Budućnost tehnologija za razvoj stolnih programskih proizvoda

Tomljenović, Vid

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vid Tomljenović

BUDUĆNOST TEHNOLOGIJA ZA RAZVOJ STOLNIH PROGRAMSKIH PROIZVODA

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vid Tomljenović

Matični broj: 37275/07 - I

Studij: Poslovni sustavi

BUDUĆNOST TEHNOLOGIJA ZA RAZVOJ STOLNIH PROGRAMSKIH PROIZVODA

Završni rad

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, srpanj 2022.

Vid Tomljenović

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Završni rad tematike „Budućnost tehnologija za razvoj stolnih programskih proizvoda“ svoje temelje gradi na problematici koja se očituje u sve većem broju mobilnih ili internetskih aplikacija ili pak migracije stolnih aplikacija na neku od spomenutih platformi. Problematika kao takva sve je relevantniji stoga je cilj ovoga rada da posluži kao instrument u razrješavanju ovog toka misli koji obuzima sve veći broj ljudi, kako u struci tako van nje. Rad će kroz dva osnovna dijela, teorijski i praktični, pokazati gdje, kako i zašto su stolne aplikacije i dalje veoma relevantne i potrebne te da će takve i ostati. Rad ovo ostvaruje teorijskim dijelom, ujedno i prvim dijelom rada, kroz teorijske i istraživačke metode poput pronalazaka i čitanja relevantnih internetskih članaka, blogova i knjiga te praktični dijelom, ujedno i drugi dijelom rada, kroz izradu dvaju primjera stolnih programskih proizvoda što bi se moglo smatrati eksperimentalnim načinom istraživanja. Kraj rezultira donošenjem zaključka na temelju oba dijela iscrpnog istraživanja, koji ustanovljuje upravo predloženu tezu: stolne aplikacije relevantne su te od iznimnoga značaja, kako za krajnje korisnike tako i za male, srednje, velike firme, samozaposlene, a samim time i tehnologije korištene za njihovu izradu i održavanje.

Ključne riječi: aplikacije, stolne aplikacije, razvoj aplikacija, budućnost stolnih aplikacija, korištene tehnologije, buduće tehnologije, programski proizvodi, programsko inženjerstvo, Flutter, Dart

Sadržaj

1. Uvod

Počnimo uvodnom riječju o terminologiji rada, nju je prijeko potrebno upoznati i razumjeti za što potpunije shvaćanje samoga rada, odnosno problema koji rad nalaže: „Budućnost tehnologija za razvoj stolnih programskih proizvoda“. Napokon, rješenje ili rješenja koja ćemo razvijati, a zatim aplicirati u svrhu rješavanja navedenoga problema moguće je realizirati jedino na takav način. Shodno tome napomenimo kako ovakav rad, kako sadržajno tako i problemski, ponajviše pripada jednoj od najopsežnijih grana računalne znanosti: „Programsko inženjerstvo“ (eng. Software engineering). „*Grana računalne znanosti koja se bavi dizajnom, implementacijom i održavanjem složenih računalnih programa*“ („Merriam-Webster Dictionary“, bez dat.), definicija je koja ponajbolje opisuje tu granu.

(Stolne) aplikacije, razvoj istih te tehnologije koje se koriste za razvoj, osnovna su terminologija koju moramo razumjeti kako bismo kompetentno raspravili o glavnomu problemu koji nalaže ovaj završni rad. Kako naslov rada navodi, najveći fokus bit će upravo na problemu koji se očituje u budućnosti tehnologija za razvoj stolnih aplikacija, odnosno programskih proizvoda. Tema kao takva veoma je aktualna, a razlog tomu jest taj što sve veći broj aplikacija migrira na internet ili su pak izvorno stvorene za njega stvorene, samim time mi kao krajnji korisnici također smo primorani „seliti“ i koristiti *cloud* kao ozračje za rad u željenim aplikacijama.

„Aplikacija (latin. applicatio) definiramo kao primjenu zakona, metoda, lijekova“ („Hrvatska enciklopedija [EH]“, bez dat.). Definicija kao takva prihvatljiva je i u informatičkom smislu riječi. Naime računalnu aplikaciju možemo stvoriti samo korištenjem određenih zakona i metoda primjerice pisanjem programskog koda u jednom od programskih jezika. Programski jezik, kao i svaki drugi, sadrži svoje sintaktičke, semantičke, pragmatičke zakone koje ga čine jedinstvenim. Također aplikaciju možemo pravilno koristiti jedino ako ispravno razumijemo njezina pravila, primjerice *web* aplikaciju ne možemo koristiti bez pristupa internetu. „Aplikacija u informatici jest primjenski ili korisnički program, skup naredbi i uputa koji omogućuje izvršenje neke zadaće, za razliku od sustavskih programa kao što su operacijski sustavi, odnosno uslužnih i pomoćnih programa te programskih jezika.“ („EH“, bez dat.).

Već smo spomenuli kako je programsko inženjerstvo disciplina koja nam omogućuje da proizvodimo krajnje programske proizvode za korisnike, bili oni pojedinci ili velika poduzeća. *Software engineering* zaslužan je za dizajniranje, kreiranje, testiranje, ispitivanje kvalitete i održavanje kvalitete programskih rješenja te time i temelj za njihovo postojanje. Strahonja programsko inženjerstvo navodi kao: „Sustavnu primjenu inženjerskog pristupa,

procesa i metoda u analizi, projektiranju, vrednovanju, primjeni, testiranju, održavanju i promjeni softvera.“ (Strahonja, 2017.).

Razriješivši osnovnu terminologiju rada spremni smo prijeći na raspravu o samoj temi. Koje se tehnologije koriste pri izradi programskih proizvoda, koje nam dolaze, a koje odlaze i koje su to osnovne razlike između stolnih i *web* tehnologija samo su jedni od dijelova slagalice koji će nam omogućiti stvoriti jedan funkcionalan stolni programski proizvod.

