Mobilna aplikacija za potporu prostornom računarstvu korištenjem uređaja Apple Vision Pro

Lešić, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Računarstvo

Mobilna aplikacija za potporu prostornom računarstvu korištenjem uređaja Apple Vision Pro

Diplomski rad

Luka Lešić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Luka Lešić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1298R, 07.10.2022.
JMBAG:	0165082003
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Mobilna aplikacija za potporu prostornom računarstvu korištenjem uređaja Apple Vision Pro
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati probleme i izazove prostornog računarstva s posebnim osvrtom na uređaj Apple Vision Pro i njegove mogućnosti korištenja za potporu u izboru i korištenju proizvoda svakodnevne upotrebe. Na temelju analize stanja u području i pregleda postojećih sličnih rješenja, treba definirati funkcionalne i nefunkcionalne zahtjeve na mobilnu aplikaciju, predložiti model i arhitekturu aplikacije, razvojnu okolinu, potrebne biblioteke i postupak korištenja uređaja Apple Vision Pro. Mobilna aplikacija zajedno s navedenim uređajem

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	17.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	04.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	10.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 10.10.2024.

Ime i prezime Pristupnika:	Luka Lešić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1298R, 07.10.2022.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za potporu prostornom računarstvu korištenjem uređaja Apple Vision Pro**

izrađen pod vodstvom mentora prof. dr. sc. Goran Martinović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Tijekom proteklih desetljeća područje proširene i virtualne stvarnosti razvija se vrlo brzo, što dovodi do potrebe za stvaranjem nove kategorije nosivih uređaja koji podržavaju prikazivanje novih tipova sučelja. Takvi uređaji postaju sve dostupniji i cjenovno prihvatljiviji velikom broju korisnika, što dovodi i do bržeg razvoja programske podrške i aplikacija kao i do novih inovacija i većih primjena u svakodnevnici. Iz navedenih razloga, pojavljuju se i razni izazovi. Trenutno ne postoji velik broj obrazovnih materijala za programere, dok postojeći programski jezici i razvojni okviri prolaze kroz brze promjene, dovodeći do osjećaja nesigurnosti oko odabira prave tehnologije za razvoj programa.

Cilj ovog rada je istražiti trenutno stanje tržišta potrošačke elektronike s podrškom za prikazivanje virtualne i proširene stvarnosti, te samostalno izraditi aplikaciju koja se temelji na navedenim načelima, omogućavajući korisniku složene interakcije s trodimenzionalnim objektima korištenjem uređaja Apple Vision Pro. Osim toga, potrebno je analizirati razvojni okvir SwiftUI i njegovu podršku za rad s mješovitom stvarnosti. Tako se u radu detaljno istražuju kompatibilnost Appleovih razvojnih okvira, kao što su RealityKit i ARKit, s postojećim razvojnim alatima, različite značajke operacijskog sustava visionOS te ponašanje i stabilnost izrađene mobilne aplikacije unutar upravljačkog okruženja visionOS. Aplikacija je temeljito testirana, te je izrađena s ciljem postizanja minimalne potrošnje već ograničenih resursa uređaja Vision Pro. U radu se testira i ponašanje trodimenzionalnih modela i maketa u okruženju visionOS i navode se prednosti i nedostatci (eng. *Universal Scene Description Zip*) tipa podatka za prikazivanje modela. Promatra se utjecaj učitavanja velikog broja maketa na potrošnju memorije te performanse virtualnog simulatora. Osim toga, primjenjuju se složena matematička rješenja za manipulaciju modelima, kako bi se postigle značajke skaliranja, pomicanja i rotacije modela.

U drugom poglavlju provodi se istraživanje tržišta, detaljan pregled načela virtualne i proširene stvarnosti, kao i pregled trenutnih prepreka koje uređaji s podrškom za mješovitu stvarnost moraju savladati. U trećem poglavlju detaljno se opisuje sklopovlje i programska podrška koji pogone uređaj Apple Vision Pro. Četvrto poglavlje opisuje građu aplikacije, programski jezik Swift te potrebne biblioteke i okruženje potrebno za izradu aplikacije. Peto poglavlje prikazuje konkretnu implementaciju potrebnih tehnologija kroz pregled programskog koda aplikacije. U šestom poglavlju prikazuju se korisničko sučelje i upute za korištenje aplikacije, te se uspješno provodi testiranje aplikacije pomoću jediničnih testova programskog koda.

2. IZAZOVI I STANJE U PODRUČJU MJEŠOVITE STVARNOSTI

Poglavlje 2 pruža opis pojmova virtualne, proširene i mješovite stvarnosti i uvid u trenutno stanje na tržištu uređaja koji podržavaju ove tehnologije prikaza. Razmatraju se izazovi s kojima se proizvođači i programeri susreću u razvoju sklopovlja i programske podrške, te se opisuju primjenjivani algoritmi za prikaz sadržaja na tim uređajima.

2.1. Usporedba virtualne i proširene stvarnosti

2.1.1. Virtualna stvarnost

Prema literaturi [1], pojam virtualne stvarnosti podrazumijeva uporabu računalne tehnologije za stvaranje dojma višedimenzionalnog okruženja, uključujući prikazivanje objekata prirodne veličine s fiksnom lokacijom neovisnom o korisnikovoj poziciji. Korisnik biva okružen simuliranim okruženjem i može se slobodno kretati i međudjelovati s objektima u prostoru. Sklopovlje uređaja virtualne stvarnosti može biti povezano žicom ili bežično, te sadrži zaslone koji se nose na glavi (*eng*. *Head-Mounted Displays*) uz mogućnost praćenja pokreta glave [2]. Iako su dostupni i neovisni uređaji posebne namjene koji postižu ovaj učinak, prikazivanje virtualne stvarnosti moguće je postići i uz pametni telefon ubačen u ergonomsko kućište namijenjeno za nošenje na glavi s odgovarajućim programskim okruženjem. Stoga, dostupnost virtualne stvarnosti postaje sve veća, pa tako proteklih godina naglo raste i njezina popularnost. Trenutno najveće primjene tehnologije virtualne stvarnosti uključuju prikazivanje panoramskih fotografija i videozapisa, igranje videoigara kao i tzv. „virtualne šetnje“, gdje korisnik može iskusiti simulaciju popularnih stvarnih okruženja i na taj način obogatiti svoje znanje i zabaviti se. Slika 2.1 prikazuje naočale za pristup virtualnoj stvarnosti Samsung Galaxy Gear VR.

Slika 2.1. Primjer uređaja za pristup virtualnoj stvarnosti [3]

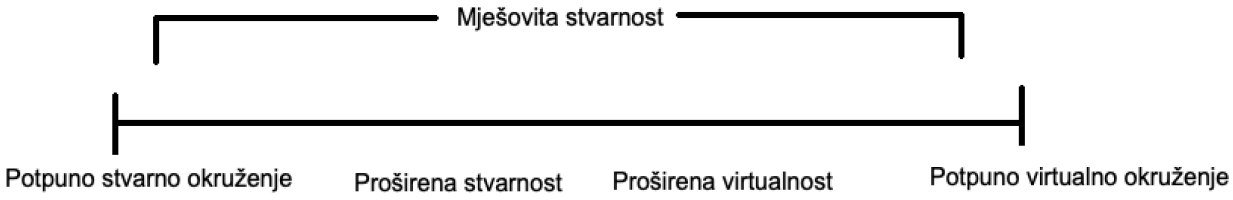
2.1.2. Proširena stvarnost

Proširena stvarnost definira se kao okruženje stvarnog svijeta poboljšano dodavanjem virtualnih računalnih informacija u njega. Proširena stvarnost u stvarnom vremenu korisniku donosi informacije vezane uz njegovo trenutno okruženje ili neke druge vrste sadržaja, poput prikaza fotografija ili videozapisa. Isto tako, aplikacije temeljene na proširenoj stvarnosti mogu pokrivati stvarne objekte virtualnim prikazom drugih objekata, dajući dojam korisniku kako se stvarni objekt ne nalazi u trenutnom okruženju [4]. Slično kao i kod prikaza virtualne stvarnosti, na tržištu su dostupni uređaji namijenjeni prikazu proširene stvarnosti, no sličan učinak se može postići i uz pametne telefone. Zbog velike dostupnosti aplikacija i iskustava temeljenih na ovoj tehnologiji, proširena stvarnost je vrlo popularna među korisnicima te nudi velik broj primjena. Tako se ova tehnologija može koristiti za prikazivanje smjera pri navigaciji korištenjem pametnog telefona, prikazivanje raznih modela u učionicama u obrazovne svrhe ili, primjerice, prikazivanje odjeće ili namještaja u stvarnom prostoru kao pomoć pri kupovini. Dojam proširene stvarnosti može biti postignut snimanjem korisnikovog okruženja, te dodavanjem informacija preko tog video prijenosa u stvarnom vremenu ili postavljanjem informacija preko prozirnih zaslona, kao što to rade razne pametne naočale. Na slici 2.2 je vidljiv prikaz proširene stvarnosti na uređaju Apple iPhone, gdje se preko snimke stvarnog okruženja prikazuje virtualni objekt koji zadržava stalnu veličinu i integrira se s okruženjem.

Slika 2.2. Prikaz proširene stvarnosti na uređaju Apple iPhone

2.1.3. Mješovita stvarnost

Iako postoji dosta različitih poimanja i definicija mješovite stvarnosti, ona se uglavnom definira kao tehnologija koja spaja stvarni svijet s virtualnim, omogućavajući stvarnim i virtualnim objektima postojanje i međudjelovanje u istom okruženju. Prema [5], Paul Milgram definirao je kontinuum virtualnosti i stvarnosti (*eng*. *Reality-Virtuality Continuum*) kojim nastoji uspostaviti skalu između potpune fizičke stvarnosti na jednom kraju i virtualne stvarnosti na drugom kraju kontinuuma, prikazan na slici 2.3. Sve razine stvarnosti između ove dvije krajnosti nazivaju se mješovitom stvarnošću. Prema ovoj definiciji, virtualna stvarnost nije dio mješovite stvarnosti, dok je proširena stvarnost podskup mješovite stvarnosti. Tako je područje primjena mješovite stvarnosti vrlo slično kao i područje primjena proširene stvarnosti. Osim vizualnih, mješovita stvarnost može se odnositi i na simulaciju iskustva zvučnih doživljaja, iskustva mirisa, dodira ili okusa [6]. Mješovita stvarnost predstavlja unosno područje raznih primjena, pa se polako počinje javljati prebačaj u industriji prema podržavanju iste.



Slika 2.3. Prikaz Milgramovog kontinuuma virtualnosti i stvarnosti

2.2. Stanje u području mješovite, virtualne i proširene stvarnosti

2.2.1. Korišteno sklopovlje, algoritmi i programska podrška

Aktualni uređaji s podrškom za prikaz virtualne, mješovite i proširene stvarnosti koriste razne proizvoljne algoritme i programske implementacije za kvalitetan prikaz sadržaja. Ipak, neki su temeljni algoritmi poznati i prihvaćeni u industriji, te su često implementirani u ovakvim uređajima. Primjerice, SLAM (eng. *Simultaneous Localization and Mapping*), također često primjenjivan u području robotike, koristi se za mapiranje virtualnog okruženja i precizno određivanje položaja njegova promatrača. Prema radu [7], vizualne varijante algoritma SLAM koriste kamere uređaja kako bi prikupile podatke o okruženju, te se grupiraju u tri glavne vrste algoritma: čisti vizualni SLAM, vizualno-inercijski SLAM te RGBD-SLAM. Ova tri tipa algoritma razlikuju se po svojstvima korištenja podataka senzora za mjerenje dubine i korištenju podataka inercijskih mjernih jedinica. Čisti vizualni SLAM koristi isključivo podatke dobivene iz kamere u svrhe mapiranja, stoga je najmanje pouzdan. Snimljeni podaci

se obrađuju tako što se uklanja šum iz slike, izvršava detekcija rubova i kutova snimljenih elemenata, te se za detektirane elemente stvaraju tzv. deskriptori koji služe za usporedbu objekata između različitih slika i videa. Na temelju tih deskriptora algoritam određuje kretanje kamere uzimajući u obzir rotaciju i translaciju rubova objekata [8].

Osim navedenog, primjenjuju se i razni algoritmi za praćenje pokreta dijelova tijela u prostoru, poput INS (eng. *Inertial Navigation Systems*). Algoritam INS radi tako što uspoređuje poziciju i orijentaciju objekta s početnim ili prethodno pohranjenim stanjem. Pozicija se uglavnom računa pomoću akcelerometra i žiroskopa u uređaju, te se ponekad koristi i sustav GPS za preciznije određivanje pozicije objekta ili promatrača [9].

Praćenje zraka (eng. *Ray tracing*) ključna je tehnologija primjenjivana za realističan prikaz objekata u virtualnoj i proširenoj stvarnosti. Prema [10], radi se o tehnologiji simulacije padanja zraka svjetlosti na objekte koja se može primjenjivati i za realističan prikaz sjena i odraza svijetla. Primjena algoritma praćenja zraka je učinkovita za primjenu na sustave u stvarnom vremenu radi toga što se računanje položaja svakog piksela i zrake provodi neovisno, što dovodi do mogućnosti obrade različitih dijelova slike na različitim jezgrama procesne jedinice. Za razliku od ovih algoritama, tehnologija rasterizacije pretvara snimljene objekte u dvodimenzionalnu mrežu piksela, te se tijekom ovog procesa primjenjuju svojstva poput teksture i svjetlosti na objekte. Zatim se objekti integriraju nazad u vidokrug promatrača virtualnog okruženja, pružajući dojam fotorealističnosti. Rasterizacija zahtijeva manje procesne snage no daje lošije rezultate u odnosu na algoritme praćenja zraka [11].

Uz navedene algoritme i tehnologije, osim navedenih senzora i fizičkih komponenti uređaja, za postizanje realističnog okruženja virtualne, mješovite ili proširene stvarnosti, koriste se i haptički senzori za pružanje taktilnih povratnih informacija, zasloni visoke kvalitete te stereo zvuk.

2.2.2. Aktualni uređaji na tržištu

Iako je tehnologija proširene i virtualne stvarnosti dostupna već desetljećima, industrija se tek nedavno usredotočila na proizvodnju uređaja s punom podrškom za prikazivanje ovakvih sučelja. Najveću popularnost pronalaze uređaji za igranje računalnih igrica u virtualnoj stvarnosti dok proširena stvarnost dobiva veću primjenu na već postojećim pametnim telefonima i tablet uređajima.

2.2.2.1. Meta Quest

Prema informacijama dostupnim u [12], Meta, organizacija prethodno poznata pod imenom Facebook, u 2014. godini preuzima vlast nad organizacijom Oculus, poznatoj po istraživanju i proizvodnji uređaja za igranje računalnih igrica temeljenih u okruženju virtualne stvarnosti. Ubrzo izbacuje uređaj Meta Quest, koji je ujedno još uvijek najveći konkurent uređaju Apple Vision Pro. Meta Quest sadrži dvije RGB kamere kao i po jedan OLED zaslon visoke razlučivosti i frekvencije osvježavanja za svako oko. Podržava prikazivanje mješovite stvarnosti, kao i potpuno uranjanje u virtualnu stvarnost, ali za razliku od uređaja Vision Pro, ne sadrži senzore za praćenje pokreta očiju. Ovaj uređaj je prvenstveno usredotočen na pružanje dobrog iskustva igranja računalnih igrica, te ima značajno nižu cijenu u slobodnoj prodaji u usporedbi s Appleovim uređajem Vision Pro. Kućište uređaja prikazano je na slici 2.4.

Slika 2.4. Izložbeni uređaj Meta Quest 3 [13]

2.2.2.2. Microsoft HoloLens 2

Microsoft Hololens 2 je uređaj prvenstveno namijenjen uporabi u profesionalnim okruženjima, poput medicinskih ili obrazovnih [14]. Isto kao i Apple Vision Pro, Microsoft Hololens 2 sadrži skup infracrvenih kamera za praćenje pokreta očiju i četiri kamere visoke razlučivosti za praćenje pokreta glave. Podržava autentifikaciju korisnika skeniranjem rožnice, upravljanje glasovnim naredbama ili gestama ruku, te pruža do tri sata autonomnog korištenja bez punjenja. Za razliku od uređaja Apple Vision Pro koji snima korisnikovo okruženje te mu prikazuje video

istog okruženja na zaslonima, Microsoft Hololens 2 ima dva prozirna staklena zaslona koja funkcioniraju tako što se prikazuje sloj informacija povrh stvarnog okruženja, dajući dojam holograma koji reagiraju kao i fizički objekti prilikom korisničke interakcije. Zbog visoke razine integracije s Microsoftovim skupom profesionalnih aplikacija, uređaj je vrlo popularan u uredskim i profesionalnim okruženjima, no zbog manjka aplikacija, kao i relativno visoke cijene, ne dobiva na velikoj popularnosti na općem tržištu potrošačke elektronike.

2.2.3. Primjena mješovite stvarnosti na drugim mobilnim uređajima

S obzirom na sveprisutnost pametnih telefona i tablet računala, te njihovu visoku razinu funkcionalnosti i autonomije, jasno je kako oni predstavljaju dostupniju alternativu pristupanju mješovitoj stvarnosti. Uz kamere visoke razlučivosti i velike zaslone kao i dugotrajnu bateriju, postaju logičan izbor za ove svrhe, pa se zato javljaju razne aplikacije koje implementiraju sučelja zasnovana u mješovitoj stvarnosti. Primjerice, aplikacije proizvođača IKEA za sustave iOS i iPadOS omogućuje pretpregled namještaja u stvarnom okruženju. Koriste se kamera i senzori pametnog telefona kao i Apple ARKit razvojni okvir kako bi se dobio precizan prikaz objekata s jasnim prikazom sjena i tekstura [15].

Aplikacija Apple Measure, čije je sučelje prikazano na slici 2.5, razvijena je za uređaje iOS i iPadOS, te je temeljena na razvojnim okvirima Apple ARKit i Core Motion. Aplikacija služi za mjerenje udaljenosti ili dimenzija objekata koristeći podatke dobivene iz senzora LiDAR, akcelerometra, žiroskopa i kamere uređaja.

Slika 2.5. Sučelje aplikacije Apple Measure

2.3. Izazovi u području mješovite stvarnosti

Iako pružaju golem potencijal i razne primjene, uređaji koji koriste tehnologiju mješovite stvarnosti još uvijek su u ranim fazama razvoja. Jedan od najvećih izazova s kojim se ovaj tip uređaja susreće je kratko trajanje baterije. Zasloni visoke razlučivosti i sustavi s više kamera, kao i procesi obrađivanja snimanog okruženja zahtijevaju veliku količinu računalne snage. Trenutno nije moguće ugraditi baterije visokog kapaciteta u sam uređaj zbog povećanja mase i velikog prostora koji baterije zahtijevaju. Stoga se neki proizvođači okreću prema nosivim baterijama povezanih kablom na glavni uređaj. Ipak, ni ove baterije trenutno ne pružaju veliku autonomiju stoga se uređaji često moraju koristiti spojeni na punjač. Neki korisnici također prijavljuju i osjećaje mučnine i vrtoglavice pri duljem boravku u virtualnoj stvarnosti, kao i bolove na licu uzrokovane povećom masom ovih uređaja. Ovo može biti posljedica nedovoljno visoke razlučivosti zaslona ili niske frekvencije osvježavanja, prema [16]. Javlja se i problem pregrijavanja, zbog vrlo velike količine procesorske snage koja je potrebna za obrađivanje sadržaja mješovite stvarnosti u stvarnom vremenu.

