|  |
| --- |
| 2023-2024 1ste semester |
| Mobiele Apps 2: verslag project |
| Wijnhuis Ronny |

|  |
| --- |
| Calvin Liaci  3ICT |

Inhoudsopgave

[Verslag project: Wijnhuis Ronny 4](#_Toc154761160)

[1. Korte toelichting app vanuit het oogpunt van de gebruiker 4](#_Toc154761161)

[2. Icon en Splash Screen 5](#_Toc154761162)

[2.1 Visuele weergave en code 5](#_Toc154761163)

[2.1.1 Icon 5](#_Toc154761164)

[2.1.2 Splash screen 6](#_Toc154761165)

[2.2 Uitleg 6](#_Toc154761166)

[3. Navigatie 6](#_Toc154761167)

[3.1 Visuele weergave en code 6](#_Toc154761168)

[3.2 Uitleg 7](#_Toc154761169)

[4. Startpagina 7](#_Toc154761170)

[4.1 Visuele weergave 7](#_Toc154761171)

[4.2 HomePagePhotosRepository 8](#_Toc154761172)

[4.2.1 Code 8](#_Toc154761173)

[4.2.2 Uitleg code 8](#_Toc154761174)

[4.3 HomepageFragment 9](#_Toc154761175)

[4.3.1 Code 9](#_Toc154761176)

[4.3.2 Uitleg code 9](#_Toc154761177)

[4.4 HomepageFragmentViewModel 10](#_Toc154761178)

[4.4.1 Code 10](#_Toc154761179)

[4.4.2 Uitleg code 10](#_Toc154761180)

[4.5 Fragment\_homepage.xml 11](#_Toc154761181)

[4.5.1 Code 11](#_Toc154761182)

[4.5.2 Uitleg code 12](#_Toc154761183)

[5. Wijnen pagina 13](#_Toc154761184)

[5.1 Wijncategorieën pagina 13](#_Toc154761185)

[5.1.1 Visuele weergave 13](#_Toc154761186)

[5.1.2 WinesCategoryPhotosRepository 14](#_Toc154761187)

[5.1.2.1 Code 14](#_Toc154761188)

[5.1.2.2 Uitleg code 14](#_Toc154761189)

[5.1.3 WinesCategoryFragment 15](#_Toc154761190)

[5.1.3.1 Code 15](#_Toc154761191)

[5.1.3.2 Uitleg code 16](#_Toc154761192)

[5.1.4 WinesCategoryViewModel 16](#_Toc154761193)

[5.1.4.1 Code 16](#_Toc154761194)

[5.1.4.2 Uitleg code 16](#_Toc154761195)

[5.1.5 Fragment\_wines\_category.xml 17](#_Toc154761196)

[5.1.5.1 Code 17](#_Toc154761197)

[5.1.5.2 Uitleg code 20](#_Toc154761198)

[5.2 Wijnen pagina 21](#_Toc154761199)

[5.2.1 Visuele weergave 21](#_Toc154761200)

[5.2.2 Wine Model 22](#_Toc154761201)

[5.2.2.1 Code 22](#_Toc154761202)

[5.2.2.2 Uitleg code 22](#_Toc154761203)

[5.2.3 WinesRepository 23](#_Toc154761204)

[5.2.3.1 Code 23](#_Toc154761205)

[5.2.3.2 Uitleg code 23](#_Toc154761206)

[5.2.4 WineAdapter 24](#_Toc154761207)

[5.2.4.1 Code 24](#_Toc154761208)

[5.2.4.2 Uitleg code 24](#_Toc154761209)

[5.2.5 WinesFragment 25](#_Toc154761210)

[5.2.5.1 Code 25](#_Toc154761211)

[5.2.5.2 Uitleg code 26](#_Toc154761212)

[5.2.6 WinesFragmentViewModel 26](#_Toc154761213)

[5.2.6.1 Code 26](#_Toc154761214)

[5.2.6.2 Uitleg code 27](#_Toc154761215)

[5.2.7 Fragment\_wines.xml 27](#_Toc154761216)