Razrada jest stoga podijeljena u dva glavna odjeljka. Prvi odjeljak, 2.1. – 2.6., jest teorijski dio rada u kojemu sagledavamo povijesno, sadašnje i buduće stanje stolnih programskih proizvoda i njihovih razvojnih tehnologija. Drugi odjeljak razrade, 3.1. – 3.4., jest praktični dio rada u kojemu sagledavamo jednu od novih (eng. Emerging) tehnologija za razvoj stolnih aplikacija, preciznije rečeno izrađujemo jednostavan primjer s osnovnim konceptima programskog proizvoda te ovaj rad koristimo kao svojevrsnu dokumentaciju za isti.

Primorani smo sagledati problem koji nalaže ovaj rad, prethodno spomenuta budućnost tehnologija za razvoj stolnih programskih proizvoda nalaže nešto čime se susreo svaki developer kao i svaka osoba zaposlena u IT industriji. Riječ je svojevrsnoj stigmi koja se nametnula razvoju stolnih programskih proizvoda. Stigma o kojoj je riječ je naravno odumiranje navedene discipline, a razloga i opravdanja za takvo mišljenje jest mnogo. Istina je dakako daleko od jednostavnih crno-bijelih subjektivnih mišljenja pojedinaca. Cilj ovoga rada tako je upravo opovrgnuti nastalu stigmu kroz ostvarivanje dva manja cilja. Teoretski cilj zadan je kako bismo uvidjeli korijene problema, *status quo* problema, budućnost problema te djelovali sukladno prikupljenim informacijama iz različitih izvora te oblikovali vlastito mišljenje bazirano na dobroj informiranosti. „Riječi govore više od djela“ stoga je zadan i praktični cilj kroz koji ćemo iskoristiti sve prikupljene informacije za stvaranje proizvoda. Cilj izrade jest predstaviti jednu od novih tehnologija za stvaranje stolnih programskih proizvoda, bilo da je riječ o tehnologiji namijenjenoj isključivo za *desktop development* ili je ono samo jedna od komponenata.

Dispozicija ovoga rada kao takva poslužit će nam za ostvarivanje jediničnih ciljeva kojima ćemo u konačnici biti u mogućnosti riješiti cjelokupni problem naložen u ovome radu. Dispozicija jest sljedeća: pregled ujedno prošlog i trenutnog stanja stolnih aplikacija i tehnologija. Nastavak slijedi s temom moguće nove industrijske revolucije, indirektna korelacija s napretkom i budućnošću razvoja novih i unaprjeđenja starih tehnologija nešto je na što bismo trebali pripaziti. Cilj da sagledamo ukupnu prošlost, sadašnjost i budućnost stolnih aplikacija i tehnologija iza njih, moguću industrijsku revoluciju, razlike između stolnih i web aplikacija / tehnologija te ipak uvidimo važnost i korisnost koja se i dalje nalazi u stolnim aplikacijskim proizvodima. Sukladno tome, logički slijed razrađivanja teorije bit će od prošlog

i trenutnog stanja tehnologija, preko usporedbe web i desktop aplikacija (gdje ćemo se više obazirati na web tehnologije) te budućnost tehnologija, pri čemu ćemo se osvrnuti i na tehnologiju koja će biti rabljena za izradu praktičnog dijela, odnosno napraviti kratki uvod.

2. Tehnologije stolnih aplikacija

Rješenje problema naloženog ovim završnim radom razloženo je na već spomenute dvije etape. Etape su teorijska razrada te praktična razrada rješenja za zadani problem. Dispozicija rada zamišljena je upravo na taj način kako bismo se upoznali s terminologijom, zanimljivostima, prednostima / manama tehnologija za izradu programskih rješenja kao takvih u slici široj od samih stolnih aplikacija. Nadolazeća potpoglavlja upravo su alat koji će nam poslužiti pri teorijskoj razradi rješenja problema, gdje je terminologija većim dijelom pokrivena u razrađenom uvodu, zanimljivosti slijede u potpoglavljima kao što su prošlost stolnih tehnologija, pete industrijske revolucije te različitim područjima primjene stolnih programskih proizvoda navedenih u potpoglavlju o važnosti istih. Završetkom teorijske razradbe spremni smo novo znanje imati na umu i iskoristiti pri praktičnoj razradi gdje ćemo naravno izrađivati jednostavno programsko rješenje u jednoj od novijih tehnologija.

2.1. Prošlost i sadašnjost stolnih aplikacija i njihovih tehnologija

Za početak pogledajmo u prošlost, IT industrija nije krenula od aplikacija odnosno softvera, već od hardverskih proizvoda koji nisu bili programirani. Njihove funkcionalnosti, njihova svrha, njihova mogućnost ostvarenja zadaće očitavala se u njihovom specifičnom obliku i inženjeringu. Najpoznatiji takav izum jest naravno *Abacus* odnosno: „Manualni računalni uređaj koji se sastoji od okvira koji drži paralelne šipke na kojima se nalaze nanizani pomični brojači“, navedeno na stranici („American Heritage - Dictionary of the English Language, Fifth Edition“, 2016).

Vratimo se u nešto bližu povijest, točnije 21. lipnja 1948. godine. Ovaj datum nam zvanično predstavlja početak programskog inženjerstva i programiranja kao takvog. Jerome Rault (2020) na stranicama *laneways-agency* navodi razlog taj što je računalo Manchesterski mali eksperimentalni stroj, kasnije poznatiji kao *Manchester baby*, bilo prvo računalo s pohranjenim programom na svijetu te je uspješno izvršilo svoj prvi program (Rault, 2020). Rault dodaje kako je navedena aplikacija pomoću uputa i strojnog koda izvodila matematičke

operacije kako bi, nakon 52 minute, izračunala najvećeg djelitelja. (Rault, 2020). Prema današnjim standardima ništa naročito, no zasigurno početak, svojevrsna revolucija. Nedugo nakon, 1950. Kathleen Booth razvila je asemblerski jezik odnosno: „Programski jezik niske razine koji je namijenjen izravnoj komunikaciji s hardverom računala te za razliku od strojnog jezika (bin i hex znakova), asemblerski jezici dizajnirani su tako da ih mogu čitati ljudi.“ (Fernando 2020.).