Osim toga, razvoj aplikacija za mješovitu stvarnost često je neisplativ i težak proces, radi malenog broja trenutno aktivnih korisnika kao i posebnih razvojnih okvira. Potrebno je rukovati velikim količinama podataka i procesorske snage, stoga je zahtjevan proces optimizirati aplikaciju za ispravan rad u mješovitoj stvarnosti. Također, potrebno je optimizirati korisničko sučelje za rad u mješovitoj stvarnosti – elementi moraju biti prilagodljivi na razne uvjete osvjetljenja i veličine te korisničko sučelje mora biti jednostavno i razumljivo. Potrebno je i prilagoditi aplikacije za alternativne načine unosa i prikaza te uzeti u obzir ljude s tjelesnim oštećenjima. Isto tako, aplikacije moraju zadovoljavati visoke kriterije sigurnosti. Potrebno je biti vrlo oprezan oko prikupljanja korisničkih podataka i pogotovo oko slanja tih informacijama vanjskim organizacijama. Javljaju se opasnosti od maliciozne programske podrške koja bi snimala korisnikova okruženja ili bez dopuštenja dodavala trodimenzionalne objekte u korisnikovo okruženje. Podaci iz senzora također moraju biti vrlo dobro zaštićeni i privatni [17].

3. PLATFORMA VISION OS I UREĐAJ VISION PRO

U trećem poglavlju se opisuju uređaj Apple Vision Pro i operacijski sustav Apple visionOS. Posebna pozornost posvećena je detaljnoj analizi hardverskih komponenti koje čine osnovu ovog uređaja, poput ekrana, sustava kamera i baterije. Uz to, temeljito se istražuju različite metode kojima korisnici mogu upravljati uređajem.

3.1. Uređaj Apple Vision Pro

Apple Vision Pro je nosiv uređaj kojeg je razvila organizacija Apple Inc. s namjerom spajanja stvarnog svijeta s elementima virtualne i proširene stvarnosti, proizvodeći efekt takozvane 'miješane' stvarnosti. Najavljen je 5. srpnja 2023. godine na konferenciji Apple Worldwide Developer, te je nedugo zatim, 2. veljače 2024. pušten u slobodnu prodaju. Pruža jedinstveno korisničko iskustvo koje ima namjeru ostavljanja dojma besprijekorne integracije stvarnog svijeta i virtualnih objekata kao i aplikacija, što se postiže korištenjem skupa kamera visokih razlučivosti koje neprekidno snimaju i analiziraju objekte iz stvarnog svijeta [18]. Taj skup videa sustav koristi kao podlogu korisničkog sučelja i omogućava postavljanje dodatnih elemenata sučelja preko njih. Sve su te informacije dostavljene korisniku u stvarnom vremenu, uz minimalno kašnjenje. Uređaj Vision Pro može se spajati na Wi-Fi mreže, pristupati trgovini aplikacijama App Store, prikazivati video sadržaje, pregledavati mrežni sadržaj te nudi još velik broj značajki.

3.2. Sklopovlje uređaja Vision Pro

3.2.1. Kućište

Apple Vision Pro sastavljen je od kombinacije raznih materijala. Kućište je napravljeno od magnezija i grafitnog vlakna obloženih recikliranim aluminijem, a sustav zaslona sastoji se od kombinacije aluminija, stakla, poliestera, najlonske tkanine i polikarbonata. Zvučnici su, s obje strane, integrirani u posebno kućište napravljeno od aluminija, nehrđajućeg čelika kao i fluoroelastomera. Samo kućište mase je 600 do 650 grama, ovisno o konfiguraciji, uz odvojenu bateriju unutar aluminijskog kućišta mase 353 grama, prema informacijama dostupnim u [19]. Kućište uređaja prikazano je na slici 3.1.

Slika 3.1. Prikaz izložbenog uređaja Apple Vision Pro [20]

3.2.2. Procesorska snaga

Apple Vision Pro je pogonjen CPU jedinicom Apple M2, sastavljenom od osam jezgri od kojih su prve četiri zadužene za zahtjevnije zadaće, dok se druge četiri bave jednostavnijim zadaćama uz prioritiziranje štednje energije. SoC (eng. *System-on-a-chip*) dizajn procesora M2 temeljen je na 5-nanometarskoj tehnologiji i sastoji se od 20 milijardi tranzistora [21]. M2 čip kombiniran je s GPU jedinicom od 10 jezgri. Obradbena jedinica Neural Engine zadužena je za ubrzavanje operacija neuronskih mreža poput množenja matrica, a može izvoditi 15.8 TFLOPS. ISP, koprocesor za obradu signala (eng. *Image signal processor*) služi za smanjenje šuma slike pri procesiranju videa i fotografija. Sve biometrijske informacije uređaj pohranjuje u posebni čip Secure Enclave.

3.2.3. Senzori

Prema popisu specifikacija dostupnih u [22], od vanjskih senzora Apple Vision Pro sadrži dvije glavne kamere visoke razlučivosti, šest pomoćnih kamera, četiri kamere sa zadaćom praćenja očiju korisnika, kameru TrueDepth koja pomaže pri biometrijskoj autentifikaciji korisnika, optički mjerni instrument LiDAR (eng. *Light Detection and Ranging*), senzor treperenja te senzor za mjerenje ambijentalnog osvjetljenja.

3.2.4. Zasloni

Dva glavna zaslona uređaja Apple Vision Pro zasnovana su na tehnologiji zaslona MicroOLED (eng. *Microscopic Organic Light Emitting Diode*). U usporedbi s klasičnim OLED zaslonima, MicroOLED sadrži znatno manje piksele, pa tako glavni zasloni u uređaju Apple Vision Pro

zajedno imaju 23 milijuna piksela veličine 7.5 μm. Zasloni su u mogućnosti prikazati 92% boja opisanih spektrom DCI-P3, te podržavaju osvježavanje frekvencijama od 90Hz, 96Hz i 100Hz.

3.2.5. Baterija

Radi smanjenja mase glavnog dijela uređaja, baterija je odvojena u posebno aluminijsko kućište povezano kablom. Kapacitet baterije unutar kućišta je 3166 mAh. Baterija podržava maksimalni izlaz napona od 13V i struje 6A.

3.2.6. Optic ID

Optic ID je sustav biometrijske autentifikacije temeljen na prepoznavanju ljudske šarenice. Pri prvom postavljanju sustava Optic ID, infracrveno svjetlo, koje je bezopasno za ljudski vid, osvjetljava oči, dok kamere uređaja Apple Vision Pro snimaju sliku šarenice. Podaci o skeniranoj šarenici šalju se na pohranu u čip Secure Enclave, gdje se podaci transformiraju u brojčani format koji predstavlja jedinstvene značajke šarenice. Prilikom svake kasnije autentifikacije oka, koristi se sličan proces kako bi se usporedila nova slika korisnikove šarenice s pohranjenim biometrijskim podacima. Optic ID je prilagodljiv na promjene u osvjetljenju jer pri svakoj uspješnoj autentifikaciji ažurira pohranjene podatke u čipu Secure Enclave. Podaci su šifrirani, a šanse slučaje gdje bi nasumična osoba ljudske populacije svojim okom mogla otključati Vision Pro koji ne pripada njima je manja od jedan prema milijun [23].

3.3. Načini interakcije s uređajem Apple Vision Pro

3.3.1. Snimanje ruku

Apple Vision Pro koristi šest vanjskih kamera u kombinaciji sa senzorom LiDAR kako bi pratio položaj ruku korisnika i tako omogućio interakciju s trenutno prikazanim elementima sučelja. LiDAR emitira svjetlosne impulse i detektira povratne signale iz okruženja. Vrijeme povratka svjetla do senzora se koristi za izračun udaljenosti, te se stvara trodimenzionalna mapa okruženja. Pomoću ovih podataka, sustav je u stanju ustanoviti relativnu lokaciju ruku korisnika [24]. Sustav od šest kamera neprekidno snima okruženje, pa tako i korisnikove ruke, te koristi dobivene podatke kao pomoć senzoru LiDAR za očitavanje gesti koje korisnik napravi. Geste rukama su glavni način interakcije sa sustavom. Sustav podržava šest različitih gesti, od kojih su dvije zamišljene za izvođenje korištenjem dviju ruku. Pritiskanje palca i kažiprsta je ekvivalent dodira zaslona na pametnom telefonu, predstavlja gestu „*Tap*“ kojom se odabire sadržaj na prozoru aplikacije. Zadržavanjem palca i kažiprsta zajedno uz povlačenje ruke gore ili dolje aktivira se gesta „*Scroll*“ za pomicanje kroz prikazani sadržaj. Sve geste su

popraćene vizualnim i zvučnim povratnim informacijama od sučelja. Razvojni programeri imaju mogućnost kreirati proizvoljne geste za svoje aplikacije i igre, no moraju paziti kako bi te geste bile razumljive i ne dolazile u konflikt s drugim, sličnim gestama.

3.3.2. Praćenje očiju

Skup malenih svjetlećih dioda osvjetljava oko uz pomoć četiri infracrvene kamere. Zajedno stvaraju uzorak svijetla na svakom oku koji koriste za očitavanje informacija o poziciji svakog oka. Dvije dodatne kamere visoke razlučivosti neprestano snimaju oči, dok sustav te informacije koristi kako bi omogućio interakciju sa sučeljem. Korisnik pogleda u element sučelja koji želi odabrati, te ga uz pomoć geste odabire. Samim gledanjem u element, pojavljuje se tzv. „*hover*“ učinak nad svakim elementom kako bi korisniku bilo jasno koji element je u fokusu. Kombinacija položaja očiju skupa s gestama ruku omogućava složene interakcije sa sustavom. Primjerice, ako korisnik otvori sliku ili video, može gledati u određeno područje i pomicati obje ruke kako bi povećao to područje slike ili videa.

3.3.3. Digitalna kruna

Apple Vision Pro na sebi sadrži i tzv. digitalnu krunu, koja služi kao metoda unosa za promjenu glasnoće, izlaz iz aplikacije, otvaranje određenih postavki, centriranje sadržaja ispred očiju, te u slučaju uporabe jedne od predinstaliranih pozadina, promjenu razine vidljivosti odabrane pozadine.

3.3.4. Pristupačnost

Apple Vision Pro podržava velik broj različitih opcija za interakciju sa sučeljem. Značajka „*VoiceOver*“ je napredni čitač zaslona koji koristi zvučne opise elementa sučelja kako bi korisniku opisao vidljive elemente sučelja. *VoiceOver* može opisivati ljude, objekte, grafove i čitati tekst u preko 60 jezika, te nudi razne opcije za tip glasa i brzinu govora. Značajka „*Zoom*“ povećava odabrani sadržaj na zaslona kako bi ga učinila lakše vidljivim korisniku. Može se povećati određen prozor ili čitavo sučelje. „*Dwell Control*“ omogućava uporabu uređaja bez korištenja ruku, dozvoljavajući upravljanje sučeljem pogledom. Dulji pogled na element sučelja se može ponašati kao klik. Značajka „*Pointer Control*“ omogućava upravljanje sustavom uz pomoć ruku, bez interakcije pogledom, dodavajući pokazivač na zaslon kojim se upravlja pomicanjem ruke. Sustavom se može upravljati i glasovnim naredbama korištenjem značajke „*Siri*“. Siri je virtualni pomoćnik koji omogućava otvaranje aplikacija, pretraživanje sadržaja kao i interakciju unutar aplikacije uz razne druge mogućnosti.

Osim navedenog, Apple Vision Pro podržava spajanje vanjskih tipkovnica, miševa, slušalica i kontrolera preko tehnologije Bluetooth.

3.4. Operacijski sustav Apple visionOS

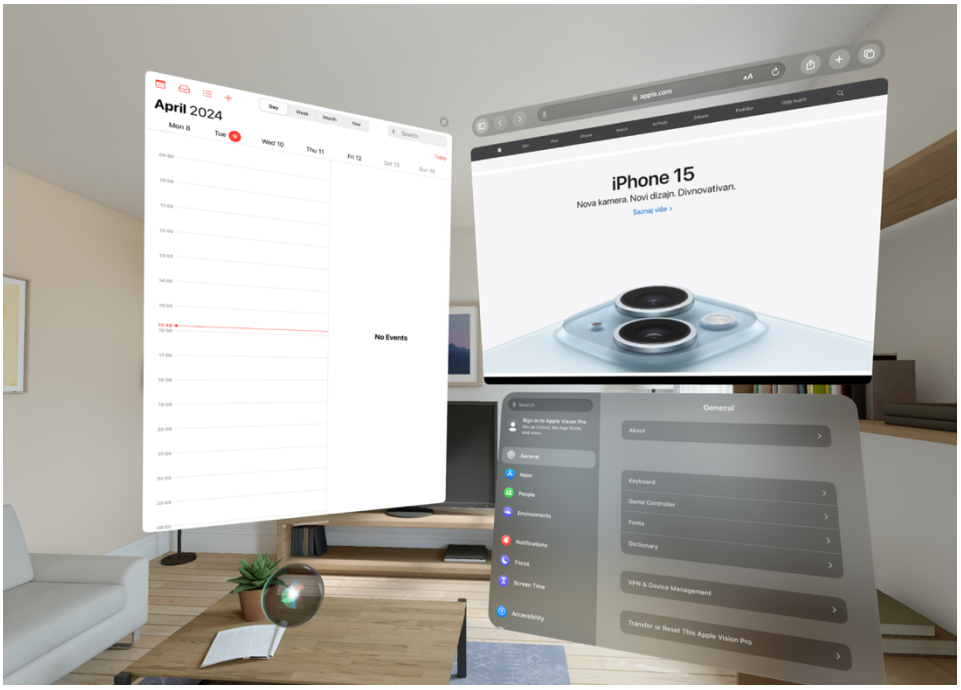
Apple Vision Pro je pokretan operacijskim sustavom Apple visionOS, koji se temelji na sustavima Apple iOS i iPadOS. Sustav je razvijen isključivo za uređaj Apple Vision Pro. Predstavljen je na konferenciji Apple Worldwide Developer 5. lipnja 2023. godine, te je u vrijeme pisanja ovog rada aktualna inačica 1.1.1. Sustav implementira vizualno sučelje kojim se upravlja glasom, očima i gestama ruku. Podržava instaliranje aplikacije trećih strana, web-preglednike kao i emuliranje aplikacija s okruženja iOS i iPadOS. Sustav dolazi s velikim brojem unaprijed instaliranih aplikacija, kao što su: kalendar, e-pošta, preglednik, galerija fotografija, svirač glazbe, postavke, bilješke i slično. Početni zaslon sustava prikazan je na slici 3.2.

Slika 3.2. Početni zaslon sustava VisionOS

Pozadina sučelja se generira na temelju videa snimanih vanjskim kamerama uređaja, no moguće je i simulirati razna unaprijed instalirana okruženja, kao što je vidljivo na slici 3.3.

Slika 3.3. Izbori pozadine sustava VisionOS

Aplikacije su predstavljene u trodimenzionalnim prozorima promjenjive veličine koji reagiraju na promjene u okruženju, te se mogu proizvoljno pomicati u prostoru. Osim toga, sustav podržava zrcaljenje zaslona uređaja Mac, što također predstavlja kao prozor. Za razliku od ostalih Appleovih operacijskih sustava, visionOS ne podržava izmjenu tamnog i svijetlog načina rada. Umjesto toga, prozori prate dizajn nadahnut staklenim površinama, te mijenjaju izgled ovisno o osvjetljenju u fizičkom okruženju korisnika. Na vrhu svakog prozora nalazi se gumb za aktivaciju kontrolnog centra – dijaloga koji omogućuje brzo upravljanje postavkama uređaja kao i brzo upravljanje pozadinama i izmjenu korisnika. Apple je omogućio razvojnim programerima razvoj i distribuciju aplikacija za sustav visionOS preko integriranog razvojnog okruženja Xcode. Prikaz pokrenutih aplikacija u sustavu visionOS vidljiv je na slici 3.4.



Slika 3.4. Prikaz višezadaćnosti u sustavu visionOS

Sustav visionOS pruža potpunu podršku za emuliranje aplikacija dizajniranih za iOS i iPadOS, te ih odvaja u posebnu mapu unutar početnog zaslona. Kao i standardne aplikacije, ove aplikacije podržavaju promjenu veličine prozora, rad s više prozora i unos preko tipkovnice. Neke od Appleovih vlastitih aplikacija se također pokreću u ovom načinu rada. Navedeno ponašanje vidljivo je na slici 3.5.

Slika 3.5. Prikaz pokretanja iPadOS aplikacija na sustavu visionOS

4. MODEL I PROGRAMSKO RJEŠENJE APLIKACIJE

Četvrto poglavlje navodi kako i što aplikacija treba raditi preko funkcionalnih i nefunkcionalnih zahtjeva i opisuje implementaciju programskog rješenja aplikacije, ulazeći u detalje implementacije ključnih elemenata projekta, poput sustava za autentifikaciju i trajnu pohranu podataka, implementaciju obrasca *Model-View-ViewModel* i mrežnog repozitorija za dohvaćanje podataka. Objašnjavaju se ključne tehnologije i programski obrasci i razvojni okviri korišteni za razvoj kao i razvojno okruženje Xcode i distribucijska platforma App Store.

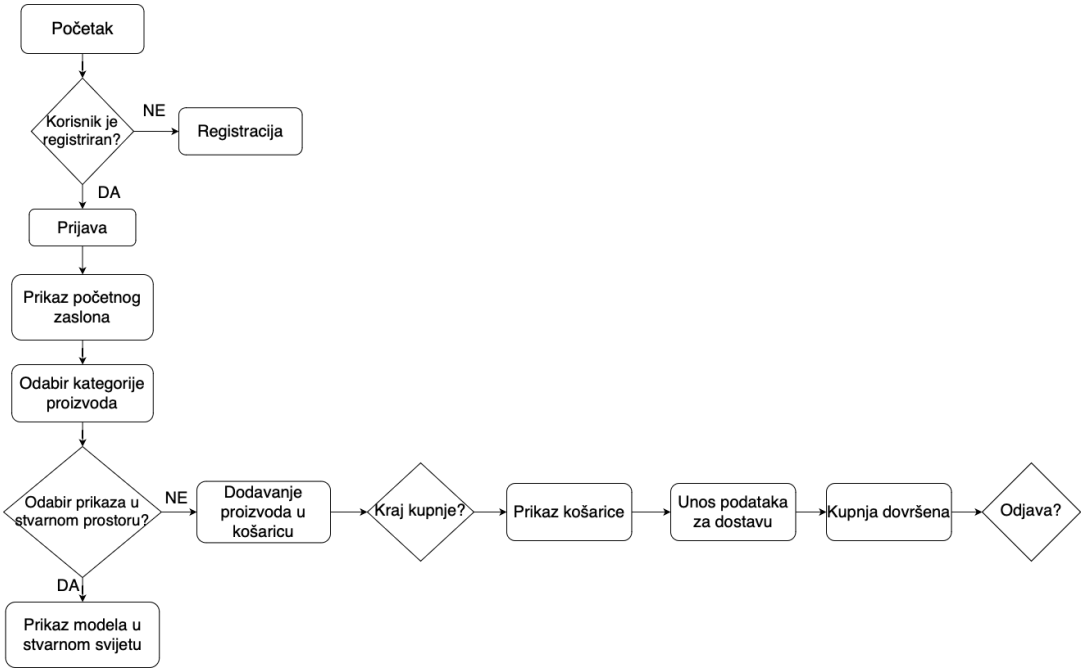
4.1. Funkcionalni i nefunkcionalni zahtjevi na aplikaciju

Aplikacija „Showroom“ razvijena je s unaprijed postavljenim funkcionalnim i nefunkcionalnim zahtjevima. Aplikacija mora zadovoljavati ove zahtjeve kako bi pružila optimalno korisničko iskustvo i bila prikladna za korištenje. Funkcionalni i nefunkcionalni zahtjevi navedeni su u tablici 4.1.