[5.2.7.1 Code 27](#_Toc154761217)

[5.2.7.2 Uitleg code 27](#_Toc154761218)

[5.2.8 Wines\_item.xml 28](#_Toc154761219)

[5.2.8.1 Code 28](#_Toc154761220)

[5.2.8.2 Uitleg code 29](#_Toc154761221)

[6. Degustatie pagina 30](#_Toc154761222)

[6.1 Visuele weergave 30](#_Toc154761223)

[6.2 WineTasting Model 30](#_Toc154761224)

[6.2.1 Code 30](#_Toc154761225)

[6.2.2 Uitleg code 30](#_Toc154761226)

[6.3 WineTastingsRepository 31](#_Toc154761227)

[6.3.1 Code 31](#_Toc154761228)

[6.3.2 Uitleg code 31](#_Toc154761229)

[6.4 WineTastingsAdapter 32](#_Toc154761230)

[6.4.1 Code 32](#_Toc154761231)

[6.4.2 Uitleg code 32](#_Toc154761232)

[6.5 WineTastingsFragment 33](#_Toc154761233)

[6.5.1 Code 33](#_Toc154761234)

[6.5.2 Uitleg code 34](#_Toc154761235)

[6.6 WineTastingsViewModel 34](#_Toc154761236)

[6.6.1 Code 34](#_Toc154761237)

[6.6.2 Uitleg code 35](#_Toc154761238)

[6.7 Fragment\_wine\_tastings.xml 35](#_Toc154761239)

[6.7.1 Code 35](#_Toc154761240)

[6.7.2 Uitleg code 35](#_Toc154761241)

[6.8 Wine\_tasting\_item.xml 36](#_Toc154761242)

[6.8.1 Code 36](#_Toc154761243)

[6.8.2 Uitleg code 37](#_Toc154761244)

[6.9 E-mailintent 38](#_Toc154761245)

[6.9.1 Visuele weergave 38](#_Toc154761246)

[7. Winkelwagen 39](#_Toc154761247)

[7.1 Winkelwagen pagina 39](#_Toc154761248)

[7.1.1 Visuele weergave 39](#_Toc154761249)

[7.1.2 ShoppingCartAdapter 40](#_Toc154761250)

[7.1.2.1 Code 40](#_Toc154761251)

[7.1.2.2 Uitleg code 40](#_Toc154761252)

[7.1.3 ShoppingCartFragment 41](#_Toc154761253)

[7.1.3.1 Code 41](#_Toc154761254)

[7.1.3.2 Uitleg code 42](#_Toc154761255)

[7.1.4 ShoppingCartViewModel 42](#_Toc154761256)

[7.1.4.1 Code 42](#_Toc154761257)

[7.1.4.2 Uitleg code 43](#_Toc154761258)

[7.1.5 Fragment\_shopping\_cart.xml 44](#_Toc154761259)

[7.1.5.1 Code 44](#_Toc154761260)

[7.1.5.2 Uitleg code 45](#_Toc154761261)

[7.1.6 Shoppingcart\_row.xml 46](#_Toc154761262)

[7.1.6.1 Code 46](#_Toc154761263)

[7.1.6.2 Uitleg code 47](#_Toc154761264)

# Verslag project: Wijnhuis Ronny

## 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.

## Icon en Splash Screen

### Visuele weergave en code

#### Icon

Afbeelding met tekst, schermopname, Lettertype, logo

Automatisch gegenereerde beschrijvingAfbeelding met elektronica, tekst, schermopname, multimedia

Automatisch gegenereerde beschrijving

#### Splash screen

Afbeelding met tekst, schermopname, Lettertype, ontwerp

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, visitekaartje, Lettertype

Automatisch gegenereerde beschrijvingAfbeelding met stoel, tekening, schets, ontwerp