Nije bilo potrebno previše vremena da se uskoro pojave i prvi programski jezici (Cobol 1959., C 1978., Pascal 1985.), prvo osobno IBM računalo 1981. te dostupnost interneta za javnu uporabu s funkcionalnim grafičkim sučeljem 1993. Ako u obzir uzmemo prethodno spomenuto pojavljivanje prvog pohranjenog programa na računalu s vlastitom memorijom (1948.) uviđamo kako je do ekstremnih promjena i napretka došlo u manje od polovice stoljeća. Ova činjenica računalstvo, programsko inženjerstvo, informatiku i cjelokupnu IT industriju stavlja u poziciju visoke kompetentnosti. Sadašnjost je još nemilosrdnija te od zaposlenika, koji žele biti donekle uspješni i kompetentni traži visoko obrazovanje, cjeloživotno učenje te prije svega visok interes i želju za samim radom i naobrazbom. Prilog ovome ide i to što je treća i četvrta industrijska revolucija proizašla upravo radi napredaka ostvarenih u IT industriji i zahvaljujući njoj. Riječi će biti i o petoj industrijskoj revoluciji u kasnijem dijelu rada.

Drugim dijelom ovoga potpoglavlja sagledavamo trenutnu sliku programskog inženjerstva, aplikacijskih proizvoda i tehnologija koje stoje iza njih. Već na samome početku moramo istaknuti bitan detalj… teško je odabrati najbolju tehnologiju koja je namijenjena isključivo razvoju stolnih aplikacijskih proizvoda za jedno ozračje. Što to znači? To znači da, primjerice Microsoft, kao vodeća tvrtka u razvoju tehnologija za razvoj programskih proizvoda iz svog pogona u javnu uporabu i nabavu pušta tehnologiju poput UWP („Universal Windows Platform“). Rault (2020) za istu stranice *laneways-technology* pod istom temom govori kako je UWP tehnologija koja omogućuje developerima da izrađuju programske proizvode za PC i tablete, ali čak i Xbox te pametne telefone. Preduvjet: Windows 10 mora biti njihov radni OS. Predstavljen 2015. zapravo nikada nije u potpunosti prihvaćen od strane developera te je i danas manje zastupljen od WinFormsa i WPF-a također tehnologija Microsofta. Nadalje Rault (2020) se osvrće i na postojanje *Cocoa*, tehnologije koja je namijenjena platformama koje koriste Macintosh OS, no zamijenio tzv. *Cocoa touch*.

Navedene tehnologije uglavnom se obaziru na dva najveća operacijska sustava stolnih računala, Microsoft i Macintosh, stoga možemo samo zamisliti kakvi problemi nastaju jednom kada idemo šire od toga! Problemi ovog tipa novonastali su upravo dolaskom više različitih tehnologija više različitih proizvođača. Odnosno s ovom situacijom se nismo susretali krajem prošloga stoljeća kada su na tržištu postojali Siemens, Microsoft, IBM….. U drugu ruku, danas

se moramo zapitati mnoštvo pitanja prije nego uopće krenemo s radom i korištenjem tehnologija. Tko su nam klijenti; žele li stolni proizvod ili aplikaciju na prijenosnim uređajima ili web servis; Koji ćemo *framework* koristiti za ostvarenje postavljenih ciljeva; Možemo li računati da su i *front end* i *back end* odijeli dovoljno kompetentni za rad s izabranim tehnologijama; Hoće li testiranje biti problematično; Možemo li si priuštiti te tehnologije (pitanje financija) itd. Izbor tehnologija će se reflektirati na sve zaposlenike i na konačnu isporuku kupcima stoga je iznimno važno upoznati sebe i suradnike i biti uključen u proces odlučivanja i informiranja. Ovdje se također uviđa detalj koji smo spomenuli u uvodu, a to je razina kompetencije, (pred)znanja, zainteresiranosti te naravno volje za napretkom i učenjem. Kontrastno rečenome, sve više znanja koriste automatizirani procesi. Ovo dovodi čovjeka kao pojedinca do toga da rjeđe pribjegava korištenju vlastitog znanja pri izradi rješenja te se oslanja na korištenje tuđih ili uvaženih (što je dopustivo), ali i prepuštanju sve više dijelova tog procesa AI-u. Upravo to je „dvosjekli mač“ koji nas dovodi do sljedećeg poglavlja, sljedeće industrijske revolucije!

2.2. Peta industrijska revolucija

Potpoglavlje jest razrađeno u obliku esejskih odgovora na nekoliko vitalnih pitanja koja će nam pomoći shvatiti sam termi. Tema kao takva nije usko vezana uz samo jednu preciznu tehnologiju ili programski jezik i okvir, no veoma je zanimljiva jer je riječ o revolucionarnom događaju, kako što i sama riječ nalaže koji će se afektirati na sve slojeve industrije, posebice IT industrije o kojoj je u ovome radu i riječ! Pitanja su sljedeća:

- Kada?

  Uskoro! Gledajući vremenske periode, ujedno navedene i od strane Ali, S.H. i suradnika (2022), onaj između treće, koja je nastala 1980-te, i četvrte, koja je nastala 2011, ovaj između četvrte i pete jest kraći. Razlog tomu je što se smatra kako će do pete industrijske revolucije (5IR) doći u trećem desetljeću 2000-tih, preciznije između 2025. i 2035. godine (Ali, S.H. *et al*, 2022). Razlika u periodu je zastrašujuće velika! Četvrta nam je došla otprilike 30 godina nakon treće, dok će nam peta doći otprilike 15 godina nakon četvrte. Razlika od petnaest godina između nastanka revolucija nešto je izvanredno! Budući da je i peta industrijska revolucija termin budućnosti možemo samo nagađati njezin vremenski opseg, odnosno trajanje prije nego se pojavi i šesta! Ipak možemo se voditi nečime u nagađanju. Gledajući kako se vremenski interval između svake revolucije sve

više smanjuje jedino je logično za zaključiti kako će peta potrajati barem 8 godina što znači da bismo mogli doživjeti tri industrijske revolucije u pola stoljeća!