Tablica 4.1. Funkcionalni i nefunkcionalni zahtjevi aplikacije

Nefunkcionalni zahtjevi	Funkcionalni zahtjevi
Podaci prijave i registracije moraju biti zaštićeni i šifrirani	Registracija korisnika preko mail adrese i proizvoljne lozinke
Aplikacija se treba moći prilagoditi različitim veličinama prozora	Prijava korisnika preko mail adrese i lozinke
Aplikacija mora pružati poruke o pogreškama korisniku	Filtriranje proizvoda po kategorijama
Korištenje kamera uređaja za prikaz 3D modela u prostoru	Pretraživanje proizvoda unutar kategorija
	Mogućnost dijeljenja informacija o proizvodu
	Označivanje proizvoda omiljenim
	Dodavanje proizvoda u košaricu

Dijagram toka aplikacije vidljiv je na slici 4.1.



Slika 4.1. Dijagram toka korištenja aplikacije

4.2. Korištene programske tehnologije, jezici i razvojne okoline

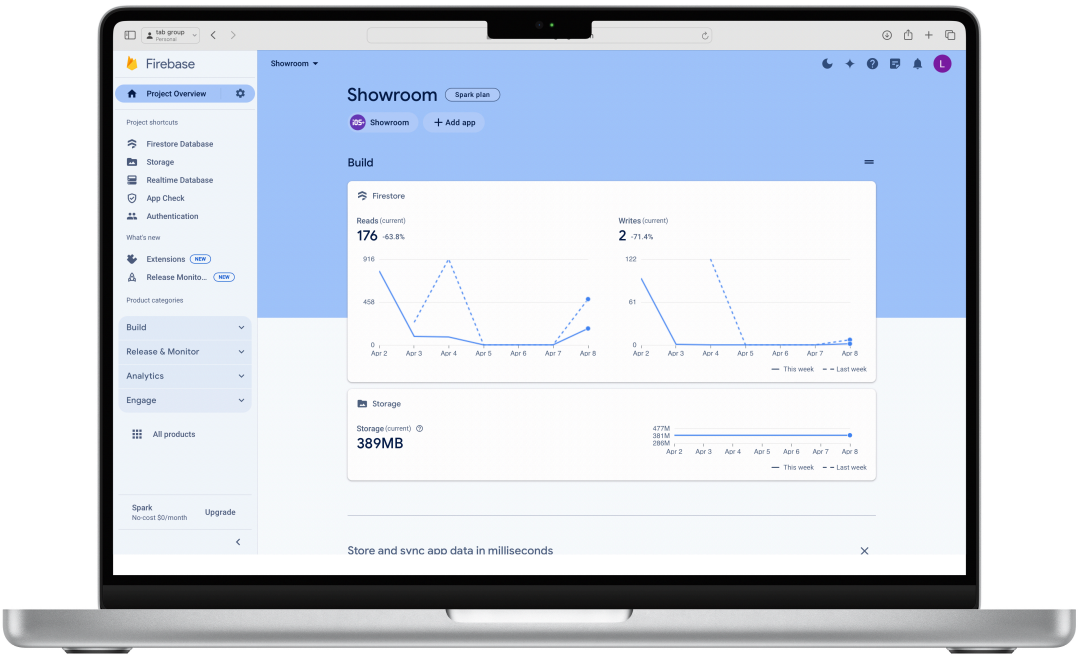
4.2.1. Vanjske biblioteke

Vanjske biblioteke implementirane su u projekt pomoću alata Swift Package Manager, koji je integriran u okruženje Xcode, a služi za automatska povezivanja, preuzimanja i ažuriranja vanjskih programskih paketa (eng. *dependencies).* Vanjske biblioteke uključene su u aplikaciju kako bi olakšale i ubrzale razvoj i održavanje različitih značajki, poput povezivanja na bazu podataka ili registracije i prijave. One su zbirke unaprijed definiranih i unaprijed sabranih klasa i funkcija koje se lako mogu uključiti u projekt. Za potrebe ove aplikacije uključene su biblioteke poput Firebase, GoogleAppMeasurement i GoogleDataTransport.

4.2.2. Baza podataka

Aplikacija rukuje trodimenzionalnim modelima koji zahtijevaju veliki prostor za pohranu, stoga ju je bilo potrebno povezati na udaljenu bazu podataka. Ovaj proces doveo je do značajne uštede na veličini aplikacije, s obzirom na to da se modeli ne moraju pohranjivati lokalno nego se učitavaju s vanjskog poslužitelja. Osim toga, implementacijom baze podataka je znatno olakšano ažuriranje i dodavanje novih modela proizvoda za sve korisnike. Također, bilo je potrebno riješiti pitanje autentifikacije korisnika kao i pohrane i pamćenja individualnih postavki za svakog korisnika, poput liste omiljenih proizvoda. S obzirom na sve navedeno, odabrana je platforma Google Firebase za potrebe razvoja aplikacije. Značajka Firebase

Authentication omogućuje gotov sustav autentifikacije korisnika, uključujući stvaranje korisničkih računa, prijavu kao i upravljanje greškama koje mogu nastati tijekom prijave. Podržava sigurnu prijavu pomoću računa e-pošte, broja telefona kao i preko vanjskih usluga za prijavljivanje. Firebase Storage usluga je koja omogućuje besplatnu pohranu velikih datoteka uz visoke razinu sigurnosti i skalabilnosti. Za potrebe ove aplikacije, koristi se za pohranu trodimenzionalnih modela proizvoda. Firebase Firestore skalabilna je baza podataka zasnovana na Google Cloud infrastrukturi koja omogućuje pohranu i upravljanje modelima podataka. Pruža hijerarhijski, organiziran pregled datoteka kojima se može upravljati preko konzole. Konzola također nudi pregled informacija o zauzeću memorije, broju preuzimanja aplikacije, napredne analitike kao i praćenje svih instanci ispada aplikacije. Prikaz sučelja konzole Firebase vidljiv je na slici 4.2.



Slika 4.2. Prikaz mrežne aplikacije Firebase

4.2.3. RealityKit

Prema [25], RealityKit je razvojni okvir temeljen na tehnologiji Apple ARKit koji se koristi za integraciju virtualnih objekata u stvarni svijet.

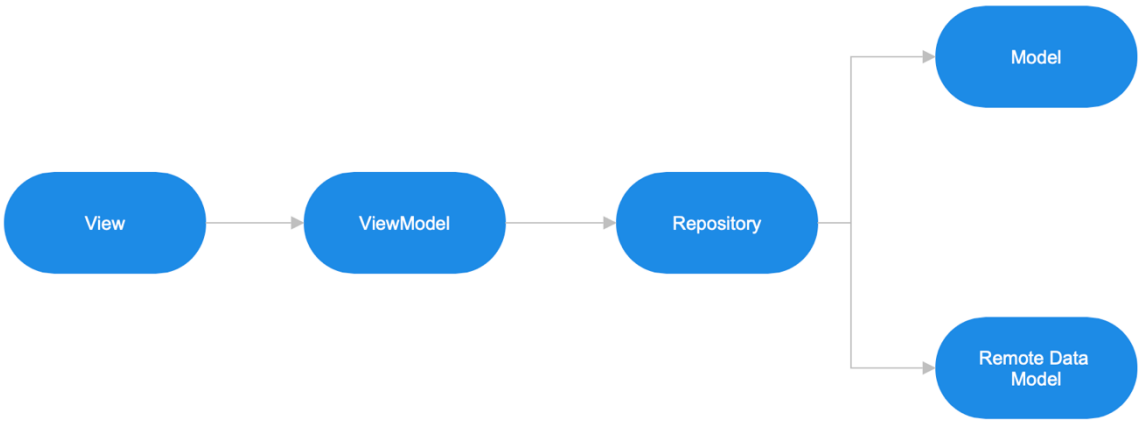
Podržava animiranje objekata, prikaz različitih materijala i odsjaja iz okruženja, sjene i slične vizualne efekte kako bi objekti izgledali što bliže stvarnima. Objekti prikazani preko okvira RealityKit mogu reagirati na temelju korisničke interakcije, biti sinkronizirani između više uređaja u stvarnom vremenu i imati interakcije sa stvarnim svijetom. RealityKit koristi grafički

procesor uređaja kako bi se postigla što bolja izvedba i učitavanje modela. Osim navedenog, moguće je podešavati svojstva virtualnih modela poput mase kako bi se postizali realistični sudari između više modela u prostoru.

4.2.4. Pregled programske građe aplikacije

Aplikacija je izrađena u okviru obrasca MVVM uz dodatan sloj mrežnog repozitorija. MVVM je obrazac razvoja programske podrške koji razdvaja programsku logiku od korisničkog sučelja [26]. Postoje tri glavne komponente arhitekture MVVM, a to su *View, Model* i *ViewModel.* Sloj *Model* odgovoran je za pohranu podatkovnih objekata modela. On komunicira s *ViewModel*om i nije izravno izložen sloju *View*. *ViewModel* je veza između *Modela* i *Viewa*. Sadrži većinu programskog koda poslovne logike, te se brine za komunikaciju između slojeva *View* i *Model* tako što rukuje podacima iz *Modela* i prosljeđuje ih u *View* u obliku lako razumljivom za korisnika [27]. *View* predstavlja podatke u prikladnom obliku za korisnika, reflektirajući trenutno stanje podataka. Idealno ne sadrži poslovnu logiku nego ima referencu na *ViewModel* koji implementira logiku i prikuplja podatke o korisničkim interakcijama [28].

Uz MVVM, aplikacija sadrži i poseban sloj mrežnog repozitorija. Ovaj sloj služi za dohvaćanje podataka s udaljenog poslužitelja i rukuje potencijalnim pogreškama pri dekodiranju ili dohvaćanju podataka. Repozitorij također ima referencu na strukture podataka iz sloja *Modela,* te ih pohranjuje nakon uspostavljanja mrežnog poziva. Tim podacima zatim pristupa *ViewModel* koji ima referencu na repozitorij. Odvajanje programske logike i podataka od korisničkog sučelja sa sobom donosi brojne prednosti, poput efikasnijeg proces izrade aplikacije gdje se više timova može posvetiti na razvoj različitih slojeva. Osim toga, mogu se pisati posebni testovi za slojeve *ViewModel* i *Model* bez pristupanja sloju *View*. Primjer arhitekture MVVM uz implementaciju mrežnog repozitorija vidljiv je na slici 4.3.



Slika 4.3. Prikaz građe arhitekturnog obrasca MVVM

4.2.5. Programski jezik Swift

Prema [29], Swift je Appleov programski jezik predstavljen 2014. godine. Predstavlja programski jezik prvenstveno namijenjen stvaranju programa za prijenosna i stolna računala kao i programske podrške za poslužitelje. Swift je siguran programski jezik lako čitljive sintakse, dizajniran kako bi maksimalno onemogućio pisanje programskog koda sklonog greškama. Swift automatski zaključuje tipove varijabli a memorijom se upravlja automatski preko brojanja referenci na svaki objekt. Objekti prema zadanim postavkama ne mogu imati *nil* vrijednost, ali se razvojnim programerima dozvoljava da ručno postave svojstva koja bi potencijalno smjela biti prazna kao tzv. *Optional* varijable. Swift podržava i asinkrono programiranje korištenjem *async-await* sintakse, te pisanje programskog koda koji se paralelno izvodi uz punu podršku za višenitnost. Swift također u potpunosti podržava objektno-orijentiranu paradigmu programiranja, pa se tako mogu koristiti klase, protokoli i nasljeđivanje. Uz navedeno, Swift uključuje automatsko upravljanje greškama pomoću sintakse *do-catch-try*. Swift je zasnovan na LLVM kolekciji modularnih i ponovno upotrebljivih tehnologija za razvojna okruženja, stoga se programski kod izvodi brzo uz maleno zauzeće memorije. Svojstva klase se označavaju sa *let* i *var* oznakama, ovisno o tome jesu li konstantna ili promjenjiva, dok se metode označuju ključnom riječi *func.* U Swiftu, metode mogu vraćati više vrijednosti odjednom, a mogu i uzimati druge metode kao argumente.

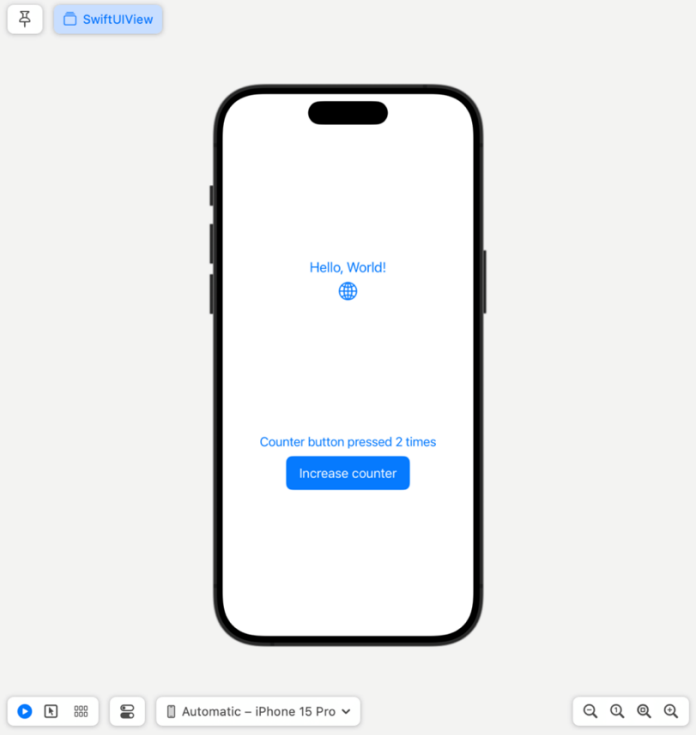
4.2.6. Razvojni okvir SwiftUI

SwiftUI je razvojni okvir za izgradnju mobilnih aplikacija na deklarativan način. Kompatibilan je s Appleovim platformama iOS, iPadOS, visionOS, macOS, tvOS i watchOS. Deklarativan okvir za razvoj aplikacija funkcionira tako što se izbjegava ručno navođenje svih prijelaza koji dolaze pri osvježavanju sučelja zbog promjena u aplikaciji, uz definiranje izgleda sučelja za više različitih stanja, te se za osvježavanje i prijenos između različitih stanja sučelja pobrine sam razvojni okvir. Prijelazi između stanja se najčešće izvode zbog promjene podataka ili korisničkih interakcija s aplikacijom [30]. U okviru SwiftUI, svaki element koji se prikazuje na zaslonu zove se *View. View* predstavlja dijelove korisničkog sučelja, poput prozora, gumba ili slike. Ovakvi elementi mogu se slagati u složenije *View* elemente. Primjer elementa *View* vidljiv je na slici 4.4.

```
struct SwiftUIView: View {
    @State private var counter = 0

    var body: some View {
        VStack(spacing: 15) {
            Spacer()

            Text("Hello, World!")
                .font(.title2)

            Image(systemName: "globe")
                .scaleEffect(1.7)

            Spacer()

            Text("Counter button pressed \(counter.description)
\(counter == 1 ? "time " : "times")")
                .animation(.easeInOut, value: counter)
                .font(.title3)

            Button {
                counter += 1
            } label: {
                Text("Increase counter")
                    .padding(.all, 5)
            }
            .buttonStyle(.borderedProminent)

            .foregroundStyle(.white)
            .scaleEffect(1.2)

            Spacer()
        }
        .foregroundStyle(.blue)
    }
```

Slika 4.4. Prikaz elementa SwiftUI View

SwiftUI podržava i funkcionalnost *Preview*, što omogućuje vizualan prikaz trenutno napisanog programskog koda. *Preview* se automatski prikazuje unutar datoteke koja se uređuje. Primjer programskog koda dan u izlistanju stvara jednostavan SwiftUI *View* koji se sastoji od tekstova, ikone te gumba, vidljiv na slici 4.5.



Slika 4.5. Prikaz izrađenog zaslona u pregledniku SwiftUI Preview

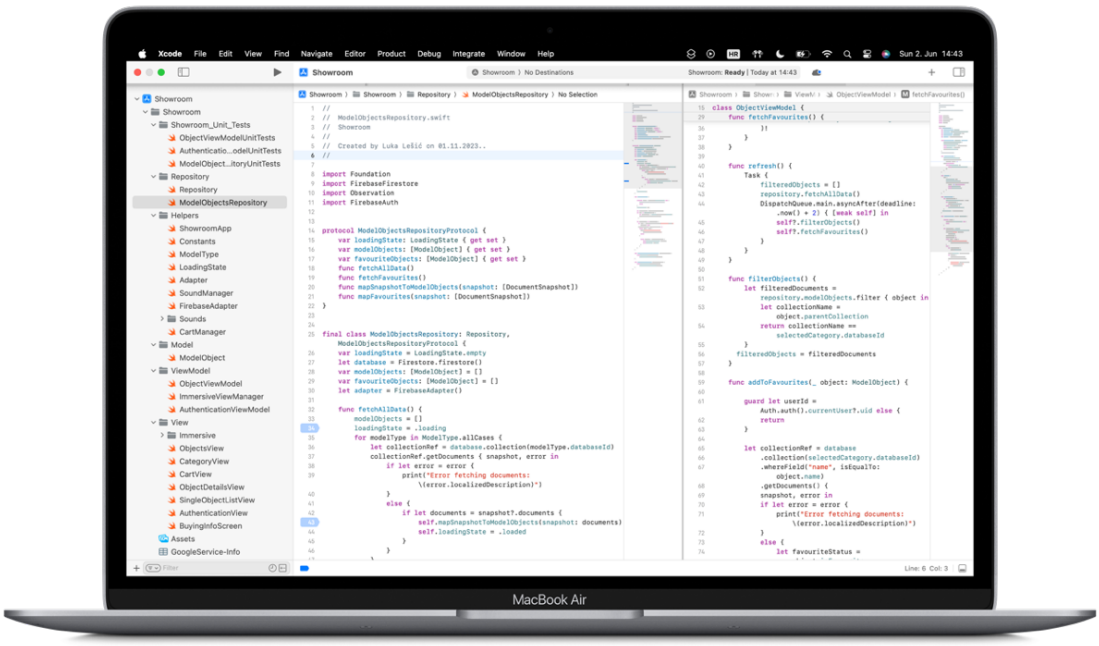
Elementi su uključeni u *View* tipa *Stack,* koji služi za pozicioniranje drugih elemenata tipa *View* unutar sebe. *VStack* pozicionira elemente u vertikalnom smjeru, *HStack* to radi za horizontalni smjer, dok *ZStack* slaže elemente jedan iznad drugog, ostavljajući dojam trodimenzionalnosti u aplikacijama. Elementi *Stack* pružaju jednostavniju alternativu ručnom postavljanju pozicija i ograničenja za svaki element na zaslonu. Elementi unutar komponente *Stack* se ravnomjerno raspoređuju te prilagođavaju prikazima na uređajima različitih veličina zaslona. Pomoću deklaracije *Spacer* stvara se prazan prostor, te se elementi unutar komponente *Stack* poguraju prema tom prostoru. *Spacer* automatski pokušava zauzeti sav dostupan prostor na zaslonu. Za elemente tipa *Stack* koji nemaju definiran *Spacer* unutar sebe, SwiftUI automatski odredi veličinu na temelju računice idealnih širina i visina elemenata unutar *Stack*a.