Automatisch gegenereerde beschrijving

### Uitleg

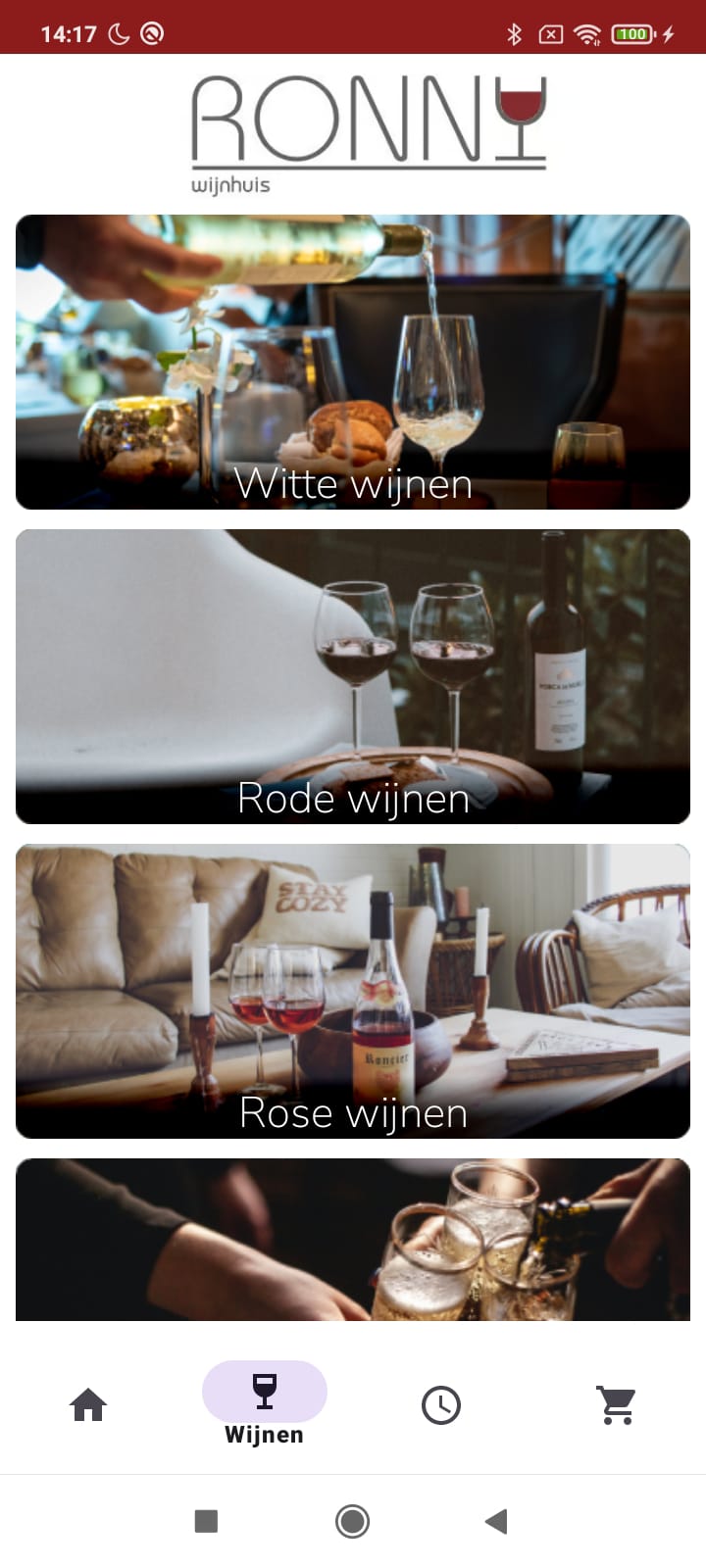
Om de Icon en Splash screen aan te passen in Android Studio, moest ik een paar wijzigingen aanbrengen in het AndroidManifest.xml-bestand. Voor het pictogram moest ik zelf een afbeelding toevoegen via `android:icon`. Wat betreft het opstartscherm, moest ik eerst een stijl maken in het styles.xml-bestand, 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`.

