlijk bedoeld om een e-mailintent te starten wanneer erop wordt geklikt. De knop is gecentreerd in het scherm en heeft de tekst "Bestelling Afronden".

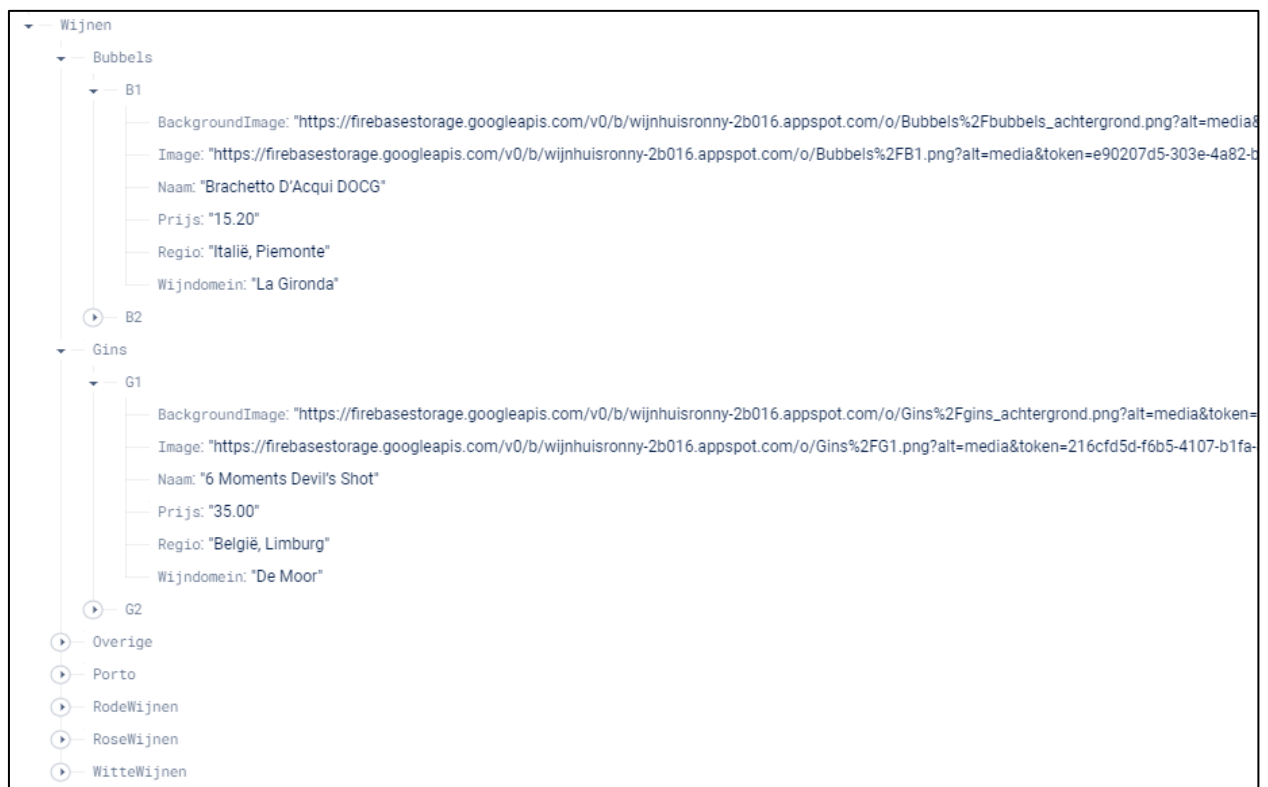
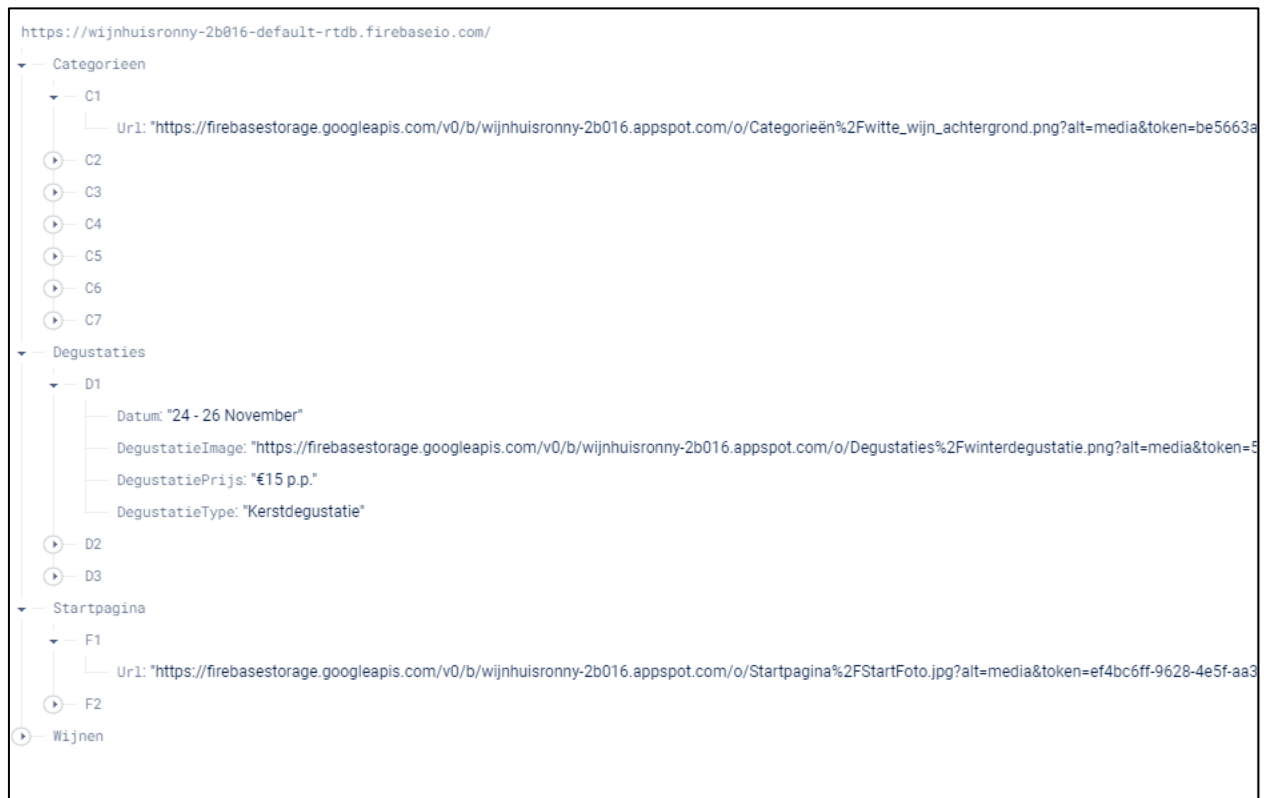
7.3 E-mailintent