Na elemente prikazane u slici 4.5. dodani su modifikatori za boju i veličinu. Modifikatori u SwiftUI okruženju omogućuju promjenu izgleda elementa. Moguće je manipulirati veličinom, bojom, pozicijom, sjenama, marginom kao i obrubima elemenata. U okruženju SwiftUI koriste se varijable stanja kako bi se kreirali dinamički elementi tipa View koji se mogu mijenjati na temelju promjene podataka ili korisničke interakcije. U primjeru iz izlistanja programskog koda, vrijednost brojača označava se kao varijabla stanja zbog čega se može uz animaciju mijenjati pri pritisku gumba. Animacije se ostvaruju pomoću modifikatora *animation*, kojemu se predaje željeni tip animacije i varijabla koja će potaknuti promjenu elementa. Ovime se

omogućuje izrada složenih animacija i nudi mogućnosti prilagodbe brzine, tipa animacije i ponavljanja animacije unaprijed postavljen broj puta.

4.2.7. Integrirano razvojno okruženje Xcode

Xcode je naziv za skup alata za programere namijenjenih za izradu i objavljivanje aplikacija na trgovinu aplikacija App Store i platformu TestFlight za testiranje aplikacija. Može se preuzeti isključivo na Apple Mac uređaje s operacijskim sustavom macOS [31]. Prikaz sučelja okruženja Xcode prikazan je na slici 4.6. Xcode uključuje uređivač programskog koda skupa s alatima za ispravljanje pogrešaka u programskom kodu kao i značajkama pretpregleda izgleda aplikacije. Nudi podršku i za povezivanje na Git platforme radi brzog verzioniranja programskog koda, te alate za detektiranje ispada aplikacije i curenja memorije. Xcode podržava razvoj programskog koda napisanog u jezicima: Python, Swift, Ruby, C++, C, Objective-C, Java, AppleScript, i ResEdit. Podržava i testno okruženje zvano Playgrounds, koje omogućava pisanje programskog koda i trenutni prikaz njegovog izlaza ili rezultata. Proces izrade aplikacije ovog rada započeo je uporabom okruženja Xcode 15 beta 8, te se kasnije uporabljuje Xcode 15.3.



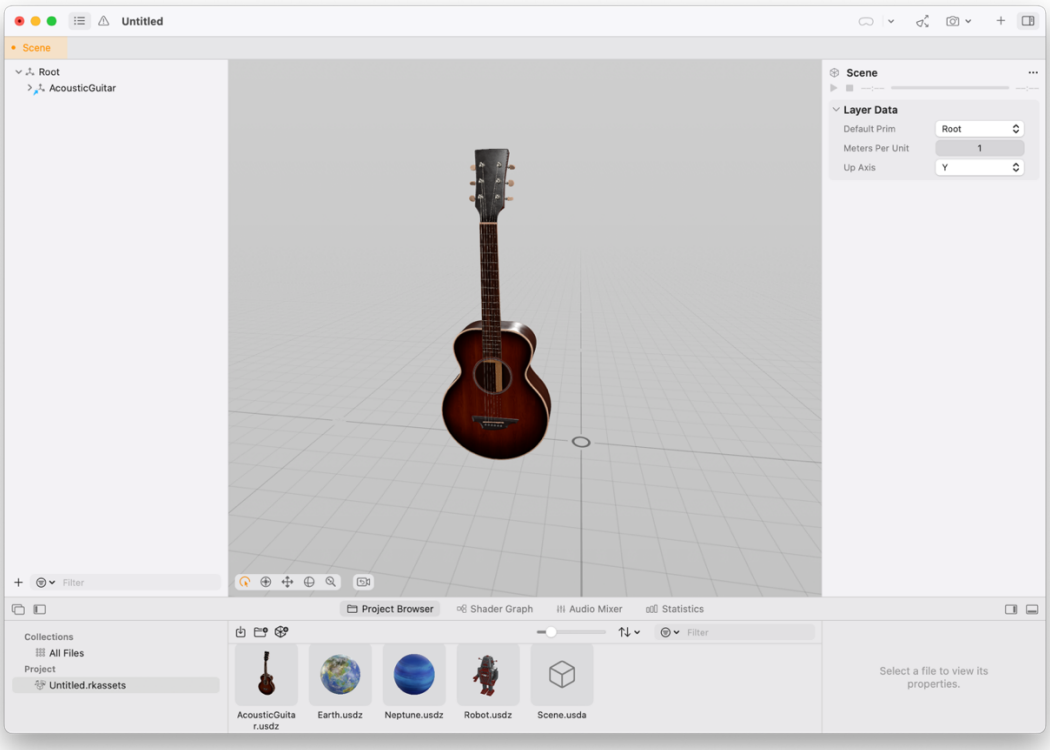
Slika 4.6. Prikaz integriranog razvojnog okruženja Xcode na uređaju Apple MacBook

4.2.8. Reality Composer Pro

Reality Composer Pro je alat namijenjen razvojnim programerima koji služi za dizajniranje i pregledavanje 3D sadržaja. Omogućava izradnju 3D modela u okruženju usko integriranom u Xcode. Reality Composer Pro omogućava pozicioniranje, skaliranje i konfiguriranje svojstava uvezenih modela u formatu USDZ. To je tip podatka koji su razvile organizacije Apple i Pixar, te mu je glavni cilj olakšati prikaz i dijeljenje trodimenzionalnih modela objekata. Neke od prednosti koje nudi su vrlo uska integracija s Appleovim operacijskim sustavima i programskim okvirima, nisko zauzeće radne memorije pri prikazivanju i brzo učitavanje s mreže. Od nedostataka su prisutni manjak kompatibilnosti s drugim platformama izvan Apple ekosustava i veličina datoteke kod složenijih modela. RealityView predstavlja SwiftUI *View* koji omogućava integriranje modela napravljenih u programu Reality Composer Pro unutar aplikacije. Reality Composer Pro podržava i biblioteku USDZ modela koji su dostupni na korištenje u javnim projektima. Primjer koda za integraciju elementa RealityView u programskom jeziku Swift vidljiv je na slici 4.7, a sučelje programa Reality Composer Pro na slici 4.8.

```
RealityView { realityContent in
    do {
        let entity = try await Entity(named: "Entity", in:
realityKitContentBundle)
        realityContent.add(entity)
    } catch {
        // Handle error
    }
}
```

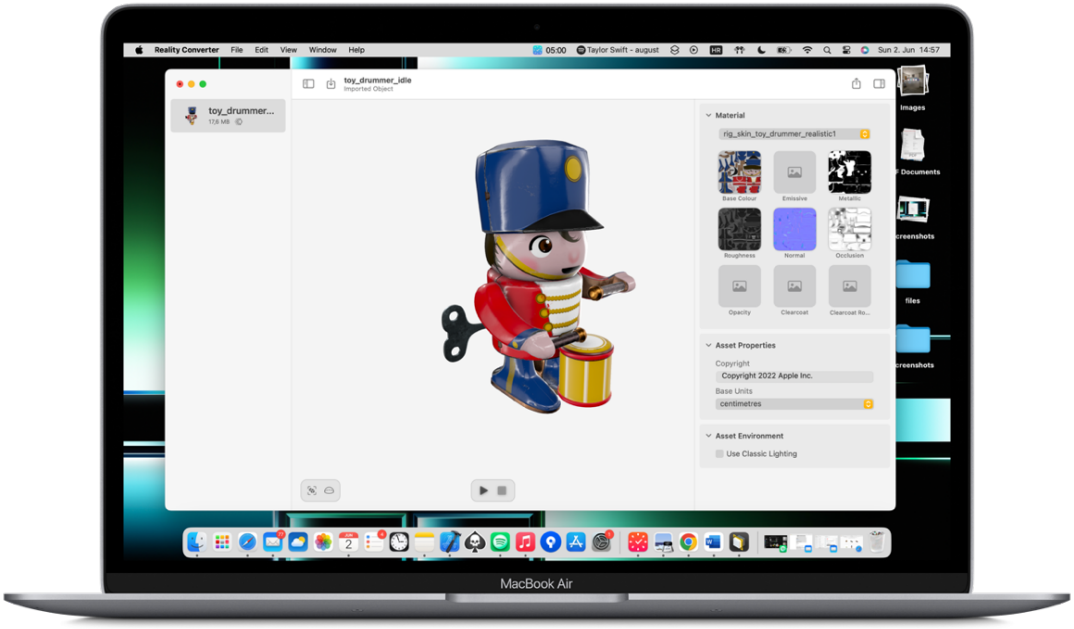
Slika 4.7. Programski kod za integraciju elementa RealityView u SwiftUI



Slika 4.8. Sučelje Reality Composer Pro programa na sustavu macOS

4.2.9. Reality Converter

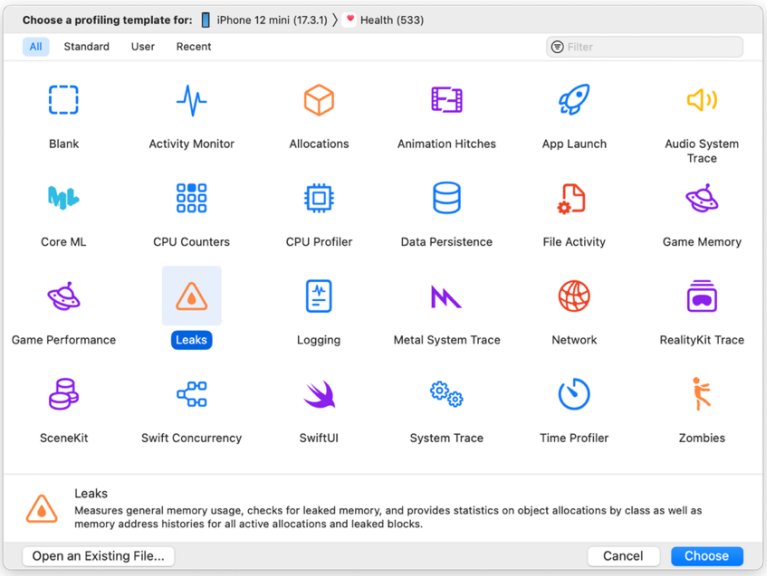
S obzirom da je USDZ Appleov preferirani tip podataka za rad s mješovitom stvarnosti, Reality Converter je program razvijen s namjerom jednostavnog pretvaranja datoteka formata poput .obj i .gltf u format USDZ. Alat je namijenjen za korištenje skupa s programom Reality Composer Pro, s namjerom ubrzanja postupka integriranja 3D sadržaja u Swift aplikacije. Alat sadrži razne značajke poput pretpregleda datoteka, pretvaranja formata datoteka i izmjene metapodataka datoteka, te brzog izvoza datoteka. Sučelje alata Reality Converter prikazano je na slici 4.9.



Slika 4.9. Prikaz Reality Converter programa na uređaju Apple MacBook

4.2.10. Instruments

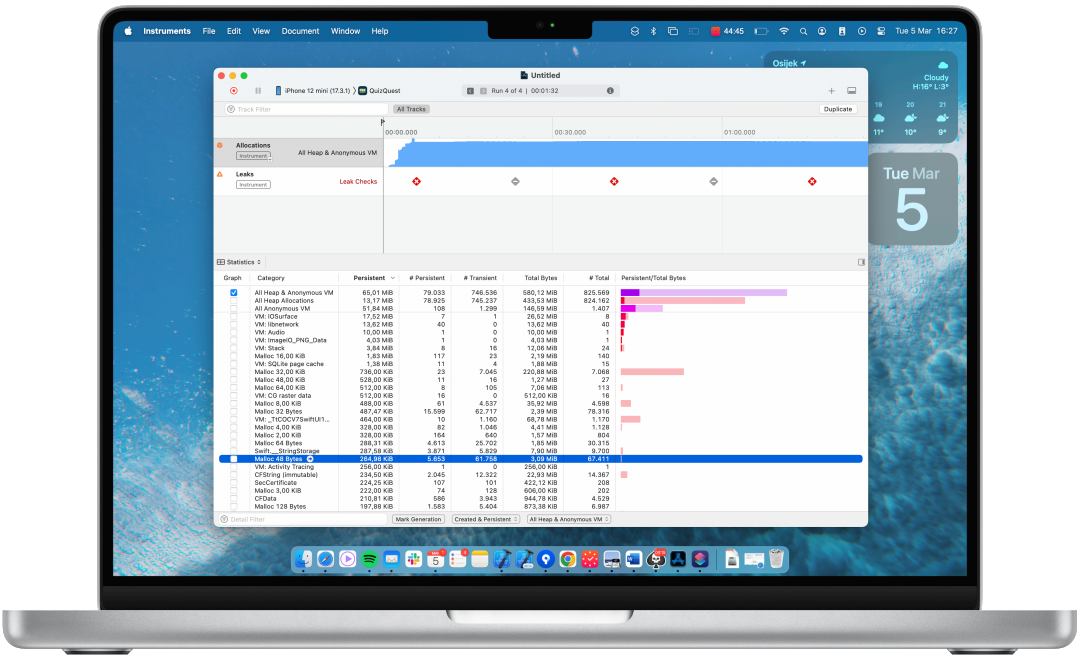
Instruments je dio Xcode okruženja koji pruža skup alata namijenjenih poboljšanju izvedbe aplikacije. Sadrži alate za provjeru nestabilnosti aplikacije, zastajkivanja animacija, gubitka performansi, nepotrebnih gubitaka radne memorije (eng. *Memory leaks*), te prevelike potrošnje baterije i ostalih resursa.



Slika 4.10. Prikaz programa Xcode Instruments na sustavu macOS

Slika 4.10 prikazuje dostupne alate pri pokretanju aplikacije Instruments. Instruments se koristi tako što se željeni uređaj spoji na računalo, nakon čeg se odabere željeni alat, aplikacija koju će se analizirati i odabere gumb „*Record*“. Započinje se praćenje odabranih parametara (memorija, baterija, animacije..) dok se na uređaju aplikacija pokreće. Moguće je normalno koristiti aplikaciju tijekom praćenja. Nakon završetka praćenja, dobije se detaljna analiza svih otkrivenih poteškoća u aplikaciji za testirane scenarije korištenja, koja se može pohraniti na računalo i dalje dijeliti.

Instruments se može preuzeti i u trgovini aplikacija Mac App Store, neovisno o prisutnosti okruženja Xcode u sustavu. Sučelje aplikacije vidljivo je na slici 4.11.

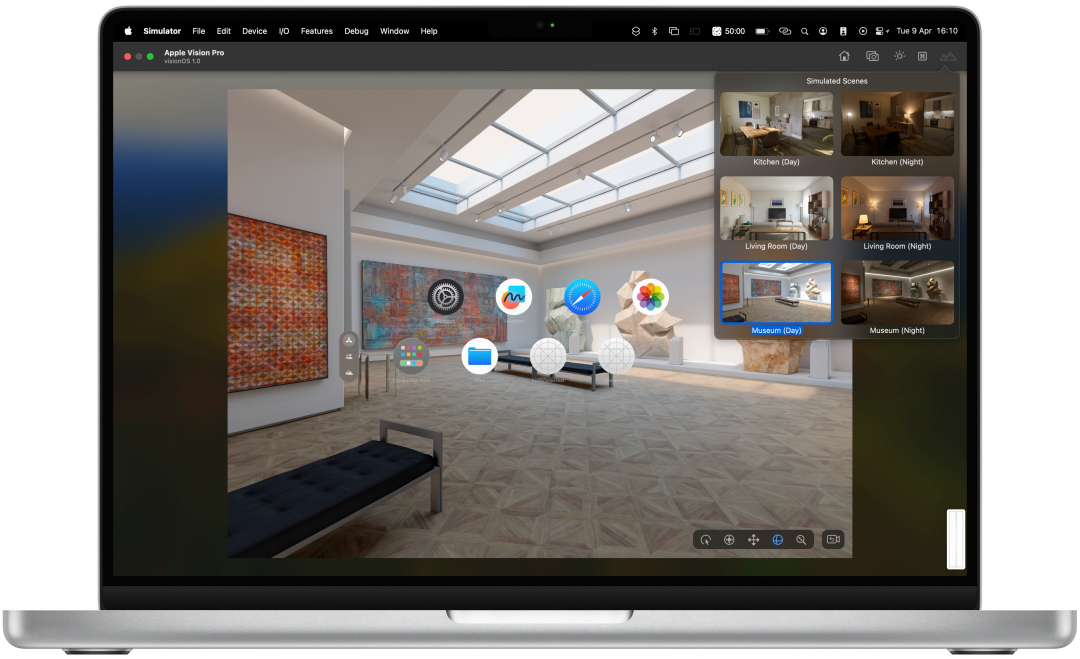


Slika 4.11. Prikaz aplikacije Xcode Instruments na uređaju Apple MacBook

4.2.11. Vision Pro Simulator

Kao dio programskog okruženja Xcode uključen je i Vision Pro Simulator, koji omogućuje simuliranje raznih Apple platformi, uključujući iOS, iPadOS, watchOS i tvOS. Xcode 15 dodaje mogućnost simuliranja sustava visionOS, olakšavajući razvojnim programerima pokretanje svoje aplikacije u tom okruženju bez posjedovanja stvarnog uređaja Vision Pro. Simulator podržava pregršt značajki namijenjenih olakšavanju razvojnog procesa aplikacije, poput snimanja zaslona, korištenje virtualne tipkovnice sustava ili stvarne tipkovnice računala, simulaciju biometrijske autentifikacije, simulaciju sustava plaćanja Apple Pay i raznih vizualnih postavki za testiranje animacija i sučelja. Simulator za sustav visionOS omogućuje

simulaciju šest različitih okruženja, uključujući okruženje dnevnog boravka, kuhinje i muzeja, vidljivo na slici 4.12.



Slika 4.12. Prikaz visionOS Simulatora na uređaju Apple MacBook

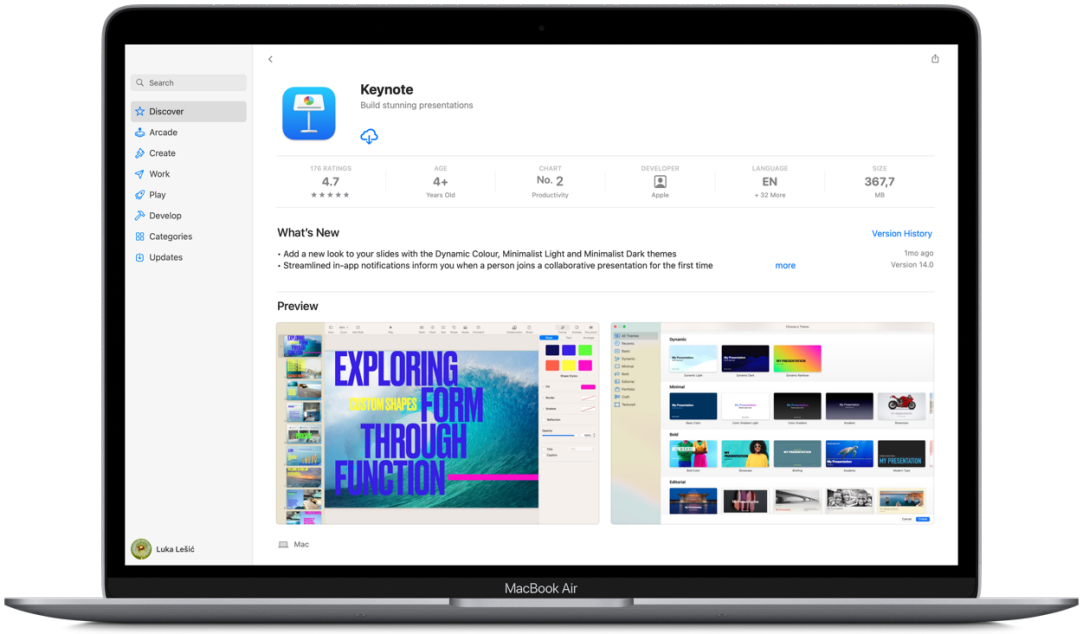
4.2.12. Distribucijska platforma App Store

Apple Vision Pro ima pristup posebnoj inačici trgovine aplikacijama Apple App Store. U vrijeme datuma puštanja uređaja u prodaju, bilo je dostupno preko 600 aplikacija optimiziranih za Apple Vision Pro, prema [32]. Osim toga, zbog kompatibilnosti s aplikacijama sustava iOS i iPadOS, korisnici imaju na raspolaganju preko milijun aplikacija za preuzimanje. Baš kao i na trgovini App Store u sustavima iOS i iPadOS, razvojni programeri mogu prodavati svoje aplikacije u preko 100 država, no one moraju proći rigorozan proces odobrenja Appleovog tima za App Store. Postoje stroge kontrole sigurnosti i korisničke privatnosti, te se provjerava sadrže li aplikacije zlonamjeran programski kod. Kako bi razvojni programer imao mogućnost distribucije svoje aplikacije u trgovini App Store, potrebno je stvoriti Apple Developer račun i platiti godišnju članarinu u vrijednosti 99 američkih dolara.