## Afbeelding met tekst, schermopname Automatisch gegenereerde beschrijvingNavigatie

### Visuele weergave en code

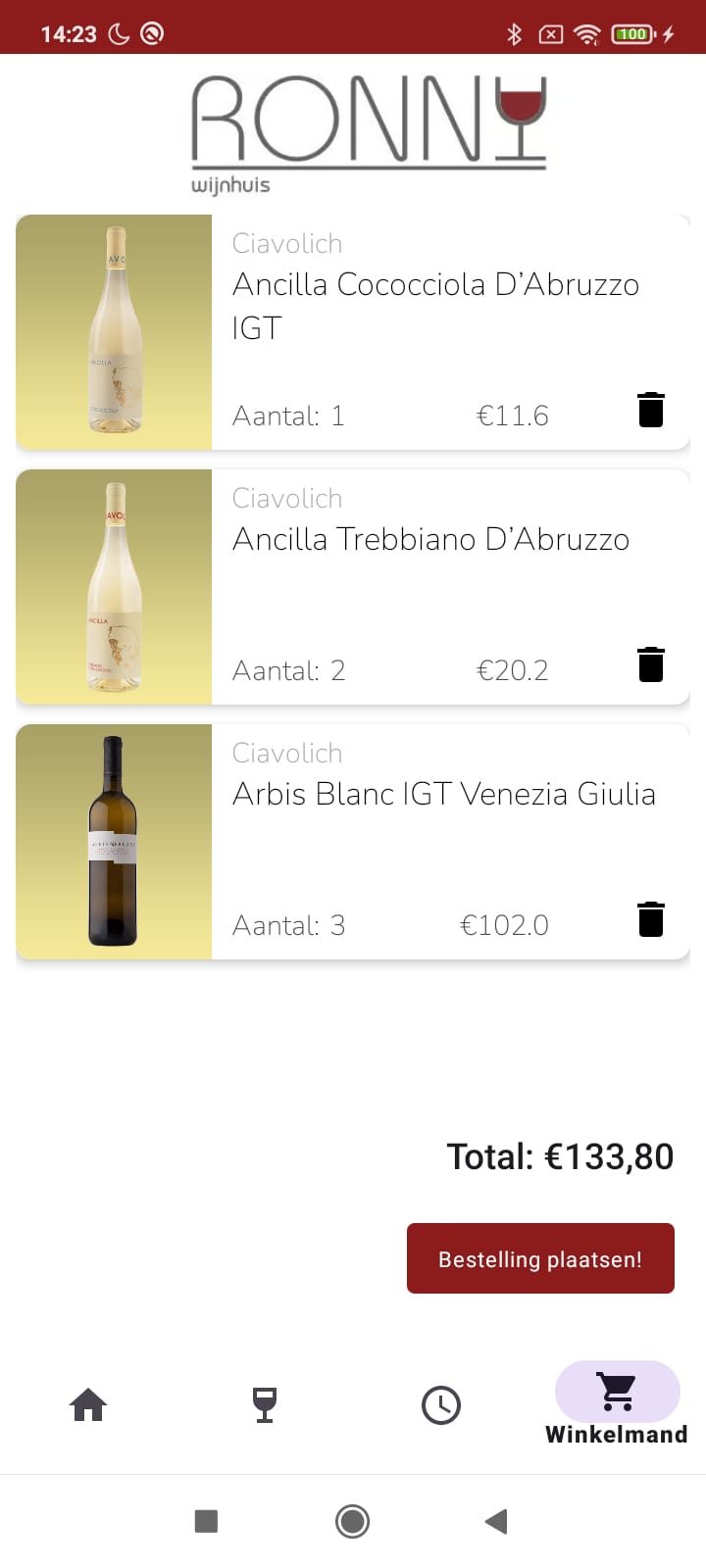
Afbeelding met tekst, schermopname, Menselijk gezicht, persoon