- Gdje?

Budući da su druga, treća i četvrta industrijska revolucija proizašle iz SAD-a ne bismo bili u krivu kada bismo nagađali kako će i sljedeća, peta imati svoje korijene baš ondje. No Ali, S.H. i suradnici (2022) navode da Sjedinjene Američke Države nisu jedini natjecatelj u utrci za ovu titulu. Riječ je i o Kini koje je bila uključena u četvrtu industrijsku revoluciju te stoji rame uz rame SAD-u… ne zaboravimo i uvijek aktualnu Rusiju i Njemačku, a ako je riječ o umjetnoj inteligenciji, nano tehnologijama i robotima, temama koje su upravo sve aktualnije, Japan vodi glavnu riječ (Ali, S.H. *et al*, 2022).Tko će od njih biti na vrhu i hoće li biti sam / sama? Možemo samo strpljivo čekati.

Na pitanje „Gdje?“ ne možemo u potpunosti odgovoriti a da se ne osvrnemo na radna mjesta i strukturu radnih mjesta koja će biti promijenjena, zamijenjena, u potpunosti nova ili nestala. *Regenesys Business School* u svome članku o petoj industrijskoj revoluciji kao neka od zanimanja koja će postati redundantna u većini razvijenih zemalja navodi službenike za manualni unos podataka, blagajnike, financijske analitičare, telemarketare, pa čak i odvjetnike, dok će zanimanja poput menadžera i izvršnih direktora, HR specijalista, inženjera, sveučilišnih profesora ostati više-manje stabilna. Nova (eng. Emerging) zanimanja nas najviše zanimaju, samim time što je velik broj njih vezan uz IT sektor kao cjelinu, ali i zbog toga što nadolazeće promjene (in)direktno utječu na tehnologije koje se koriste pri dizajniranju, proizvodnji, održavanju (stolnih) programskih proizvoda. Neka od tih zanimanja *Regenesys Business School* (2020) za isti članak navodi analitičare podataka, specijaliste za AI i strojno učenje, specijaliste za društvene mreže, dizajnere servisa i rješenja itd. („Regenesys Business School“ ,2020).

- Kako (očekivanja)?

Iako je razvoj tehnologije ono što pokreće svaku industrijsku revoluciju, tehnologija nije jedina okosnica nadolazeće revolucije. *Regenesys Business School* (2020) glavnu riječ stavlja na naglasak personalizacije! Inovacije će biti više inkluzivne, drugim riječima čovjek neće morati biti izrazito naučan da bi ih mogao koristiti za svakodnevno rješavanje prepreka. Primjer toga može biti dostava hrane ili svakojakih namirnica putem samovozećih dronova, automobila („Regenesys Business School“, 2020). Korisniku tada ostaje jednosavan zadatak narudžbe putem / korištenjem aplikacija aplikaciju za narudžbu, što uključuje i pregled i plaćanje i ostavljanje ili čitanje povratnih informacija vezanih uz cjelokupni proces.

Upravo zato sljedeću industrijsku revoluciju i naša očekivanja ne možemo i ne smijemo svesti na samo jedno znanstveno područje. Neka od mogućih očekivanja i predikcija koje Ali, S.H. i suradnici (2022) navode su sljedeća: „Revolucija spašavanja planeta“, „Revolucija vrijednosti i povjerenja“, „Revolucija AI tehnologija“ (najizvjesnija). Vidljivo je kako su uključene i prirodne i društvene i tehnološke znanosti što ovu industrijsku revoluciju čini iznimno zanimljivom (Ali, S.H. *et al*, 2022)! Budućnost kao takva skriva mnogo toga, no na nama je da kao korisnici, zaposlenici, developeri, studenti te građani za istu budemo što spremniji. Kada je pak riječ o (stolnim) tehnologijama i aplikacijama preciznije i detaljnije ćemo iste sagledati u potpoglavlju 2.5. istoimenog naslova budućnost stolnih aplikacija i njihovih tehnologija.

2.3. Stolne vs mrežne aplikacije

Potpoglavlje bavi se različitim prednostima i manama pristupa stvaranja programskih proizvoda, bolje znano kao *pro's & con's*. Razlog tomu jest što mnoge prednosti jednog pristupa bit će upravo nedostaci drugoga (i *vice versa*).

Već smo razjasnili stolne (engl. *desktop*) aplikacije kao pojam, stoga razjasnimo pojam *web* aplikacija. „Računalni program dizajniran za određenu svrhu kojeg osoba može koristiti na internetu umjesto da je preuzima.“ (Cambridge dictionary, bez dat.). Ovdje nailazimo na jednu od najvećih razlika među stolnim i internetskim aplikacijama, a to je naravno potreba za preuzimanje sadržaja na računalo. Za razliku od stolnih aplikacija koje se ručno dodjeljuju svakom računalu / korisniku, internetske aplikacije su trenutno dostupne svim računalima / korisnicima u isto vrijeme. Sam korisnik za pristup željenoj internetskoj aplikaciji izabire omiljeni internetski preglednik (Mozilla, Safari, Internet Explorer, Google Chrome itd.) te unosom zadane adrese (URL-a) pristupa stranici, servisu ili aplikaciji. Razjasnimo također kako je svaka internetska aplikacija ujedno i internetska stranica, no obrnuta analogija ne vrijedi. Naime nije svaka internetska stranica ujedno i internetska aplikacija, a razlog tomu je što da bi se nešto klasificiralo kao aplikacija mora ispuniti osnovni zahtjev, a to je da obavlja neku funkciju za krajnjeg korisnika.