4.2.13. App Store Connect

App Store Connect je mrežna aplikacija za praćenje i upravljanje aplikacija objavljenih na trgovini App Store. Pruža značajku stvaranja naslovne stranice za svaku aplikaciju na trgovini App Store, uz dodavanje naslova, podnaslova, opisa, ključnih riječi i slika zaslona za razne uređaje, a podržava 40 jezika. Osim toga, podržava registriranje kupnja unutar aplikacije (eng.

In-App Purchases) kao i pretplati te postavljanje njihovih cijena i opisa. Korisnici mogu vidjeti popis svih kupnja unutar aplikacije kao i pretplata u posebnim odjeljcima na naslovnoj stranici svake aplikacije na usluzi App Store. Razvojni programeri mogu dodavati i informacije o svakom ažuriranju aplikacije, odnosno detalje o promjenama unutar svake inačice aplikacije. Prikaz sučelja trgovine App Store vidljiv je na slici 4.13.



Slika 4.13. Prikaz stranice App Store „Product Page“ na uređaju Apple MacBook

4.2.14. TestFlight

TestFlight je platforma uključena u App Store Connect namijenjena za testiranje aplikacija. Podržava aplikacije za visionOS, iOS, iPadOS, macOS, watchOS i tvOS. Moguće je pozvati do 10 000 korisnika za beta testiranje jedne aplikacije, uz automatska ažuriranja i period valjanosti od 90 dana po inačici aplikacije. Moguće je pozivati testere preko mail adrese ili javnog linka. Razvojni programer objavljuje inačicu aplikacije na App Store Connect te u odjeljku TestFlight za svaku novu inačicu može pisati detaljne informacije o inačici aplikacije. Korisnici jednostavno mogu dijeliti informacije s razvojnim programerima vezane za prijedloge, poboljšanja ili otkrivene probleme u aplikaciji.

4.3. Programska implementacija ključnih komponenata aplikacije

4.3.1. Model podataka

Pri izradi aplikacije bilo je ključno osmisliti model podataka koji će se prikazivati korisniku. Model proizvoda iz trgovine mora biti dovoljno općenit kako bi se mogao prilagoditi za

različite slučajeve uporabe, ali u isto vrijeme mora obavještavati ostale dijelove programskog koda o promjenama svog stanja i biti povezan s bazom podataka. Programski kod za model proizvoda prikazan je na slici 4.14.

```
@Observable
class ModelObject: Decodable {
    var name: String
    var modelURL: String?
    var price: Int?
    var parentCollection: String
    var description: String?
    var isFavourite: Bool

    init(name: String, modelURL: String, price: Int, parentCollection:
String, description: String, isFavourite: Bool) {
        self.name = name
        self.modelURL = modelURL
        self.price = price
        self.parentCollection = parentCollection
        self.description = description
        self.isFavourite = isFavourite
    }
}
```

Slika 4.14. Programski kod modela za proizvod

Model implementira protokol „Observation“, koji mu omogućava pohranu popisa objekata „promatrača“ koje može obavještavati o promjenama stanja. Ovo omogućuje slabije veze između objekata kao i istovremeno slanje obavijesti većem broju objekata.

4.3.2. Prilagodba modela bazi podataka

Korištenjem programskog obrasca „Adapter“, model proizvoda pripremljen je na rad s bazom podataka. Metoda „mapDocumentSnapshot()“ poziva metodu „adaptSnapshot()“ kojoj se predaje struktura podataka koja se iščitava iz baze podataka Firebase. Ova struktura osigurava jedinstven skup dokumenata s vlastitim identifikatorima i potrebnim metapodacima, te podržava ažuriranje informacija o modelima u stvarnom vremenu. Programski kod adaptera vidljiv je na slici 4.15.

```
class FirebaseAdapter: Adapter {
    func adaptSnapshot(_ snapshot: DocumentSnapshot) -> ModelObject {
        let name = snapshot.data()?["name"] as? String ?? ""
        let modelURL = snapshot.data()?["modelURL"] as? String ?? ""
        let description = snapshot.data()?["description"] as? String
let isFavourite = snapshot.data()?["isFavourite"] as? Bool ?? false
        let price = snapshot.data()?["price"] as? Int ?? 0
        let parentCollection = snapshot.reference.parent.collectionID
        let modelObject = ModelObject(name: name, modelURL: modelURL,
price: price, parentCollection: parentCollection, description:
description, isFavourite: isFavourite)
        return modelObject
    }

    func mapDocumentSnapshot(_ snapshot: [DocumentSnapshot]) ->
[ModelObject] {
        let modelObjects = snapshot.map {
            self.adaptSnapshot($0)
        }
        return modelObjects
    }
}
```

Slika 4.15. Programski kod adaptera za prilagodbu modela bazi podataka

4.3.1. Komunikacija s bazom podataka i rad s modelom

ViewModel za rad s objektima osigurava kako bi svaka korisnička interakcija u aplikaciji vezana uz proizvode obavila odgovarajuću funkcionalnost sa strane baze podataka. To uključuje slanje mrežnog poziva mrežnom repozitoriju za dohvaćanje omiljenih proizvoda, dohvaćanje kategorija i filtriranje proizvoda te dodavanje proizvoda u listu omiljenih proizvoda. Ovaj *ViewModel* ključan je za ispravno funkcioniranje aplikacije, te se ubacuje u sloj tipa *View* za prikaz proizvoda. Programski kod *ViewModela* za rukovanje proizvodima prikazan je na slici 4.16.

```
@Observable
class ObjectViewModel {
    var selectedCategory: ModelType
    let repository = ModelObjectsRepository()
    var filteredObjects: [ModelObject] = []
    var selectedObject: ModelObject?
    let database = Firestore.firestore()

    init(selectedCategory: ModelType) {
        self.selectedCategory = selectedCategory
        repository.fetchAllData()
    }

    //MARK: filtering data

    func fetchFavourites() {
        repository.fetchFavourites()
        DispatchQueue.main.asyncAfter(
            deadline: .now() + 2
        ) { [weak self] in
            self?.filteredObjects = (
                self?.repository.favouriteObjects
            )!
        }
    }

    func filterObjects() {
        let filteredDocuments = repository.modelObjects.filter { object
in
            let collectionName = object.parentCollection
            return collectionName == selectedCategory.databaseId
        }
      filteredObjects = filteredDocuments
    }

...
        }
    }
}
```

Slika 4.16. Programski kod ViewModela za upravljanje objektima

Metoda *addToFavourites*(), prikazana na slici 4.17, provjerava korisnikov identifikator, nakon čega pristupa odgovarajućoj zbirci u bazi podataka, pronalazi odabran objekt i mijenja mu status u omiljen tako što *Boolean* varijabli imena „isFavourite“ dodijeljuje odgovarajuću vrijednost. Promjene ovog tipa u stvarnom vremenu postaju vidljive unutar konzole Firebase.

```
    func addToFavourites(_ object: ModelObject) {

        guard let userId = Auth.auth().currentUser?.uid else {
            return
        }

        let collectionRef = database
            .collection(selectedCategory.databaseId)
            .whereField("name", isEqualTo: object.name)
            .getDocuments() {
            snapshot, error in
            if let error = error {
                print("Error fetching documents:
\(error.localizedDescription)")
            }
            else {
                let favouriteStatus = object.isFavourite

                object.isFavourite = !favouriteStatus
                let document = snapshot!.documents.first
                document?.reference.updateData(
                [
                    "isFavourite": !favouriteStatus,
                    "favorites.\(userId)": !favouriteStatus
                ])
            }
        }
    }
```

Slika 4.17. Programski kod funkcije za dodavanje modela u listu omiljenih

4.3.2. Sustav za autentifikaciju

Firebase omogućuje implementaciju sustava za prijavu i registraciju korisnika. Prvo je potrebno definirati strukturu podataka koja predstavlja korisnika s jedinstvenim identifikatorom. Prikaz strukture korisnika vidljiv je na slici 4.18.

```
struct User {
    let uid: String
    let email: String
}
```

Slika 4.18. Programski kod strukture koja predstavlja korisnika

ViewModel pod imenom „AuthenticationViewModel“, prikazan na slici 4.19 pohranjuje referencu na trenutno prijavljenog korisnika i pristupa bazi podataka kako bi uspješno autentificirao korisnika. Osim toga, mora dohvatiti sve potencijalne greške pri prijavi i prikaže ih korisniku na razumljiv način.

```
@Observable
class AuthenticationViewModel {
    var email: String = ""
    var password: String = ""
    private var _currentUser : User? = nil
    var hasError = false
    var errorMessage = ""
    var isLoggedIn = false

    private var handler = Auth.auth().addStateDidChangeListener{ _,_ in
}

    var currentUser: User {
        return _currentUser ?? User(uid: "", email: "")
    }

    init() {
        handler = Auth.auth().addStateDidChangeListener { auth, user in
            if let user = user {
                self._currentUser = User(uid: user.uid, email:
user.email!)
                self.isLoggedIn = true
            } else {
                self._currentUser = nil
                self.isLoggedIn = false
            }
        }
    }

    func signIn() async {
        hasError = false
        do {
            try await Auth.auth().signIn(withEmail: email, password:
password)
        } catch {
            hasError = true
            errorMessage = error.localizedDescription
        }
```

Slika 4.19. Programski kod ViewModela za autentifikaciju korisnika

Metoda *signUp*() poziva odgovarajuću funkcionalnost na bazi podataka za stvaranje novog korisnika, dok metoda *signOut*() prekida trenutnu korisničku sesiju. Programska implementacija navedene funkcionalnosti prikazana je na slici 4.20.

```
    func signUp() async {
        do {
            try await Auth.auth().createUser(withEmail: email,
password: password)
        } catch {
            hasError = true
            errorMessage = error.localizedDescription
            print(error.localizedDescription)
        }
    }

    func signOut() async {
        hasError = false
        do {
            try Auth.auth().signOut()

        } catch {
            hasError = true
            errorMessage = error.localizedDescription
        }
    }
```

Slika 4.20. Programski kod funkcija za prijavu i odjavu korisnika

4.3.3. Implementacija mrežnog repozitorija

Mrežni repozitorij sloj je aplikacije koji na strukturiran i siguran način omogućuje interakciju s udaljenim poslužiteljima. Implementirani repozitorij pri pokretanju aplikacije šalje API poziv bazi podataka Firebase za dohvaćanje odgovarajuće zbirke modela, te ih mapira u polje objekata za proizvode, kao što je vidljivo na slici 4.21. Instanca mrežnog repozitorija se zatim ubacuje u odgovarajući *ViewModel* koji upravlja preuzetim podacima. Repozitorij također prosljeđuje sve moguće greške koje se mogu pojaviti pri obradi podataka *ViewModel*u koji ga poziva.

```
final class ModelObjectsRepository: Repository {
    var loadingState = LoadingState.empty
    let database = Firestore.firestore()
    var modelObjects: [ModelObject] = []
    var favouriteObjects: [ModelObject] = []
    let adapter = FirebaseAdapter()

    func fetchAllData() {
        modelObjects = []
        loadingState = .loading
        for modelType in ModelType.allCases {
            let collectionRef =
database.collection(modelType.databaseId)
            collectionRef.getDocuments { snapshot, error in
                if let error = error {
                    print("Error fetching documents:
\(error.localizedDescription)")
                }
                else {
                    if let documents = snapshot?.documents {
                        self.mapSnapshotToModelObjects(snapshot:
documents)
                        self.loadingState = .loaded
                    }
                }
            }
        }
        fetchFavourites()
    }

    func mapSnapshotToModelObjects(snapshot: [DocumentSnapshot]) {
        self.modelObjects.append(contentsOf:
adapter.mapDocumentSnapshot(snapshot))
```

Slika 4.21. Implementacija mrežnog repozitorija

4.4. Implementacija sučelja aplikacije

Kako bi aplikacija pružala dobro iskustvo korisniku, mora na brz i vizualno ugodan način omogućiti pregledavanje i interakciju s trodimenzionalnim modelima proizvoda. Potrebno je implementirati funkcionalnost rotiranja i skaliranja proizvoda, kao i prikaz detalja o svakom proizvodu.

4.4.1. Prikazivanje trodimenzionalnih modela u aplikaciji

Kako bi se modeli ispravno učitali i prikazali korisniku, korištena je tzv. struktura „Model3D“ koja je dio Appleovog programskog paketa RealityKit. „Model3D“ omogućuje učitavanje USDZ tipa datoteka, te podržava razne modifikatore za prikaz kako bi se moglo upravljati veličinom, sjenama i izgledom objekta. Osim toga, podržava asinkrono učitavanje modela s

udaljenih mrežnih mjesta, kao što je u ovom slučaju poslužitelj Firebase. Prikaz implementacije strukture vidljiv je na slici 4.22.

```
    @ViewBuilder
    func modelView() -> some View {
        if let modelURL = object.modelURL,
             let url = URL(string: modelURL) {
            Model3D(url: url) { phase in
                switch phase {
                case .success(let model):
                    model
                        .resizable()
                        .scaledToFit()
                        .frame(maxWidth: 300, maxHeight: 300)
                case .failure(let error):
                    Text(error.localizedDescription)
                default:
                    ProgressView()
                        .frame(width: 300, height: 300)
                }
            }
            .animation(.easeInOut, value: isShowingFavouritesAlert)
            .animation(.easeInOut, value: isShowingCartAlert)
        }
    }
```

Slika 4.22. Programski kod za prikaz trodimenzionalnog modela

Prikazan programski kod implementira podršku za razna stanja objekta, pa se tako prikazuje indikator za učitavanje tijekom čekanja odgovora poslužitelja, poruka o pogrešci u slučaju manjka mrežne veze ili model promjenjive veličine u slučaju uspješnog spajanja na poslužitelj. Isti se element može iskoristiti na početnom zaslonu, gdje su svi proizvodi prikazani u rešetci promjenjive veličine i pojedinačno učitani. Na svaki proizvod dodan je element NavigationLink, kako bi se na odabir proizvoda moglo navigirati na prikaz detalja o proizvodu. SwiftUI podržava dodavanje modifikatora za pretraživanje i animacije direktno iz elementa *View*. Programski kod za prikaz proizvoda na početnom zaslonu naveden je na slici 4.23.

```
struct CategoryView: View {
    @Environment(ObjectViewModel.self) private var viewModel:
ObjectViewModel
    var columns: [GridItem] = [.init(.adaptive(minimum: 300), spacing:
30)]
    @Environment(ImmersiveViewManager.self) private var manager:
ImmersiveViewManager

    @State private var searchText = ""

    var body: some View {
        NavigationView {
            ScrollView {
                CustomGridLayout(searchResults, numberOfColumns: 3) {
object in
                    NavigationLink(
                        destination: ObjectDetailsView(
                            object: object
                        ).environment(
                            manager
                        ).environment(
                            viewModel
                        )
                    ) {
                            SingleObjectListView(object: object)
                        }
                            .buttonStyle(PlainButtonStyle())
                    }
            }
            .refreshable {
                viewModel.refresh()
            }
            .searchable(text: $searchText)
            .animation(.easeInOut, value: self.searchText)
            .navigationTitle("\(viewModel.selectedCategory)")
        }.navigationViewStyle(.stack)
    }
```

Slika 4.23. Programski kod prikaza proizvoda po kategorijama

4.4.2. Prikazivanje modela u stvarnom prostoru

U programskom kodu je implementirana mogućnost prikazivanja trodimenzionalnih modela u stvarnom prostoru. Kako bi modeli bili što bliži stvarnosti, dodana je funkcionalnost rotiranja kao i proizvoljne promjene veličine. Klikom na model u prostoru otvara se prozor s dodatnim informacijama o modelu. Koristi se slična logika prikazivanja modela preko strukture Model3D kao i u samom prozoru aplikacije. „ObjectView“ unutar kojeg je model implementiran je volumetrijski prozor dimenzija jednog metra u širinu, visinu i dubinu koji se može pomicati po volji u prostoru. U modifikator „SimultaneousGesture“ dodana je standardna gesta za povećavanje i smanjivanje elementa pomoću dva prsta. Parametar

„magnifyValue.magnification“ predstavlja trenutnu vrijednost skale elementa, te se na svaku korisničku interakciju množi s inicijalnom vrijednošću skale elementa i dodjeljuje se objektu u stvarnom vremenu. U isto vrijeme, gesta za rotiranje elementa funkcionira tako što se analiziraju vrijednosti x i y iz geste pomicanja prstiju u krug, te se zatim računa njihova euklidska udaljenost kojoj se pridodaje početna vrijednost kuta u stupnjevima. Na temelju dobivenih podataka se ažurira trenutna vrijednost rotacije objekta. Programska implementacija navedene funkcionalnosti prikazana je na slici 4.24.

```
private extension ObjectView {
    func modelView() -> some View {
        if let modelURL = self.activeObject?.modelURL, let url = URL(string: modelURL)
{
            Model3D(url: url) { phase in
                switch phase {
                case .success(let model):
                    model
                        .frame(maxWidth: 1500, maxHeight: 1500)
                        .rotation3DEffect(angle, axis: axis)
                        .scaleEffect(scale)
                        .onTapGesture {
                            withAnimation(.bouncy(extraBounce: 0.4)) {
                                isDisplayingInfoPopup.toggle()
                            }
                        }
                        .simultaneousGesture(MagnifyGesture()
                            .onChanged({ magnifyValue in
                                if let initialScale {
                                    scale = max(0.2, min(1.5,
magnifyValue.magnification * initialScale))
                                }
                                else {
                                    initialScale = scale
                                }
                            })
                                .onEnded({ magnifyValue in
                                    initialScale = scale
                                    SoundManager.shared.playSound(soundName: "correct")
                                }))
                        .simultaneousGesture(DragGesture()
                            .onChanged({ dragValue in
                                if let initialAngle, let initialAxis {
                                    let _angle = sqrt(pow(dragValue.translation.width,
2) + pow(dragValue.translation.height, 2)) + initialAngle.degrees
                                    let XAxis = ((-dragValue.translation.height +
initialAxis.0) / CGFloat(_angle))
                                    let YAxis = ((dragValue.translation.width +
initialAxis.1) / CGFloat(_angle))
                                    angle = Angle(degrees: Double(_angle))
                                    axis = (XAxis, YAxis, 0)
                                } else {
                                    initialAxis = axis
                                    initialAngle = angle
                                }
                            }).onEnded({ dragValue in
                                initialAxis = axis
                                initialAngle = angle
                            }))
```