Automatisch gegenereerde beschrijving



Afbeelding met tekst, schermopname, fles

Automatisch gegenereerde beschrijving

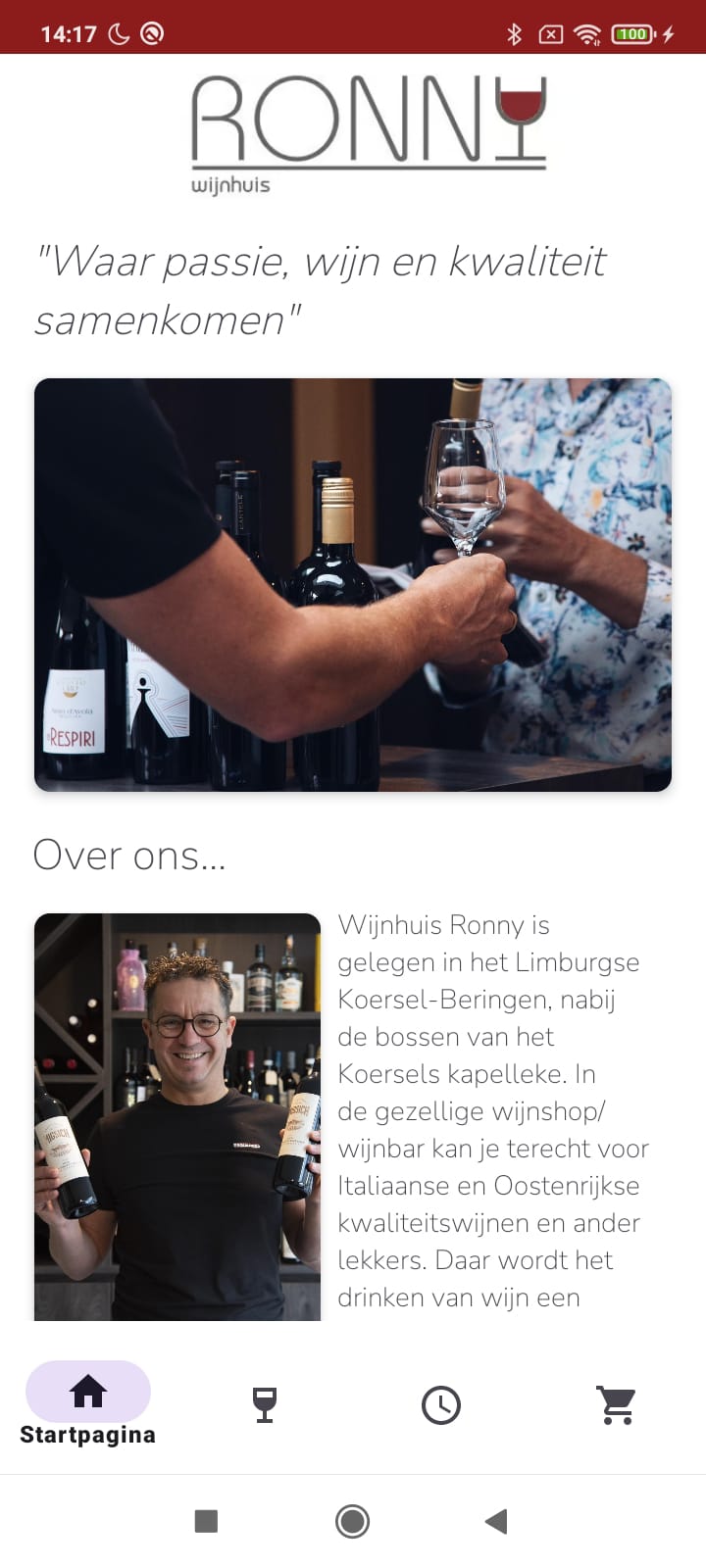


### Uitleg

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.

## Startpagina

### Visuele weergave



### HomePagePhotosRepository

#### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

#### 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 geïnitialiseerd.

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.

### HomepageFragment

#### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

#### 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 geïnitialiseerd, 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.

### HomepageFragmentViewModel

#### Code

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijving

#### 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.

### Fragment\_homepage.xml

#### Code

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, menu, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

#### 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 `<data>`-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 `ImageViews`. 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.

## Wijnen pagina

### Wijncategorieën pagina

#### Visuele weergave

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, voedsel

Automatisch gegenereerde beschrijving

#### WinesCategoryPhotosRepository

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

##### 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 "Categorieen"-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 geïnitialiseerd.

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.