Važno je naglasiti kako u ovome potpoglavlju neće biti u potpunosti razrađene prednosti stolnih aplikacija. Razlog tomu je što one logički znatno više pripadaju u potpoglavlje „Važnost stolnih aplikacija“, koje je sljedeće potpoglavlje, potpoglavlje 2.4. Ondje će nam te prednosti uvelike pomoći uvidjeti važnost stolnih aplikacija i važnost njihove daljnje primjene, naravno kao i tehnologija koje su iza njih, iako veliki dio kolača tržišta uzimaju internetske aplikacije.

Mnoge od nedostataka stolnih aplikacija možemo prepoznati samostalno, a prepoznali su ih i na stranicama *Qulix* za koje je Nadezhda Mal (2022) navela fiksiranje na jedan sustav (odnosno računalo), što nas dovodi i do saznanja kako one zauzimaju neki prostor na našem disku. Sukladno tome stolne aplikacije zahtijevaju preuzimanje, ponekad raspakiravanje te instalaciju, a to pak znači kako će aplikacija biti dostupna na tome uređaju / sustavu, odnosno instaliranu aplikaciju koristimo na svome prijenosnom računalu te ne možemo istu instalirati na drugo prijenosno računalo i očekivati da će nas dočekati naš spremljeni napredak s prethodnog računala (Nadezhda Mal, 2022). Svakako, današnje stolne aplikacije dobivaju sve više mogućnosti i funkcionalnosti da upravo to učine mogućim, no za to koriste upravo „*web*“ servise za koje nam je potrebna povezanost s internetom. Iako se navedene stavke ne čine kao ne premostivi nedostaci, upravo su oni razriješeni korištenjem „*web*“ tehnologija za stvaranje *web* aplikacija. Mal (2022) napominje još jednu boljku stolnih aplikacijskih proizvoda, a riječ je o: Zauzimanju prostora nakon što se provede proces instalacije, koji će potrajati, jednako kao i proces ažuriranja koji je u većini slučajeva manualan. Ukoliko se ista više ne koristi potrebno je provesti još jedan proces, proces deinstalacije, koji će ponovno potrajati… i tako za sve različite uređaje (Nadezhda Mal, 2022)! Ipak da nije sve „tako sivo“ za stolne aplikacije dokazat ćemo već u sljedećem poglavlju.

Nadalje, prednosti koje se očituju kod internetskih aplikacija upravo su nabrojeni nedostaci stolnih aplikacija. Mal (2022) za iste stranice kao oprečnu sliku nedostataka stolnih aplikacija navodi neke od prednosti internetskih: Nepostojanje potrebe za preuzimanjem, raspakiravanjem i instaliranjem aplikacije. Shodno tome ne zauzimamo prostor na našem računalu / sustavu što također znači da naše računalo ne mora biti najmodernije kako bi pokretalo neku od aplikacija jer to ne radi na sebi (Nadezhda Mal, 2022). Samostalno zaključujemo kako to znači da internetske aplikacije možemo pokretati s različitih uređaja, kao što su stolna, prijenosna, pa čak i mobilni uređaji, što također znači mogućnost pokretanja na različitim operacijskim sustavima i okolišima (npr. iOS i MS).

Nažalost, podrazumijeva se kako svaki proizvod, aplikacija, tehnologija ima svoje nedostatke, pa tako i *web* tehnologije i *web* aplikacije. Shodno tome mnoge od njihovih nedostataka pokriveni su prednostima stolnih aplikacija. Naravno prvi i najveći od nedostataka, zaključujemo, jest potreba za stalnom povezanošću s internetom. Povezanost s internetom predstavlja i sigurnosni rizik za nas pošto su svi podaci čuvani na tzv. javnom oblaku koji služi upravo razrješavanju prethodno spomenutog nedostatka stolnih aplikacija. Riječ je naravno o pristupu sačuvanom napretku u našoj aplikaciji s bilo kojeg uređaja u bilo kojem trenutku. Stranice Qulix još jednom uz Nadezhdu Mal (2022) nadopunjuju naša razmišljanja i navode kako oblak zahtjeva iznimnu dozu pozornosti po pitanju zaštite, što opet znači veći trošak za uspostavu i održavanje te sigurnosti. Kada je riječ o troškovima, mnoge

web aplikacije su u suštini jeftinije od stolnih aplikacija. Ovo je naravno istina za prvih par mjeseci / godina, no trošak se povećava sukladno s vremenom, sve dok ne dostigne i nadiđe trošak stolnih aplikacija. Jedan od čimbenika tomu jest što web stranica i web aplikacija zahtijevaju održavanje domene i hostinga te ostalih usluga. Zadnja , svakako ne manje bitna stavka koju Mal (2022) prepoznaje jest očita ovisnost aplikacije o stranici, što može utjecati na njenu brzinu izvedbe, veliki broj istovremenih korisnika, interni problem tvrtke, servera ili jednostavno loš hosting dovode do otežanog rada (Nadezhda Mal, 2022).

Razradivši sve prednosti i nedostatke, izuzev prednosti stolnih aplikacija ponovno zaključujemo da je teško napraviti pravi izbor pri odabiru najboljeg pristupa izradi aplikacija, izboru tehnologija, izboru jezika. Najpoželjniji pristup ipak je krenuti od klijenta, odnosno od onoga što on želi, a što mi možemo isporučiti, napraviti kompromis te isporučiti najbolji proizvod, ali uvijek težiti boljitku!

2.4. Važnost i primjena stolnih aplikacija

Potpoglavlje donosi prednosti stolnih programskih proizvoda nad internetskim te samim time i važnost korištenja, ali i razvijanja istih. Jednako tako važno je i naglasiti uporabnu svrhu stolnih aplikacija kako u poslovnim tako i privatnim ozračjima.