Slika 4.24. Programski kod za prikaz modela u stvarnom svijetu

5. KORIŠTENJE I ISPITIVANJE APLIKACIJE

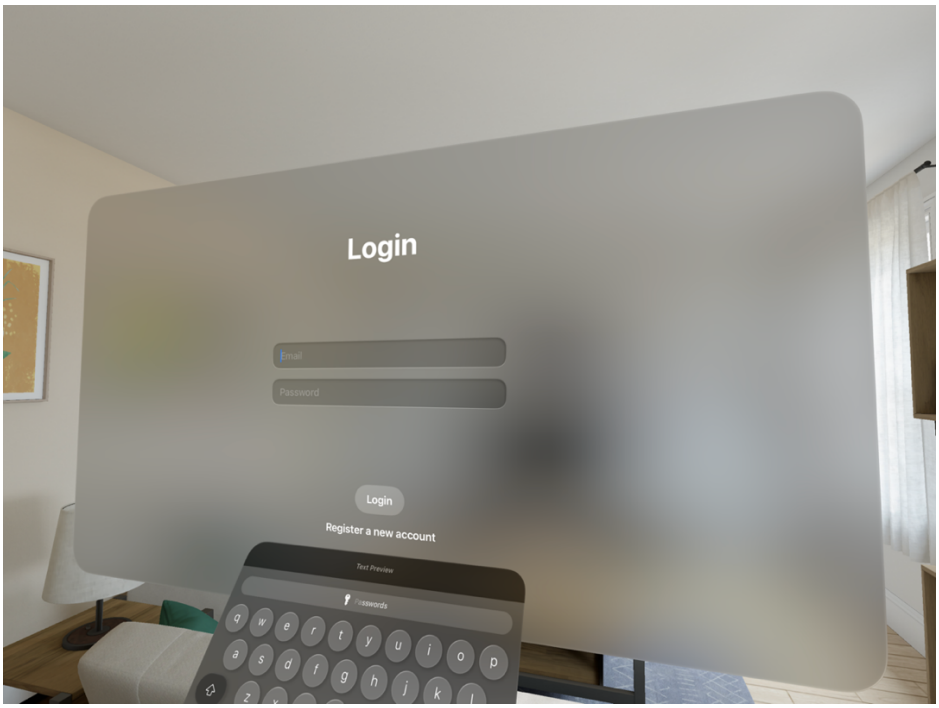
Poglavlje 5 donosi detaljan pregled korisničkog sučelja aplikacije, pružajući temeljit opis i analizu svih ekrana. Objašnjava se proces simulacije narudžbe proizvoda te se prikazuju jedinični testovi programskog koda skupa s rezultatima testiranja.

5.1. Prikaz korištenja aplikacije

U ovom poglavlju je kroz razne slučajeve uporabe prikazano sučelje kao i dizajn aplikacije te mogućnost postavljanja i interakcije s 3D modelima u stvarnom prostoru. Uz to su opisani i navedeni jedinični testovi programskog koda (eng. *Unit tests*) koji su izvršeni kako bi se osigurala sigurnost i stabilnost aplikacije.

5.1.1. Prijava i registracija u aplikaciju

Nakon prvog otvaranja aplikacije, prikazuje se sučelje za prijavu u korisnički račun, prikazano na slici 5.1. U slučaju da korisnik nema račun, klikom na odgovarajući gumb može pristupiti stranici za registraciju. Korisnički račun je nužan iz razloga što je lista želja povezana s bazom podataka i personalizirana, tako da podaci ostaju pohranjeni i nakon odjave korisnika. Nakon prve prijave nije potrebno ponovno se prijavljivati pri kasnijim pokretanjima aplikacije.



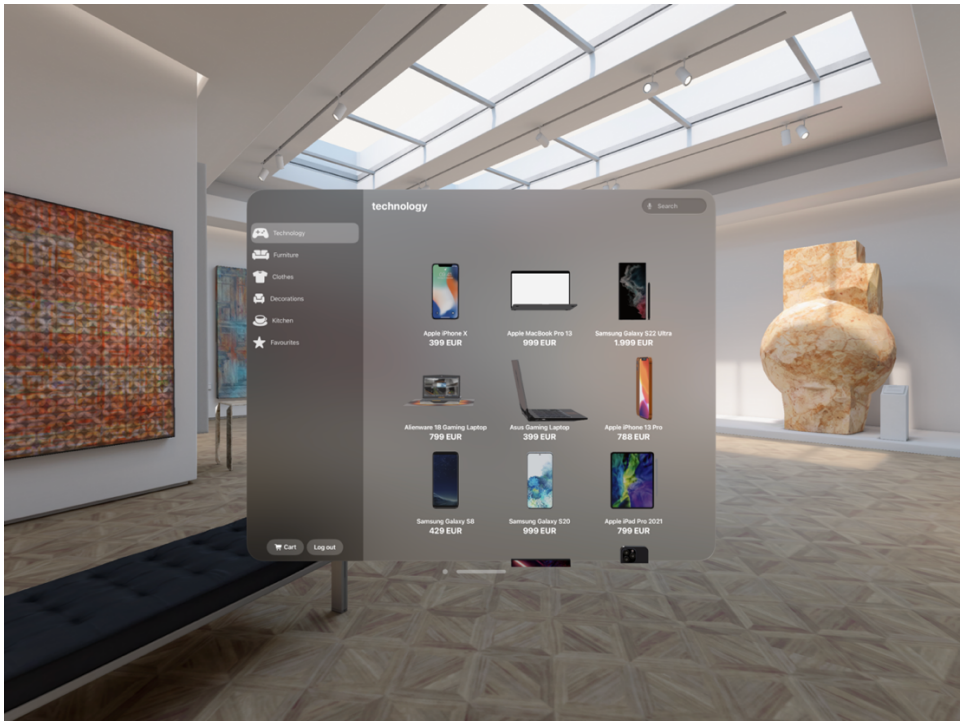
Slika 5.1. Prikaz početne stranice za prijavu u aplikaciji Showroom

Korisnik može koristiti sustavsku tipkovnicu za unos podataka ili može povezati fizičku tipkovnicu pomoću Bluetooth tehnologije. Ugrađene su provjere ispravnosti korisničkog imena i lozinke, te nije moguće registrirati isti račun više od jednog puta.

5.1.2. Početni zaslon

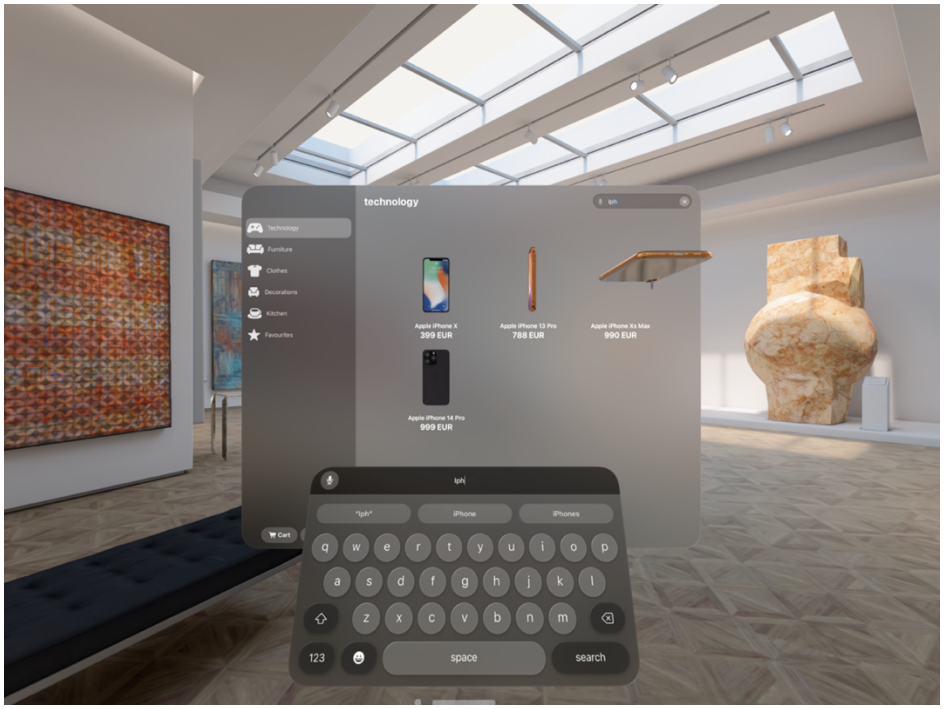
Nakon uspješne prijave u račun, korisniku se prikazuje početni zaslon aplikacije, koji je vidljiv na slici 5.2. Na raspolaganje je za odabrati različite preddefinirane kategorije proizvoda, te se nakon odabiranja jedne od opcija iz izbornika počinju učitavati proizvodi iz odabrane kategorije. U slučaju da nijedna kategorija nije odabrana, korisnik dobiva uputu kako može odabrati jednu iz izbornika.

Mreža proizvoda prilagodljiva je s obzirom na veličinu prozora, te se proizvodi učitavaju po potrebi, odnosno u obziru na trenutnu „*scroll*“ poziciju na zaslonu.



Slika 5.2. Prikaz popisa proizvoda u aplikaciji Showroom

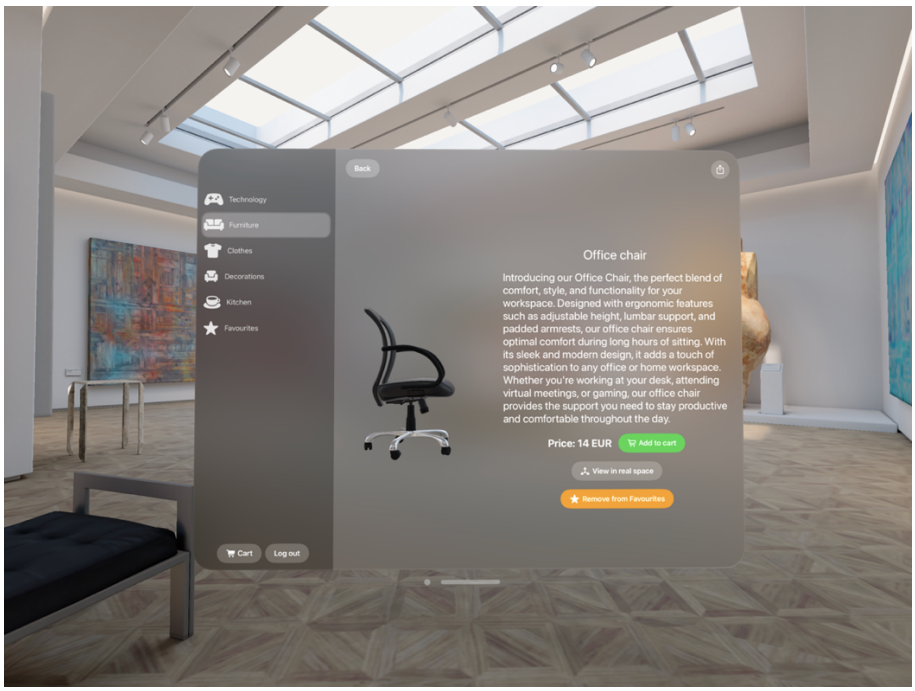
Odabirom na polje za pretraživanje proizvoda prikazuje se sustavska tipkovnica, te je moguće pretraživati proizvode po imenu ili kategoriji u kojoj se nalaze. Podržan je unos teksta i preko vanjskih tipkovnica. Nakon upisivanja željenog proizvoda, pojavljuju se poveznice za stranicu o detaljima proizvoda. Opisana funkcionalnost prikazana je na slici 5.3.



Slika 5.3. Prikaz stranice za pretragu u aplikaciji Showroom

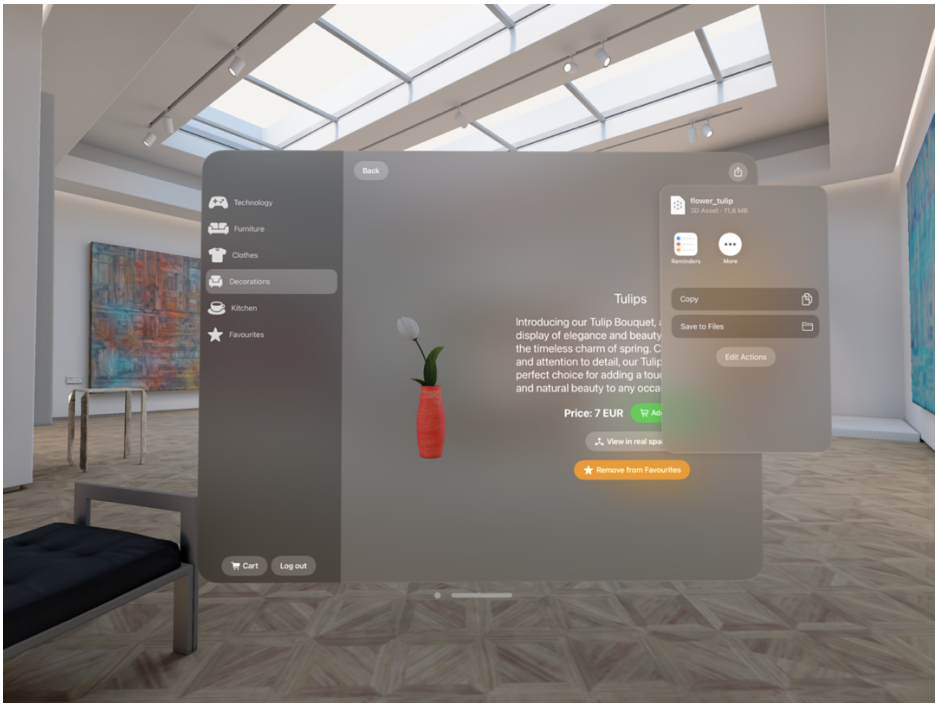
5.1.3. Prikaz detalja i dijeljenje proizvoda

Nakon što odabere željeni proizvod, prikazuje se stranica s detaljima proizvoda. Na ovoj se stranici nalazi detaljan opis proizvoda, cijena te gumbi za dijeljenje, dodavanje u košaricu, dodavanje ili uklanjanje proizvoda iz liste favorita te gumb za prikaz proizvoda u stvarnom okruženju. Stranica s detaljima proizvoda prikazana je na slici 5.4.



Slika 5.4. Prikaz stranice s detaljima o proizvodu u aplikaciji Showroom

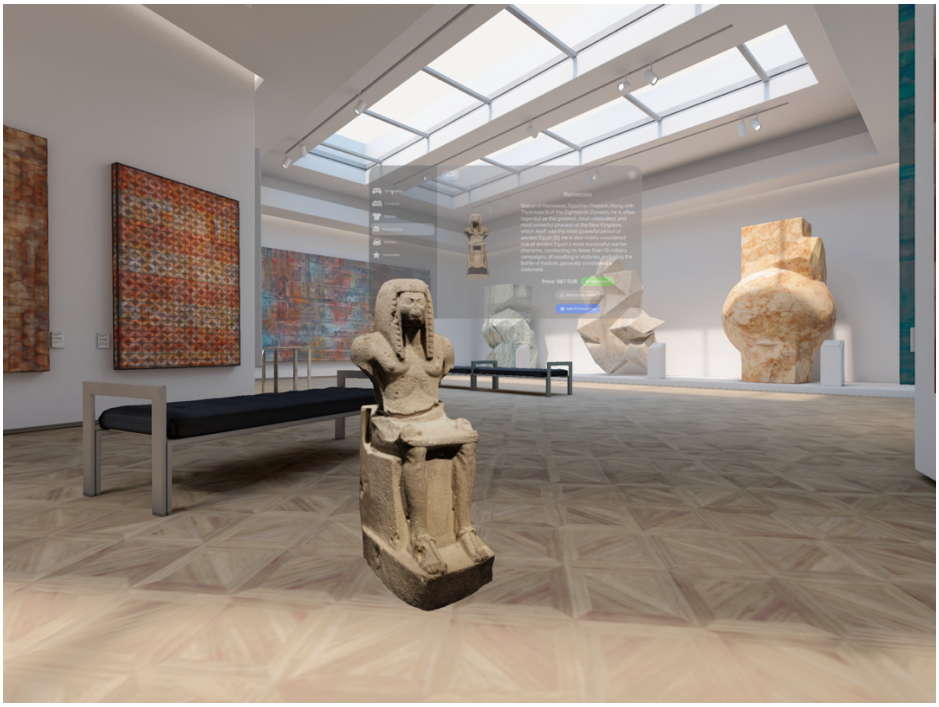
Odabirom gumba za dijeljenje korisnik dobiva okno za dijeljenje s popisom aplikacija i osoba, te može generirati opis i poveznicu na model proizvoda. Sustav visionOS automatski prepoznaje trodimenzionalan model proizvoda i predlaže popis aplikacija koje podržavaju prikaz istih. Prikaz implementiranog okna za dijeljenje proizvoda vidljiv je na slici 5.5.



Slika 5.5. Prikaz okna za dijeljenje proizvoda u aplikaciji Showroom

5.1.4. Prikaz proizvoda u stvarnom prostoru

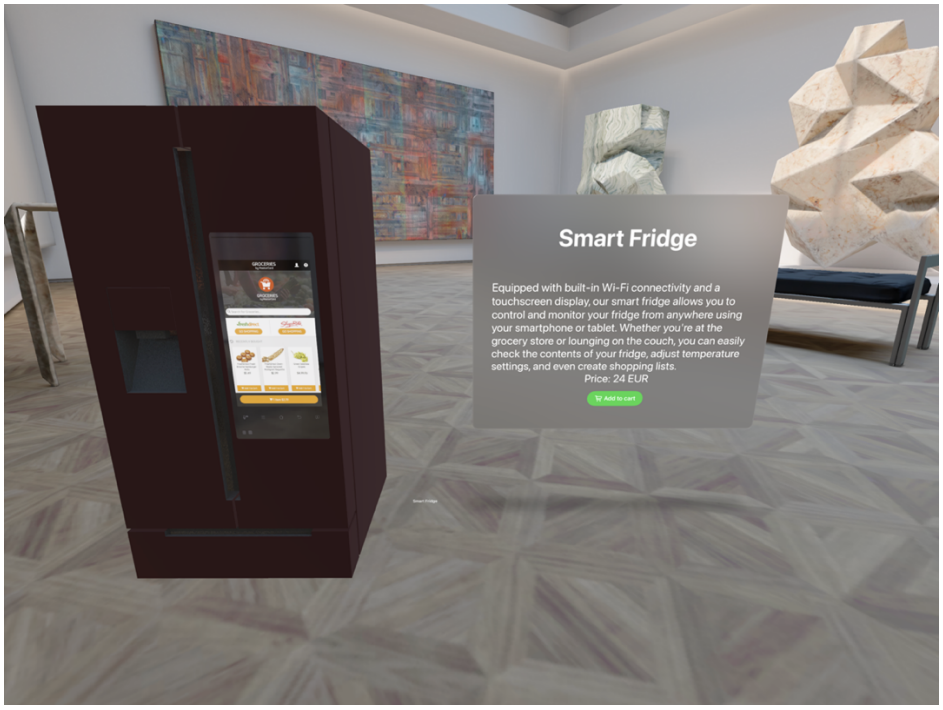
Klikom na gumb za prikaz proizvoda u stvarnom prostoru, prozor aplikacije izblijedi, te se učitava novi model proizvoda prilagođen za prikaz u prostoriji. Model se može rotirati i skalirati po želji, te ima podršku za sjene i odsjaje vidljive u prostoriji. Model također podržava slobodno pomicanje unutar prostorije i dinamičko skaliranje ovisno o svojoj udaljenosti od korisnika, kako bi uvijek davao dojam stalne veličine. Primjeri postavljanja trodimenzionalnih objekata u okoline kuhinje i galerije vidljivi su na slici 5.6 i slici 5.7.