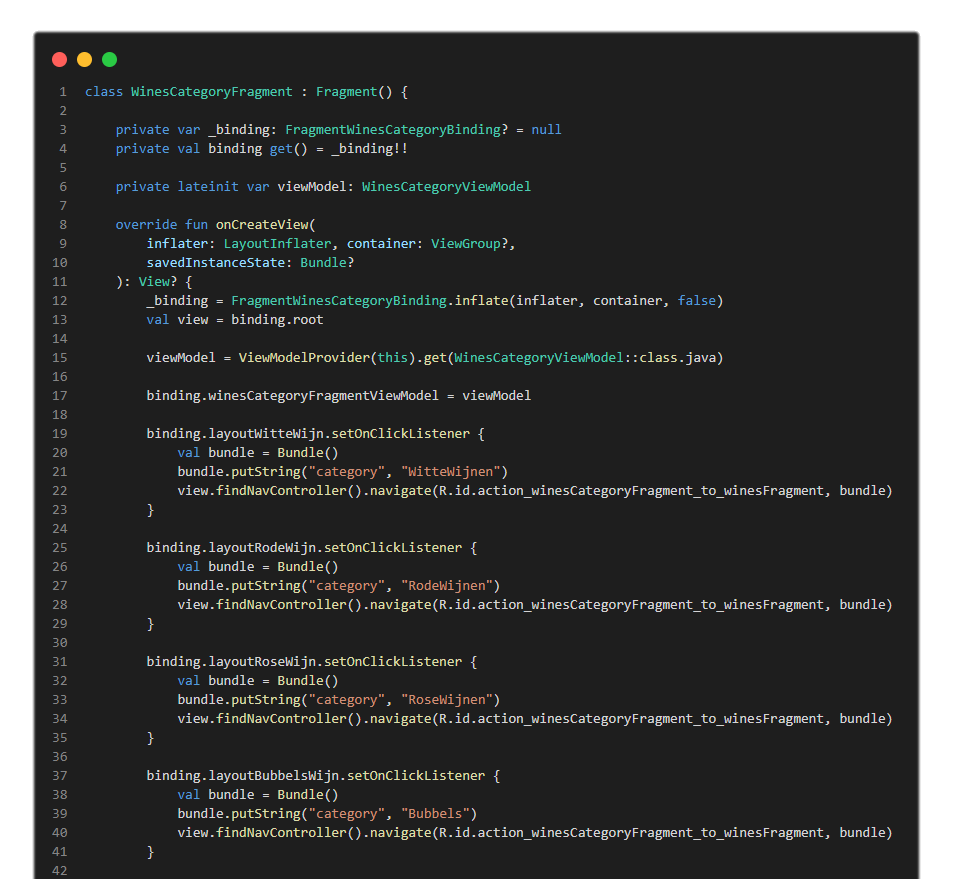
De `getPhotoUrls`-methode wordt gebruikt om foto-URL's op te halen vanuit de database. Hiervoor wordt een `ValueEventListener` aan de database-referentie toegevoegd om eenmalig gegevens op te halen. In het geval van een succesvolle gegevenswijziging worden de foto-URL's uit de database gehaald en doorgegeven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

#### WinesCategoryFragment

##### Code



Afbeelding met tekst, elektronica, schermopname, scherm

Automatisch gegenereerde beschrijving

##### 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 geïnitialiseerd 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.

#### WinesCategoryViewModel

##### Code

Afbeelding met tekst, schermopname, software, multimedia

Automatisch gegenereerde beschrijving

##### 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` geïnitialiseerd, 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.