Zaključujemo, najočitija prednost upravo je bespotrebnost povezanosti s internetom. Jednom preuzeta i instalirana aplikacija izvodi se na našem sustavu bez potrebe za pristup internetu, samim time dobivamo i na sigurnosti. Nadalje Mal (2022) napominje kako oblak, spomenut u prethodnome potpoglavlju, ne nalazi svoje mjesto u svakoj stolnoj aplikaciji, a razlog tomu je što naše podatke spremamo lokalno, što ovakve aplikacije, odnosno naše podatke čini daleko zaštićenijima i sigurnijima (Nadezhda Mal, 2022). Samostalno zaključujemo kako tada nema potrebe za plaćanjem dodatnog održavanja sigurnosti tih podataka, zasigurno ne u mjeri kao što je to u internetskih aplikacija. Također stolne aplikacije jeftinije su na duže vremenske periode, razlog jest da za jednom kupljenu aplikaciju ili tehnologiju obično nema potrebe plaćati pretplate, domene itd.

Sagledavši konačno sve prednosti i nedostatke stolnih, ali i internetskih aplikacija, potrebno je uvidjeti kako u vremenu sve više rastuće popularnosti i primjene internetskih aplikacija, i dalje, neosporivu važnost imaju one stolne. Tehnologije koje stoje iza istih također su dalje vrlo aktualne, poput prethodno nabrojanih *Win forms*, *Electron*, *Swing*. Vrlo dobar primjer korištenja, i korisnosti stolnih aplikacija možemo pronaći i u poslovnom okruženju, jednako kao i u osobnom okruženju, odnosno tzv. kućnoj primjeni.

Kada je riječ o osobnoj primjeni u obzir možemo, moramo uzeti konkurente kao što su *MS Office*, *Adobe Reader & Photoshop*, *Unity* te mnoge videoigre. Nadalje programski proizvodi koji se manje baziraju na korištenju i manipulaciji medijskih datoteka također pripadaju kućnoj uporabi, a najčešće je riječ o programima za praćenje resursa. Za takve potrebe Marsh (2021) na stranicama *Datafloq* preciznije nas upoznaje s proizvodom „Nest" koji služi kao digitalni termostat, *Dropcountr* koji služi za očitanje potrošnje vode, *Breaker panel* koji naravno služi kao električna ploča s osiguračima i ostali (Jane Marsh, 2021). Veoma zanimljivu aplikaciju spominje nam Marsh (2021) na istoj stranici, a riječ je o aplikaciji koja objedinjuje i praćenje resursa i zabavu pod nazivom *JouleBug*. Povezujući svoj komunalni račun s aplikacijom, *JouleBug* dodjeljuje bodove i značke svaki puta kada se izvrši neka održiva radnja, poput štednje u kupovini (postoji i mogućnost natjecanja s ostalim korisnicima) (Jane Marsh, 2021).

Ne zaboravimo spomenuti i sve prisutnu uredsku primjenu stolnih programskih proizvoda. Kada je pak riječ o uredskoj primjeni u obzir također možemo uzeti *MS Office*, vrlo popularni email softver *MailChimp*, softver za kreiranje marketinških istraživanja *AdWords*, uvijek popularni *Skype* koji je uvelike zamijenjen novom stolnom aplikacijom „Zoom" itd. Budimo specifičniji i sagledajmo stolne programske proizvode vezane za određenu nišu. Primjerice Samsukhaza (2021) za stranice *Emizentech* navodi veoma poznatu tehnologiju za dizajniranje modela građevinskih ili električnih instalacija: *AutoCAD*. Osvrće se i na automobilsku industriju i programske proizvode poput: *Vehicle Safety Software*, *Blender*, *AutoCAD Revit LT Suite*. Također vrlo specifični stolni programski proizvod jest onaj bankarski, a Samsukhaza navodi *Bank of America* (Amit Samsukhaza, 2021).

Jasno je kako stolni programski proizvodi i dalje zauzimaju velik udio na tržištu, jednako kao i tehnologije koje se koriste za njihovu izradu, a mnoge od njih pripadaju velikim imenima poput Microsoft-a, IBM-a, Adobe-a. Kako i blog stranice *Cebu Web Maker* (2018) neosporivo navode, stolne programske proizvode odlikuju sigurnost, mogućnost rada bez potrebe za povezanošću s internetom, lako praćenje napretka i projekata i zaposlenika… Za mnoge tvrtke, upravo su takvi programski proizvodi ono što čini svakodnevni oslonac uspješnog djelovanja na tržištu rada i osigurava stalnu kompetenciju („Cebu Web Maker", 2018)! Mnogi uredi, ali i kućanstva nastavit će koristiti rečene proizvode i u budućnosti, a upravo nas rečena budućnost dovodi do sljedećeg i najbitnijeg potpoglavlja.

2.5. Budućnost stolnih aplikacija i njihovih tehnologija

Najvažnije potpoglavlje teorijskog dijela konačno upotpunjuje prethodna potpoglavlja te razrješava upitnu budućnost stolnih programskih proizvoda i tehnologija koje se koriste za njihovu izradu. Razriješivši prednosti, mane i stolnih i internetskih aplikacija, sagledavši trenutno aktualne tehnologije za njihovu izradu te osvrnuvši se na prošlost i sadašnjost istih sa sigurnošću možemo prijeći na budućnost koju čeka neke od velikih kompanija i njihove proizvode. Uz tržište koje biva sve više preplavljeno internetskim stranicama sa svojim internetskim aplikacijama, stolno programsko inženjerstvo i dalje drži blizak korak i nalazi svoje „mjesto pod suncem". Pitanje jest „Do kada će i na koji način to uspjevati?". Također sagledat ćemo tri posebna primjera tehnologija koje i dan danas stoji rame uz rame jedna drugoj jednako kao i svojim najvećim najvećim internetskim konkurentima. Riječ je naravno o .NET-u, Pythonu i Perlu, Swinga... Preciznije, o budućnosti i razvoju (stolnih) programskih proizvoda u .NET-u, Pythonu, Perlu, Swinga..