Slika 5.6. Prikaz postavljanja 3D modela ukrasa u galeriji u aplikaciji Showroom

Slika 5.7. Prikaz postavljanja 3D modela hladnjaka u kuhinji u aplikaciji Showroom

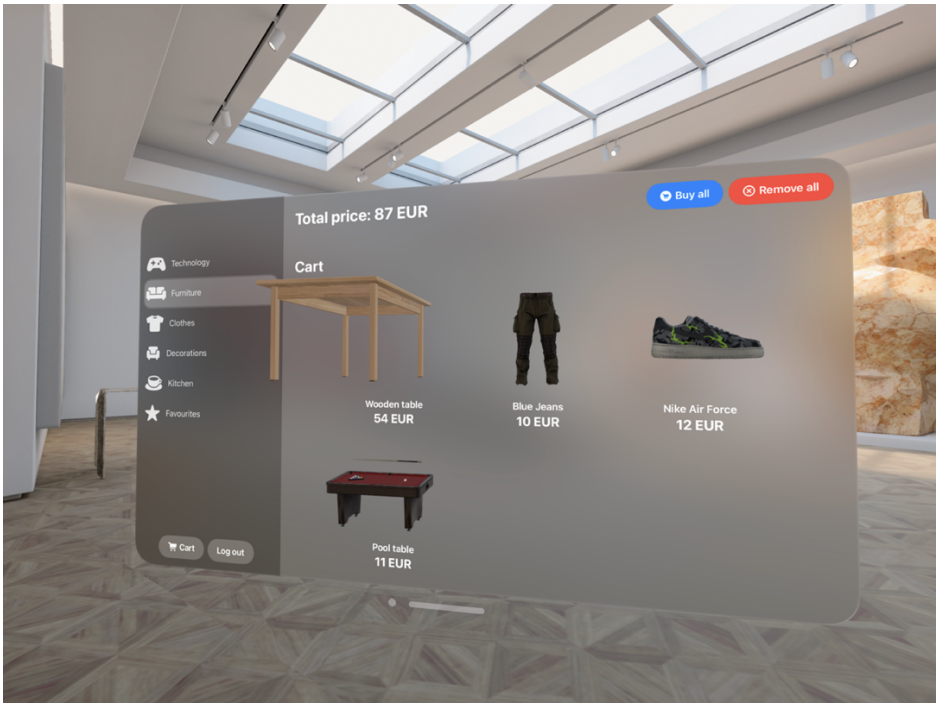
Klikom na proizvod on se pomakne u stranu, a zatim se otvara novi prozor koji sadrži opis proizvoda, cijenu i gumb za dodavanje u košaricu, kao što je prikazano na slici 5.8.



Slika 5.8. Prikaz interakcije s 3D modelom hladnjaka u aplikaciji Showroom

5.1.5. Košarica

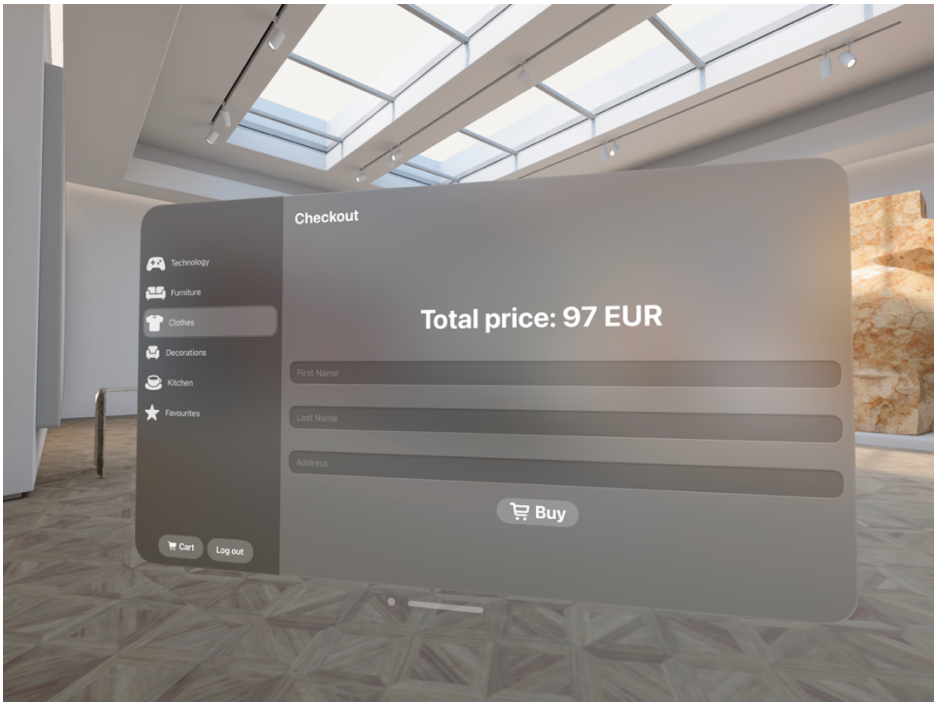
Aplikacija podržava dodavanje proizvoda u košaricu odabirom odgovarajućeg gumba. Košarica također podržava dodavanje veće količine istog proizvoda. U slučaju da je prazna, prikazuje se odgovarajuća poruka korisniku. Na vrhu stranice nalazi se ukupna cijena svih dodanih proizvoda te gumb za uklanjanje svih proizvoda kao i za kupnju proizvoda. Prikaz košarice vidljiv je na slici 5.9.



Slika 5.9. Prikaz popunjene košarice u aplikaciji Showroom

5.1.6. Završni korak kupnje

Nakon što korisnik odluči dovršiti kupnju i odabere odgovarajući gumb unutar stranice za košaricu, prikazuje mu se konačna stranica s ukupnom cijenom te poljima za unos punog imena i prezimena, kao i adrese, vidljivo na slici 5.10.



Slika 5.10. Prikaz završnog koraka kupnje u aplikaciji Showroom

Nakon odabira gumba za kupnju, korisniku se prikazuje skočni prozor s potvrdom uspješne kupnje, te se proces kupovine smatra završenim. Korisnik može navigirati nazad na popis proizvoda i ponovno pregledavati druge proizvode. Prikaz opisane funkcionalnosti dovršavanja kupnje vidljiv je na slici 5.11.

Slika 5.11. Prikaz uspješne kupnje proizvoda u aplikaciji Showroom

5.2. Postavke ispitivanja aplikacije

Korištenjem razvojnog okvira XCTest provodi se jedinično testiranje (eng. *Unit testing*) glavne funkcionalnosti aplikacije. Aplikacija ima jasno razdvojene klase od kojih svaka ima svoju ulogu, poput dohvaćanja podataka ili upravljanja s postojećim podacima. Kao dio ovog rada testiraju se funkcionalnosti upravljanja i učitavanja proizvoda, kao i funkcionalnost košarice. Konkretno, pišu se jedinični testovi za klase „ObjectsViewModel“ i „CartManager“. U aplikaciji se glavnina logike za upravljanje proizvodima nalazi u klasi „ObjectsViewModel“, koja filtrira proizvode, kategorizira ih, te koristi instancu mrežnog repozitorija „ModelObjectsRepository“ kako bi se pozvali mrežni pozivi za dohvaćanje podataka. Tako je za uspješan proces testiranja potrebno stvoriti simulaciju mrežnog repozitorija, te ju pomoću protokola „Dependency Injection“ ubaciti u *ViewModel* koji se testira. Prije svega je potrebno izvesti protokol, što je ekvivalent sučelja u Swift programskom jeziku, koji propisuje općenite funkcionalnosti mrežnog repozitorija. Zatim se u konstruktor *ViewModela* dodaje mogućnost ubacivanja instance protokola mrežnog repozitorija, umjesto konkretne instance neke klase.

Ovime smo omogućili stvarnom *ViewModelu* da po potrebi rukuje stvarnim mrežnim repozitorijem, ali i maketom mrežnog repozitorija koji je implementiran za potrebe testiranja.

Za ispravno testiranje *ViewModela* potrebno je napraviti funkcionalnu maketu mrežnog repozitorija. Maketa implementira protokol zvan *ModelObjectsRepositoryProtocol,* koji je prikazan na slici 5.12. Programski kod makete mrežnog repozitorija vidljiv je na slici 5.13.

```
protocol ModelObjectsRepositoryProtocol {
    var loadingState: LoadingState { get set }
    var modelObjects: [ModelObject] { get set }
    var favouriteObjects: [ModelObject] { get set }
    func fetchAllData()
    func fetchFavourites()
    func mapSnapshotToModelObjects(snapshot: [DocumentSnapshot])
    func mapFavourites(snapshot: [DocumentSnapshot])
}
```

Slika 5.12. Programski kod protokola za mrežni repozitorij

```
class MockModelObjectsRepository: ModelObjectsRepositoryProtocol {
    var mockModelObjects: [ModelObject] = []
    var mockFavouriteObjects: [ModelObject] = []
    var mockLoadingState = LoadingState.empty
    let adapter = FirebaseAdapter()
    func fetchAllData() {
        mockLoadingState = .loading
        modelObjects = [
            ModelObject(name: "MockObject1", modelURL: "", price: 10,
parentCollection: "ClothesModels", description: "", isFavourite: true),

            ModelObject(name: "MockObject2", modelURL: "", price: 20,
parentCollection: "TechnologyModels", description: "", isFavourite:
false)
        ]
        mockLoadingState = .loaded
    }

    func simulateLoadingError() {
        mockLoadingState = .error
    }

    func fetchFavourites() {
        favouriteObjects = mockModelObjects.filter { $0.isFavourite }
    }

    func mapSnapshotToModelObjects(snapshot: [DocumentSnapshot]) {
        self.mockModelObjects.append(contentsOf:
adapter.mapDocumentSnapshot(snapshot))
    }

    func mapFavourites(snapshot: [DocumentSnapshot]) {
        self.mockFavouriteObjects.append(contentsOf:
adapter.mapDocumentSnapshot(snapshot))
    } }
}
```

Slika 5.13. Prikaz programskog koda makete mrežnog repozitorija

Nakon uspostavljanja makete repozitorija, stvara se testna klasa za testiranje funkcionalnosti klase „ObjectViewModel“, kako bi se spoznalo upravlja li *ViewModel* objektima na ispravan i siguran način. Programski kod uspostavlja instancu *ViewModela* za upravljanje proizvoda, te ubacuje maketu mrežnog repozitorija u njega. Metoda *setUp()* pokreće se prije svake testne metode, a metoda *tearDown()* nakon svake testne metode. U testnu klasu se dodaju testovi funkcionalnosti kroz testne metode. Testna klasa je vidljiva na slici 5.14.

```
import XCTest
@testable import Showroom

class ObjectViewModelTests: XCTestCase {

    var viewModel: ObjectViewModel!
    var mockRepository: MockModelObjectsRepository!

    override func setUp() {
        super.setUp()
        mockRepository = MockModelObjectsRepository()
        viewModel = ObjectViewModel(selectedCategory: .clothes,
repository: mockRepository)
    }

    override func tearDown() {
        viewModel = nil
        mockRepository = nil
        super.tearDown()
    }
}
```

Slika 5.14. Implementacija testne klase za ViewModel proizvoda

```
    func testFetchFavourites() {
        // Given
        mockRepository.fetchAllData()
        // When
        viewModel.fetchFavourites()

        // Then
        XCTAssertEqual(self
            .viewModel.repository.favouriteObjects.count, 1)

            XCTAssertEqual(self
                .viewModel
                .repository
                .favouriteObjects
                .first?
                .name, "MockObject1")
    }
```

Slika 5.15. Testna metoda za provjeru liste favorita

Slika 5.15 prikazuje testnu metodu za provjeru popunjavanja liste favorita. Kako bi razvojno okruženje uspješno prepoznalo testnu metodu, njen naziv uvijek mora započinjati prefiksom „test“. Prije pokretanja testa, u metodi *fetchAllData()* se dodaje maketa proizvoda u popis omiljenih. Testna metoda *testFetchFavourites()* funkcionira tako što se poziva programski kod za simuliranje dohvaćanja podataka mrežnog repozitorija, filtrira ih u favorite te provjerava jesu li točni objekti proizvoda dodani u listu favorita i filtrirani.

```
func testFilterObjects() {
        // Given
        viewModel.selectedCategory = .technology
        mockRepository.fetchAllData()

        // When
        viewModel.filterObjects()

        // Then
        XCTAssertEqual(viewModel.filteredObjects.count, 1)
        XCTAssertEqual(viewModel.filteredObjects.first?.name,
"MockObject2")
    }
```

Slika 5.16. Testna metoda za provjeru funkcionalnosti filtriranja proizvoda

Slika 5.16 prikazuje testnu metodu *testFilterObjects() koja* provjerava filtrira li ViewModel proizvode po kategorijama. To radi pozivanjem metode *filterObjects()* i provjeravajući količinu filtriranih objekata i usporedbu varijable imena objekta s unaprijed poznatim imenom.

```
    func testAddToFavourites() {
        // Given
        mockRepository.fetchAllData()

        let object = mockRepository.modelObjects.first!

        viewModel.selectedCategory = .clothes
        mockRepository.modelObjects = [object]
        viewModel.filterObjects()
        // When
        viewModel.addToFavourites(object)
        sleep(5)
        // Then
        XCTAssertEqual(object.isFavourite, true)
    }
```

Slika 5.17. Testna metoda za dodavanje proizvoda u listu favorita

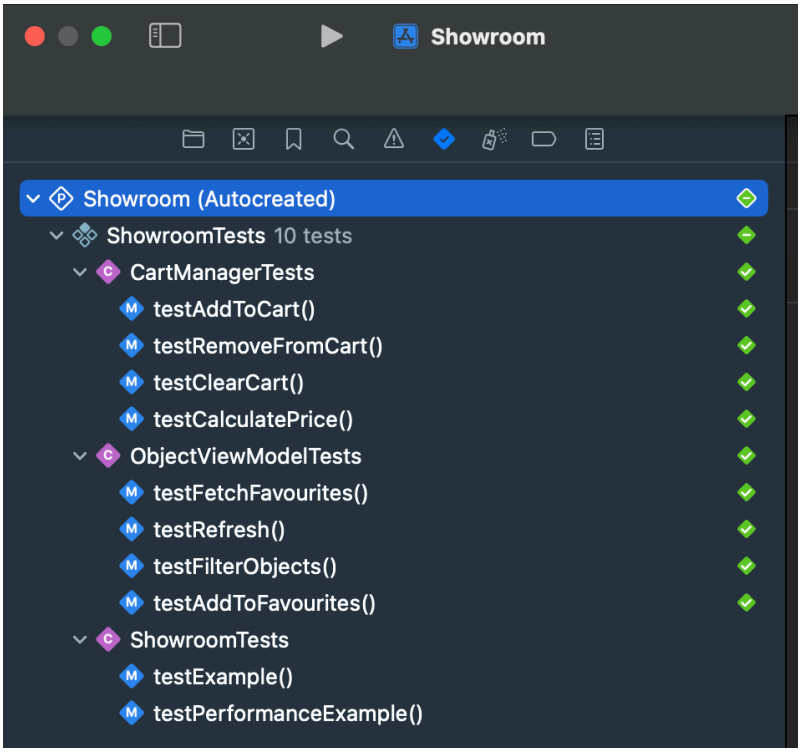
Metoda *testAddToFavourites(),* vidljiva na slici 5.17 instancira jedan proizvod, dodjeljuje mu kategoriju, te ga dodaje u listu favorita. Zatim provjerava njegov status svojstva omiljenosti. Na sličan način može se uspostaviti klasa za testiranje funkcionalnosti košarice, kako bi se dobio uvid u ponašanje „CartManager“ klase.

```
class CartManagerTests: XCTestCase {

    var cartManager: CartManager!
    var mockObject1: ModelObject!
    var mockObject2: ModelObject!

    override func setUp() {
        super.setUp()
        cartManager = CartManager.shared
        mockObject1 = ModelObject(name: "Object1", modelURL: "", price:
10, parentCollection: "Category1", description: "", isFavourite: false)
        mockObject2 = ModelObject(name: "Object2", modelURL: "", price:
20, parentCollection: "Category2", description: "", isFavourite: true)
    }

    override func tearDown() {
        cartManager = nil
        mockObject1 = nil
        mockObject2 = nil
        super.tearDown()
    }

    func testAddToCart() {
        cartManager.addToCart(object: mockObject1)

        XCTAssertEqual(cartManager.totalPrice, 10)
    }

    func testRemoveFromCart() {
        cartManager.addToCart(object: mockObject1)
        cartManager.addToCart(object: mockObject2)

        cartManager.removeFromCart(name: "MockObject1")

        XCTAssertEqual(cartManager.items.count, 1)
        XCTAssertEqual(cartManager.totalPrice, 20)
    }
  }
```

Slika 5.18. Testna klasa za provjeru funkcionalnosti košarice

Testovi navedeni na slici 5.18 provjeravaju funkcionalnost dodavanja i uklanjanja proizvoda iz košarice. Testovi instanciraju dvije makete proizvoda, te pozivaju metodu *addToCart()* iz *ViewModela*, a zatim provjeravaju odgovara li ukupna cijena proizvoda u košarici zadanoj. Metoda *testRemoveFromCart()* dodaje dva proizvoda u košaricu i zatim na sličan način usporedbe ukupne cijene proizvoda u košarici provjerava je li proizvod uklonjen iz košarice.

Nakon pokretanja svih testova, vidljivo je kako su svi testovi uspješno prošli. Slika 5.19 i slika 5.20 prikazuju informacije o uspješnosti pokrenutih testova.