#### Fragment\_wines\_category.xml

##### Code

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, menu, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, menu, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, menu, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

##### 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.

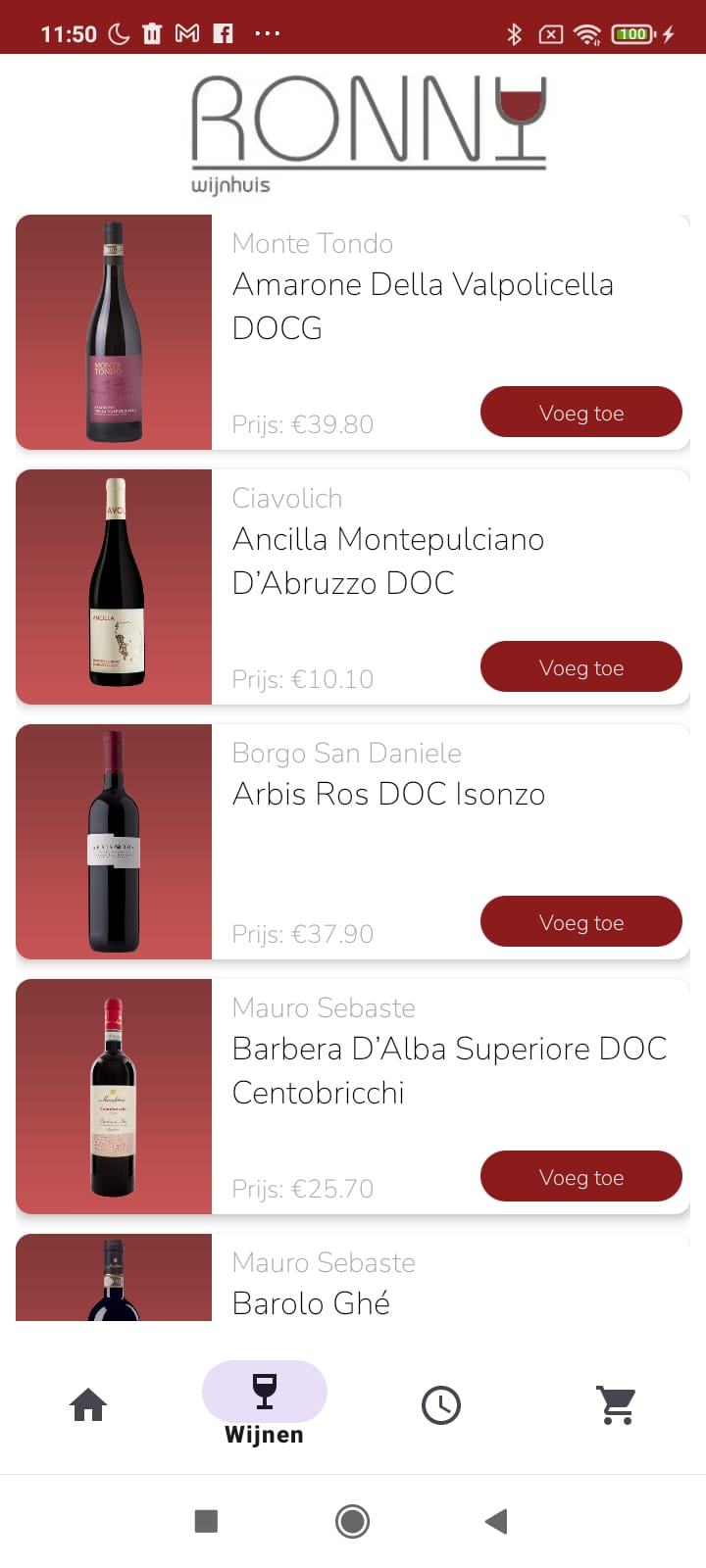
Elk `ConstraintLayout`-element bestaat uit een achtergrondafbeelding (`ImageView`) en een tekstlabel (`TextView`). De afbeeldingen worden gespecificeerd met unieke identificatoren zoals "image\_witte\_wijn," "image\_rode\_wijn," enzovoort, elk met een bijbehorende achtergrondafbeelding en tekstlabel 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 achtergrondafbeelding 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.

### Wijnen pagina

#### Visuele weergave

Afbeelding met tekst, fles, schermopname, ontwerp

Automatisch gegenereerde beschrijving

#### Wine Model

##### Code

Afbeelding met tekst, schermopname, scherm, multimedia

Automatisch gegenereerde beschrijving

##### 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 achtergrondafbeelding 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.

#### WinesRepository

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

##### 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 geïnitialiseerd 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 geïnitialiseerd.