7.3.1 Visuele weergave



8. Database en Storage

8.1 Indeling Firebase Realtime Database



8.2 Indeling Firebase Storage

gs://wijnhuisronny-2b016.appspot.com		Upload file		
Size	Type	Last modified		
Size	Type	Last modified	Folder path	Name
—	Folder	—	Bubbels/	
—	Folder	—	Categorieën/	
—	Folder	—	Degustaties/	
—	Folder	—	Gins/	
—	Folder	—	Overige/	
—	Folder	—	Porto/	
—	Folder	—	RodeWijnen/	
—	Folder	—	RoseWijnen/	
—	Folder	—	Startpagina/	
—	Folder	—	WitteWijnen/	

gs://wijnhuisronny-2b016.appspot.com > RodeWijnen		Upload file		
Size	Type	Last modified		
Size	Type	Last modified	Name	
114.37 KB	image/png	Dec 28, 2023	RW1.png	
72.86 KB	image/png	Dec 28, 2023	RW2.png	
78.71 KB	image/png	Dec 28, 2023	RW3.png	
72.08 KB	image/png	Dec 28, 2023	RW4.png	
264.03 KB	image/png	Dec 28, 2023	RW5.png	
2.78 KB	image/png	Dec 28, 2023	wijn_achtergronden_kleur_rood.png	