Slika 5.19. Prvi rezultati pokretanja testova

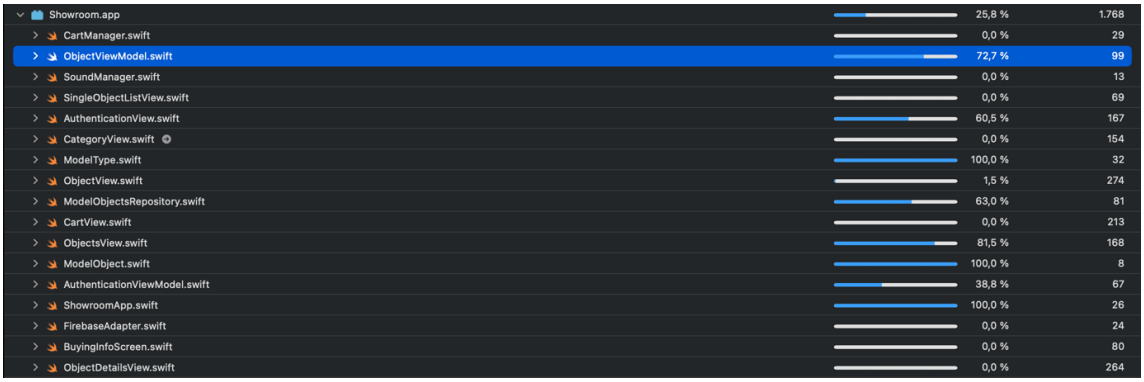
```
Test Suite 'ObjectViewModelTests' started at 2024-09-10 17:06:09.284.
Test Case '-[ShowroomTests.ObjectViewModelTests testAddToFavourites]' started.
Test Case '-[ShowroomTests.ObjectViewModelTests testAddToFavourites]' passed (5.004 seconds).
Test Case '-[ShowroomTests.ObjectViewModelTests testFetchFavourites]' started.
Test Case '-[ShowroomTests.ObjectViewModelTests testFetchFavourites]' passed (0.002 seconds).
Test Case '-[ShowroomTests.ObjectViewModelTests testFilterObjects]' started.
Test Case '-[ShowroomTests.ObjectViewModelTests testFilterObjects]' passed (0.001 seconds).
Test Case '-[ShowroomTests.ObjectViewModelTests testRefresh]' started.
Test Case '-[ShowroomTests.ObjectViewModelTests testRefresh]' passed (4.003 seconds).
Test Suite 'ObjectViewModelTests' passed at 2024-09-10 17:06:18.297.
     Executed 4 tests, with 0 failures (0 unexpected) in 9.010 (9.013) seconds
```

Slika 5.20. Informacije o uspješnosti pokretanja testova

5.3. Rezultati ispitivanja s analizom

Glavni cilj testiranja bio je otkriti rukuje li aplikacija preuzimanjem i prikazivanjem proizvoda, kao i funkcionalnosti košarice na ispravan način, stoga je bilo nužno pokriti što veću količinu programskog koda u datotekama *ObjectsViewModel* kao i *CartManager*. Xcode sadrži funkcionalnost provjere pokrivenosti programskog koda jediničnim testovima preko značajke *Report Navigator*, čije je sučelje prikazano na slici 5.21. Jedno od ograničenja pokretanja jediničnih testova u okruženju Xcode je nemogućnost slijednog pokretanja testova različitih testnih klasa, stoga su testovi pokrenuti kroz dvije iteracije. Slika 5.21 prikazuje kako je pokrivenost programskog koda u datoteci ObjectViewModel približno 73% uz stopostotnu

pokrivenost testiranja programskog koda modela objekata, dok slika 5.22 prikazuje kako je pokrivenost testiranog programskog koda u datoteci CartManager približno 93%. Ostale datoteke nisu izravno testirane, no dio njih je također pokriven radi dijeljenog programskog koda, inicijalizacije i ovisnosti objekata jednog o drugom. Zaključuje se kako su svi testovi programskog koda sadržanog u ove dvije datoteke prošli uspješno, te aplikacija nudi stabilno korisničko iskustvo pri upravljanju objektima i kupnji. Kako bi se povećala ukupna pokrivenost testiranog programskog koda, moguće je proširiti testnu grupu dodavanjem testnih klasa koje sadrže pripadajuće jedinične testove, primjerice testove za autentifikaciju ili zvukovni sustav.



Slika 5.21. Rezultati pokretanja testova za upravljanje modelima proizvoda

Slika 5.22. Rezultati pokretanja testova košarice

S obzirom na to da u trenutku pisanja rada aplikacija nije objavljena na trgovinu App Store, jedini način distribucije je spajanje fizičkog uređaja Vision Pro na računalo i ručno dodavanje datoteke s nastavkom *.app*. Kako se u simulatoru uređaja Vision Pro ne mogu dodavati proizvoljne pozadine, aplikacija bi trebala biti testirana na fizičkom uređaju u različitim okruženjima, za razliku od predodređenih okruženja u kojima su testovi sprovedeni. Također, iako se u okruženju Xcode mogu iščitati informacije o iskorištenosti procesorskih resursa i radne memorije simuliranog uređaja, bez stvarnog uređaja Vision Pro nije sasvim moguće

ispitati ima li aplikacija nekih neželjenih pojava poput pregrijavanja uređaja ili ispada radi velike potrošnje radne memorije. Interakcije s trodimenzionalnim proizvodima izvršene su u simulatoru korištenjem dodirne plohe prijenosnog računala, te bi pri nabavi uređaja bilo potrebno testirati implementirane geste ruku za rotaciju i skaliranje, te translaciju objekata. Aplikacija strogo prati Appleove smjernice izgradnje korisničkog sučelja, pa je programski kod posebno usmjeren prema višenitnosti i stabilnosti, te odvajanje pozadinskih procesa od procesa vezanih uz prikaz i obradu komponenata korisničkog sučelja, kao i na strogo upravljanje memorijom. Na temelju toga, može se pretpostaviti da se korisničko iskustvo u stvarnoj okolini ne bi previše razlikovalo od ispitne simulacijske okoline. Korisnik koji posjeduje stvarni uređaj Apple Vision Pro može koristiti sustavske geste za interakciju s aplikacijom i vidjeti aplikaciju u okruženjima po želji. Korisnik može koristiti funkcionalnost dijeljenja proizvoda, te dodavanja proizvoda u stvarno okruženje, kao i funkcionalnost rotiranja i pomicanja. Korisniku je potrebna stabilna mrežna veza kao i pristup računalu MacBook za ručno dodavanje aplikacije na uređaj jer ju u trenutku pisanja rada ne može preuzeti s trgovine App Store. Osim toga, bi aplikacija bila povezana na bazu podataka stvarne trgovine kao i stvarni sustav narudžbi, korisnici bi nesmetano mogli naručivati proizvode. U trenutku pisanja rada mogu se samo izvršiti simulacije narudžbi.

6. ZAKLJUČAK

Aplikacija „Showroom“ razvijena u sklopu ovog diplomskog rada pruža brz i intuitivan način pregledavanja virtualnih proizvoda u stvarnom prostoru. Uporabom programskog jezika Swift i razvojnog okvira SwiftUI razvijeno je korisničko sučelje koje je jednostavno za uporabu, a istovremeno omogućuje složene interakcije s trodimenzionalnim virtualnim modelima. U radu je detaljno opisano sklopovlje uređaja Vision Pro, te operacijski sustav visionOS, a dan je i pregled novih značajki i mogućnosti Apple platformi u kontekstu mješovite stvarnosti. Opisan je postupak testiranja aplikacije korištenjem jediničnih testova programskog koda, te su demonstrirani scenariji korištenja aplikacije.

Rezultati testiranja pokazuju da aplikacija ispunjava svoju svrhu i pruža funkcionalnu i stabilnu maketu mrežne trgovine, uz uspješno prikazivanje trodimenzionalnih modela proizvoda u formatu USDZ, kao i pretraživanje, filtriranje i kategoriziranje proizvoda. Postupak testiranja ističe prednosti datoteka tipa USDZ, poput brzog učitavanja, lake integracije s razvojnim okvirom Apple RealityKit i podrške za asinkrono dohvaćanje s udaljenog poslužitelja. Iako je jedan od nedostataka uporabe ovog tipa podataka potencijalno veliko zauzeće memorije i prostora za pohranu podataka, uz implementaciju asinkronog učitavanja modela s poslužitelja i automatskog oslobađanja radne memorije, aplikacija postiže punu funkcionalnost bez da narušava performanse virtualnog testnog uređaja. U radu se prikazuju i nove tehnologije dodane u programski jezik Swift i razvojni okvir SwiftUI, te istražuje postupak implementacije višedimenzionalnih vizualnih elemenata u korisničko sučelje. Iako je aplikacija stabilna i blaga prema resursima uređaja, otkriveno je da ipak nedostaje podrška za razne biblioteke koje su prisutne u okruženjima iOS i iPadOS, pa uvijek postoji pregršt operativnih pogrešaka u razvojnom okviru SwiftUI pri izradi aplikacija za mješovitu stvarnost. Također, otkrivena su ograničenja u simulatoru uređaja Vision Pro. Simulator je ograničen na prikaz nekoliko predodređenih pozadinskih okruženja i ne podržava sve funkcionalnosti uređaja Apple Vision Pro, poput spajanja vanjskih periferija i kupnje proizvoda preko interneta. Nakon nabave fizičkog uređaja za testiranje, aplikacija bi mogla biti preuređena i optimizirana tako da se spaja s bazom podataka neke stvarne trgovine, te bi mogla biti objavljena u trgovini App Store.

LITERATURA

[1] S. Bryson, »Virtual Reality: A Definition History - A Personal Essay,« 15.122013. [Mrežno]. Dostupno: https://arxiv.org/pdf/1312.4322. [Pokušaj pristupa 17.5.2024]. pp. 3-5.

[2] C. Anthes, R. Hernandez, D. Kranzlmueller, M. Wiedemann, »State of the Art of Virtual Reality Technologies,« u *Human-Computer Interaction International Conference (HCII)*, Toronto, Canada, 2016. pp. 3-4.

[3] M. Pesce, »Samsung Gear 360 and Gear VR headset (26693467445),« Meta Inc., [Mrežno]. Dostupno: https://commons.wikimedia.org/w/index.php?curid=51016371. [Pokušaj pristupa 9.8.2024].

[4] J. Carmigniani, B. Furht, Augmented Reality: An Overview, Boca Raton, Florida: Florida Atlantic University, 2011. pp. 3-5.

[5] M. Speicher, B. Hall, M. Nebeling, »What is Mixed Reality?,« u *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*, Glasgow, 2019, pp. 2-3.

[6] L. Lim, W. Goh, M. Downing, M. Sra, »A Spatial Music Listening Experience in Augmented Reality,« u *Adjunct Publication of the 34th Annual ACM Symposium on User Interface Software and Technology (UIST 2021)*, 2021. pp. 23-24.

[7] X. Sheng, S. Mao, Y. Yan, X. Yang, »Review on SLAM algorithms for Augmented Reality,« *Displays,* svez. 84, 2024. pp 1-4.

[8] C. Theodorou, V. Velisavljevic, V. Dyo, F. Nonyelu , »Visual SLAM algorithms and their application for AR, mapping, localization and wayfinding,« *Array,* svez. 15, 2022. pp 1-7.

[9] A. E. Mahdi, A. Azouz, A. E. Abdalla, A. Abosekeen , »A Machine Learning Approach for an Improved Inertial Navigation System Solution,« *Sensors,* svez. 22, br. 2, 2022.

[10] R. Weitzman, A. Mizrahi, R. Perlman, E. Haviv, O. Pyshchenko, Trevor Stephenson »Real Time Ray Tracing in AR,« u *ACM Symposium on Spatial User Interaction (SUI) 2023*, Sydney, 2023, pp. 132-134.

[11] A. L. Dos Santos, D. Lemos, J. E. F. Lindoso, V. Teichrieb , »Real Time Ray Tracing for Augmented Reality,« u *Virtual and Augmented Reality (SVR), 2012 14th Symposium*, 2012. pp 131-136

[12] Meta Inc., »Facebook to Acquire Oculus,« 25.3.2014. [Mrežno]. Dostupno: https://about.fb.com/news/2014/03/facebook-to-acquire-oculus/. [Pokušaj pristupa 2024].

[13] Wikimedia User, - "Ihazacatnamedmax", »Meta Quest 3 display unit,« 29 10 2023. [Mrežno]. Dostupno: https://commons.wikimedia.org/w/index.php?curid=139735935. [Pokušaj pristupa 9.8.2024].

[14] Microsoft Inc., »Microsoft Hololens 2,« Microsoft Inc., 2021. [Mrežno]. Dostupno: https://www.microsoft.com/de-de/d/hololens-2/91pnzzznzwcp?activetab=pivot:übersichttab.

[15] IKEA, »IKEA Place app launched to help people virtually place furniture at home,« 12. 9.2017. [Mrežno]. Dostupno: https://www.ikea.com/global/en/newsroom/innovation/ikea-launches-ikea-place-a-new-app-that-allows-people-to-virtually-place-furniture-in-their-home-170912/. [Pokušaj pristupa 21.4.2024].

[16] Varjo, »Motion Sickness In VR,« [Mrežno]. Dostupno: https://varjo.com/learning-hub/motion-sickness/#:~:text=Motion%20sickness%2C%20also%20known%20as,%2C%20eye%20strain%2C%20and%20drowsiness.

[17] S. Rokhsaritalemi, A. Sadeghi-Niaraki, S. Choi , »A Review on Mixed Reality: Current Trends, Challenges and Prospects,« 16 1 2020. [Mrežno]. Dostupno: applsci-10-00636.pdf. [Pokušaj pristupa 17.5.2024].

[18] Apple Inc., »Apple Inc.,« 5.6.2023. [Mrežno]. Dostupno: https://www.apple.com/apple-vision-pro/. [Pokušaj pristupa 2.5.2024].

[19] Apple Inc., Apple Inc., 28.2.2024. [Mrežno]. Dostupno: https://support.apple.com/en-us/118475. [Pokušaj pristupa 4.5.2024].

[20] M. Fujimiya, »The outer glass of Apple Vision Pro,« 3.2.2024. [Mrežno]. Dostupno: https://commons.wikimedia.org/w/index.php?curid=144986304. [Pokušaj pristupa 9.8.2024].

[21] Apple Inc., »Apple unveils M2, taking the breakthrough performance and capabilities of M1 even further,« Apple Inc., 6.6.2022. [Mrežno]. Dostupno: https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/. [Pokušaj pristupa 3.5.2024].

[22] Apple Inc., »Apple Vision Pro Specifications,« Apple Inc., 5.6.2023. [Mrežno]. Dostupno: https://www.apple.com/apple-vision-pro/specs/. [Pokušaj pristupa 6.5.2024].

[23] Apple Inc., »About Optic ID advanced technology,« Apple Inc., 28.2.2024. [Mrežno]. Dostupno: https://support.apple.com/en-us/118483. [Pokušaj pristupa 7.5.2024].

[24] Apple Developer, »Design For Spatial Input,« Apple Inc., 3.6.2023. [Mrežno]. Dostupno: https://developer.apple.com/videos/play/wwdc2023/10073/?time=741. [Pokušaj pristupa 8.5.2024].

[25] Apple Developer, »RealityKit,« Apple Inc., 2019. [Mrežno]. Dostupno: https://developer.apple.com/documentation/realitykit/. [Pokušaj pristupa 27.3.2024].

[26] TechTarget, »Model-View-ViewModel (MVVM),« TechTarget, 1.2.2019. [Mrežno]. Dostupno: https://www.techtarget.com/whatis/definition/Model-View-ViewModel. [Pokušaj pristupa 28.3.2024].

[27] M. Aljamea, M. Alkandari, »MMVMi: A Validation Model for MVC and MVVM Design Patterns in iOS Applications,« *IAENG International Journal of Computer Science,* svez. 45, br. 3, pp. 1-3, 2018.

[28] J. Kouraklis, MVVM in Delphi, Twickenham: Apress, 2016.

[29] Apple Inc., The Swift Programming Language, Cupertino: Apple Inc., 2014.

[30] Hacking With Swift, »What is SwiftUI?,« 17.6.2023. [Mrežno]. Dostupno: https://www.hackingwithswift.com/quick-start/swiftui/what-is-swiftui. [Pokušaj pristupa 3.4.2024].

[31] Apple Developer, »Xcode,« Apple Inc., 2019. [Mrežno]. Dostupno: https://developer.apple.com/documentation/xcode. [Pokušaj pristupa 3.4.2024].

[32] Apple Newsroom, Apple Inc., 1.2.2024. [Mrežno]. Dostupno: https://www.apple.com/newsroom/2024/02/apple-announces-more-than-600-new-apps-built-for-apple-vision-pro/. [Pokušaj pristupa 12.4.2024].

[33] Apple Developer, Apple Inc., 12.6.2023. [Mrežno]. Dostupno: https://developer.apple.com/news/?id=8sntwknb. [Pokušaj pristupa 12.4.2024].

SAŽETAK

MOBILNA APLIKACIJA ZA POTPORU PROSTORNOM RAČUNARSTVU KORIŠTENJEM UREĐAJA APPLE VISION PRO

U ovom diplomskom radu provedeno je temeljito istraživanje platforme Apple visionOS i izrađena aplikacija koja predstavlja prototip web trgovine u proširenoj stvarnosti. Istražena su načela mješovite, virtualne i proširene stvarnosti, te je dan pregled postojećih programskih i sklopovskih rješenja za uređaje koji pristupaju takvim sučeljima. Aplikacija je razvijena korištenjem programskog jezika Swift i razvojnog okvira SwiftUI, uz korištenje baze podataka Firebase za pohranu i učitavanje podataka. Omogućuje jednostavan pregled i složene interakcije s trodimenzionalnim modelima proizvoda kao i simulaciju narudžbe proizvoda te filtriranje, dijeljenje i pretraživanje proizvoda. Aplikacija upravlja velikim brojem USDZ datoteka za prikazivanje modela proizvoda te se analizira njihov utjecaj na performanse i zauzeće memorije virtualnog simulatora. Temelji se programskom obrascu MVVM uz proširenja za mrežni repozitorij i nudi podršku za razvojne okvire Apple ARKit i Reality Kit. Korišteno je razvojno okruženje Xcode uz visionOS Simulator. Provedeno je temeljito testiranje aplikacije, no zbog ograničenja postavljenih u simulatoru, postoji mogućnost poboljšanja, primjerice pokretanjem aplikacije s različitim pozadinama i okruženjima. Cilj diplomskog rada je postignut, no s obzirom na brzi napredak platforme visionOS kao i na nestabilnost početnih inačica sustava, moguće je poboljšavati i dodavati još funkcionalnosti u aplikaciju.

Ključne riječi: mješovita stvarnost, proširena stvarnost, Swift, uređaj Apple Vision Pro, VisionOS.

ABSTRACT

MOBILE APPLICATION FOR SPATIAL COMPUTING SUPPORT USING THE APPLE VISION PRO DEVICE

In this Master's Thesis, a thorough research of the Apple visionOS platform is carried out, and an application which represents a prototype of an augmented reality web-store is created. The concepts of augmented, virtual and mixed realities are researched, and an overview of existing hardware and software solutions for devices accessing such interfaces is provided. The application is developed using the Swift programming language as well as the SwiftUI development framework, with the usage of Firebase database for storing and loading data. The application enables a simple overview and complex interactions with three-dimensional product models as well as simulations of product orders and filtering, sharing and searching for products. The application handles a large number of USDZ files used for displaying product models and their impact on the performance and memory usage of the virtual simulator is analyzed. It's based on the MVVM programming pattern along with network repository extensions and offers Apple ARKit and Reality Kit framework support. The Xcode Integrated Programming Environment along with the visionOS Simulator have been used. A thorough testing process of the application has been conducted, but because of limitations set by the simulator, there is possibility for improvement, e.g. by running the application with different backgrounds and environments. The goal of the thesis has been achieved, but taking into account the fast-paced progress of the visionOS platform and the instability of initial versions of the system, it's possible to add additional improvements and functionality to the application.

Keywords: Apple, augmented reality, mixed reality, Swift, VisionOS.

ŽIVOTOPIS

Luka Lešić rođen je 2. prosinca 2000. godine u Vinkovcima. Pohađa Opću Gimnaziju u Županji, nakon koje upisuje preddiplomski studij te zatim diplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Profesionalno se bavi razvojem iOS i iPadOS aplikacija kao razvojni programer u organizaciji Check24 Vergleichsportal GmbH.

_____

Potpis autora

PRILOZI (on-line)

Prilog 1. Diplomski rad u datoteci .docx

Prilog 2. Diplomski rad u datoteci .pdf

Prilog 3. Prezentacija za obranu rada u datoteci .pptx

Prilog 4. Programski kod mobilne aplikacije (Poveznica na Github repozitorij)