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`.

#### WineAdapter

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

##### 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".

#### WinesFragment

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

##### 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.

#### WinesFragmentViewModel

##### Code

Afbeelding met tekst, schermopname, software, multimedia

Automatisch gegenereerde beschrijving

##### 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 geïnitialiseerd als een lege lijst. Het geheel wordt afgerond met een logbericht dat de succesvolle initialisatie van het `ViewModel` met de opgegeven categorie aangeeft.

#### Fragment\_wines.xml

##### Code

Afbeelding met tekst, schermopname, software

Automatisch gegenereerde beschrijving

##### 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.

#### Wines\_item.xml

##### Code

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

##### 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 wijnafbeeldingen worden weergegeven.

Een `FrameLayout` fungeert als container voor deze afbeeldingen, waarbij de achtergrondafbeelding een achtergrond heeft en de wijnafbeelding hier bovenop wordt geplaatst. Verschillende `TextViews` 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.

## Degustatie pagina

### Visuele weergave

Afbeelding met tekst, schermopname, Website, Onlineadvertenties

Automatisch gegenereerde beschrijvingAfbeelding met tekst, schermopname, fles

Automatisch gegenereerde beschrijving

### WineTasting Model

#### Code

Afbeelding met tekst, schermopname, scherm, multimedia

Automatisch gegenereerde beschrijving

#### 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.

### WineTastingsRepository

#### Code

Afbeelding met tekst, elektronica, computer, schermopname

Automatisch gegenereerde beschrijving

#### 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 geretourneerd.

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.

### WineTastingsAdapter

#### Code

Afbeelding met tekst, elektronica, schermopname, software

Automatisch gegenereerde beschrijving

#### 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 geïnflateerd 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 geïnitialiseerd in de `onCreateViewHolder`-methode.

### WineTastingsFragment

#### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu, Lettertype

Automatisch gegenereerde beschrijving

#### 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 geïnitialiseerd 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.

### WineTastingsViewModel

#### Code

Afbeelding met tekst, schermopname, computer, software

Automatisch gegenereerde beschrijving

#### 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.

### Fragment\_wine\_tastings.xml

#### Code

Afbeelding met tekst, schermopname, computer, software

Automatisch gegenereerde beschrijving

#### 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.

### Wine\_tasting\_item.xml

#### Code

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

#### 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".

### E-mailintent

#### Visuele weergave



## Winkelwagen

### Winkelwagen pagina

#### Visuele weergave

Afbeelding met tekst, schermopname, ontwerp

Automatisch gegenereerde beschrijving

#### ShoppingCartAdapter

##### Code

Afbeelding met tekst, schermopname, computer, software

Automatisch gegenereerde beschrijving

##### 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.

#### ShoppingCartFragment

##### Code

Afbeelding met tekst, schermopname, computer, software

Automatisch gegenereerde beschrijving

##### 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 geïnitialiseerd 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 geïnitialiseerd 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.

#### ShoppingCartViewModel

##### Code

Afbeelding met tekst, schermopname, computer, software

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, multimedia, software

Automatisch gegenereerde beschrijving

##### 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.

#### Fragment\_shopping\_cart.xml

##### Code

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

##### 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.

#### Shoppingcart\_row.xml

##### Code

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

##### 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 wijnitem, 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.

### Checkout pagina

#### CheckoutOrderFragment

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software

Automatisch gegenereerde beschrijving

##### 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`).

#### CheckoutOrderViewModel

##### Code

Afbeelding met tekst, elektronica, schermopname, computer

Automatisch gegenereerde beschrijving

##### 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`.

#### Fragment\_checkout\_order.xml

##### Code

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

Afbeelding met tekst, menu, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, menu

Automatisch gegenereerde beschrijving

##### 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".

## Database en Storage

### Indeling Firebase Realtime Database

Afbeelding met tekst, schermopname, document

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, document

Automatisch gegenereerde beschrijving

### Indeling Firebase Storage

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

Afbeelding met tekst, nummer, Lettertype, lijn

Automatisch gegenereerde beschrijving