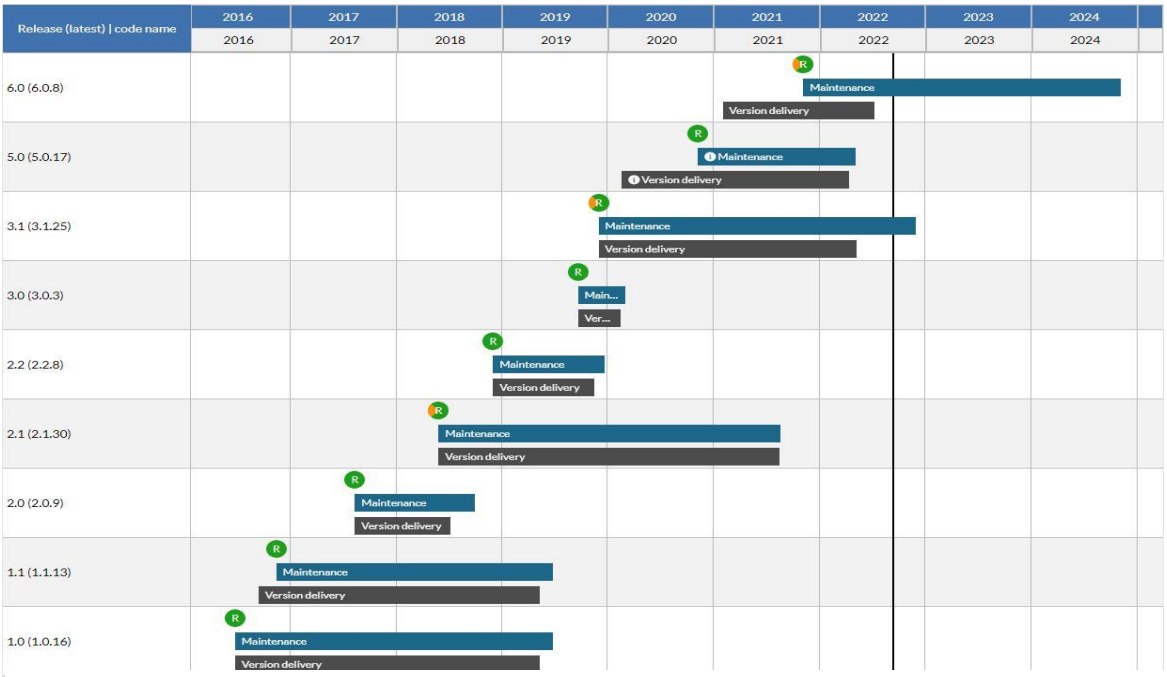
Ukoliko internetom pretražite ključne riječi vezane uz ovu tematiku poput: „*Future of desktop development*", „*Desktop development trends*", „*New desktop development technologies*" ili izvorno hrvatski „Budućnost razvoja (stolnih) aplikacija" doznat će te kako mnogi od portala, blogova i foruma pažnju na svoje domene skreću vrlo jednostavnim, a opet kompleksnim pitanjem: „*Is desktop development dead and or dying?*"…Ne! Rečeno u najkraćim crtama razvoj stolnih programskih proizvoda nije mrtvo, a zasigurno ne njihove tehnologije!

Kako bismo pobliže razumjeli trenutno činjenično stanje tehnologija za razvoj stolnih programskih proizvoda sagledat ćemo jedne od najdominantnijih na tržištu te u njima iznesti njihove općenite informacije te prednosti i nedostatke koje su ih upravo plasirale na pravedno mjesto na tržištu. Tehnologije koje su odabrane, odabrane su na način kojim sagledavamo one koje na tržištu imaju jaku poziciju, ali i one koje su u sve većem opadanju. Elaborirajmo stoga, kao što je i najbolje, na primjerima i to njih pet sljedećim redoslijedom: .NET, Python, Perl, Swing, Rust.

2.5.1. .NET

.NET se nakon, mogli bismo reći, mnogo godina vladavine, kao jedan od najrasprostranjenijih i najcjenjenijih okvira (eng. Framework) za izradu stolnih programskih proizvoda nalazi na velikom križanju. U posjedu Microsofta, i *.NET Framework* i *.NET Core* sve češće se nalaze pred jednakim problemom, a to jest kako držati korak na tržištu na kojemu

su internetski programski proizvodi, jednostavno i kratko rečeno bolji od onoga što oni mogu isporučiti? Naravno, mišljenja o trenutnoj i budućoj relevantnosti .NET-a su kao i za svaku kompleksnu temu: podijeljena. Microsoftov partner, *iFour Techno Lab* (2021) na svojim je stranicama postavio pitanje, koje smo kroz ovaj rad i sami zapitali: „Je li *.NET Framework* mrtav i kakva je njegova budućnost?“. Zaposlenici različitih informatičkih poduzeća ima li svoja mišljenja, u kratkim crtama ovo su jedna od mnogih: „*Da, .NET Framework je mrtav, a mrtav je zato što nikada više neće zaprimiti novo ažuriranje*“, smatra Biran Donovan iz tvrtke *Timeshatter*. Suosnivač stranice *PeopleFinderFree* Eden Cheng iznesao je svoje oprečno mišljenje: „*Ne baš! .NET Framework najbolji je Microsoftov framework za kreiranje web i desktop aplikacija.*“ (iFour Techno Lab, 2021). Teško je izvući jednu zasebnu misao kao nit vodilju za predikciju relevantnosti .NET okoliša, kako u bližoj tako i daljoj budućnosti. Usprkos svemu, jedan od komentara se istaknuo. Katherine Brown (2021), osnivačica i direktorica marketinga tvrtke *Spyic* za *iFour Technolab* blog, objedinila je osobnu procjenu s realnim činjenicama i napisala: „*U 2019. Microsoft je objavio kako će .NET Framework 4.8 biti zadnji .NET Framework. Nadalje objavili su kako će .NET Core 3.0 biti referiran kao 5.0.“* (iFour Techno Lab, 2021). Stranice *Technobrains* također su se uhvatile u koštac s temom budućnosti i novosti glede .NET-a te na svome blogu navele kako .NET Core, odnosno .NET 5.0 za vrijeme pisanja ovog rada je postao 6.0, a ne planira se zaustaviti tek tako, zasigurno ne kada istoj tehnologiji u anketi od 80 000 developera diljem svijeta njih 72% da „palac gore“ (Technobrains, 2022). .NET, kakav god bio, u rukama je vrsnih developera, dok mu leđa čuva Microsoft, a zajedno dijele dugu povijest. .NET također računa na pomoć od strane: *Blazor* (okvira za izgradnju web aplikacija),*Xamarin* (platforme za izgradnju mobilnih aplikacija) i naravno *Unity* (višeplatformski motor za videoigre)..NET čeka (svijetla) budućnost, a vidljivo je iz sljedećeg priloženog infografa gdje se evidentno navodi životni ciklus podrška određenih verzija te učestalost razvoja istih što upućuje na dosljednost i uspješnost navedene tehnologije:



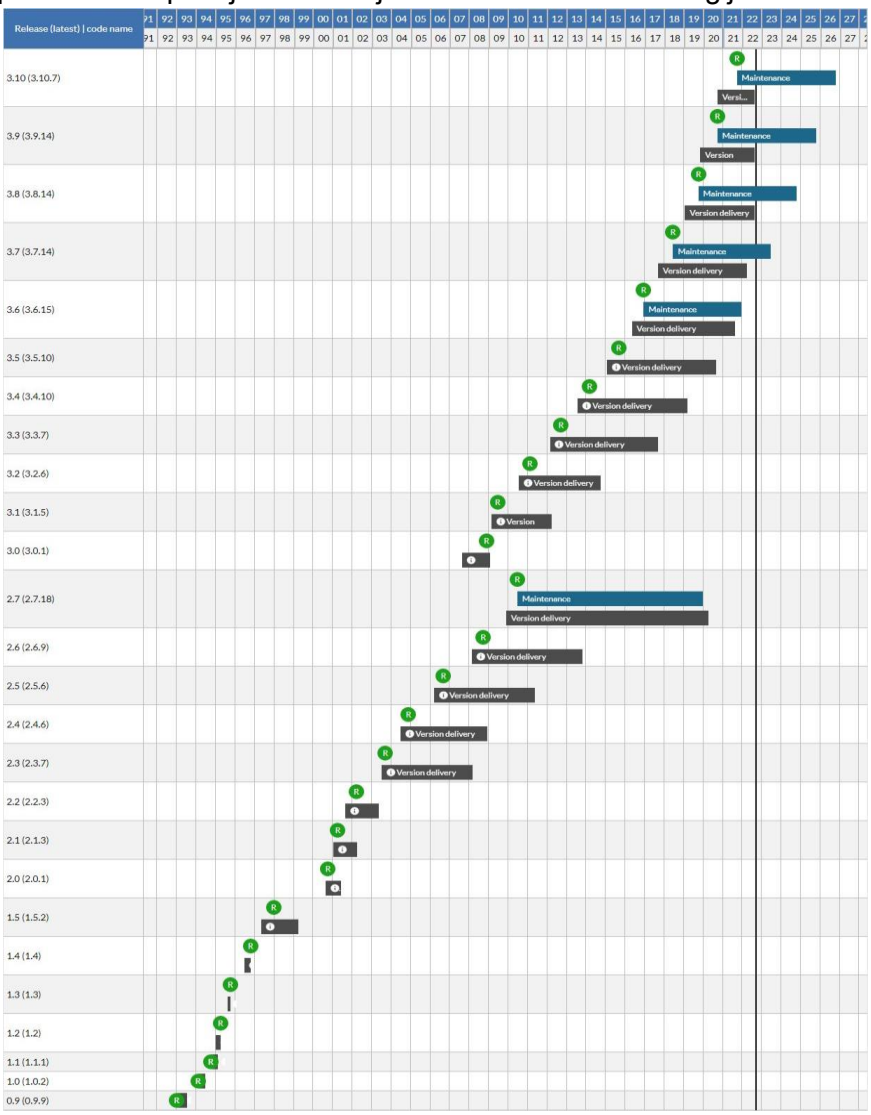
Slika 1: .NET verzije i podrška kroz godine (versio.io 9.9.2022.)

2.5.2. Python

Pretražimo li ključne pojmove na internetu u sljedećem obliku: „Budućnost Pythona“, dobivamo naslove web stranica i portala koje koreliraju sa zadanim pojmovima ali zvuče vrlo, imperativno! „Python, budućnost programiranja“, „Python. Budućnost developmenta“, neki su od takovih naslova, ali imaju i svoje dobre razloge. Vrlo specifične i konkretne odlike Pythona koje će ga (za)držati visoko plasiranog na tržištu i trenutno i u budućnosti za stranice *Medium* naveo je Bokchoy direktor *The code Panda*. Nekoliko od ključnih stavki koje primjećuje Bokchoy (2022) jesu otvorenost koda koja je sama po sebi prednost (nema plaćanja), što stvara i uzročno posljedičnu vezu. Posljedica toga jest veći broj korisnika, što ponovno za posljedicu ima puno više dostupne dokumentacije. U ovaj prilog mu također ide i nevjerojatna relevantnost od tri desetljeća te činjenica da je gotovo 13% svih korisničkih upita na najpoznatijem developerskom forumu *Stack Overflow* upravo bilo vezano uz Python. Više od 80 tisuća radnih mjesta na svijetu pod okriljem je Pythona samo u 2021 (Bokchoy, 2022).

Zapitali bismo se: „Uz sve rečeno, kako je moguće da Python i dalje nije broj 1?“, a razlog je upravo u tome što nismo prosudili sve, točnije: nedostatke i iznimke Pythona. Dva najočitija nedostatka vezana su uz same performanse Pythona, Ashwin Joy (2021) na stranicama *Pythonisaplanet* iz osobnog iskustva navodi nekolicinu nedostataka kao što su sporost prilikom pokretanja (radi uporabe interpretera umjesto prevoditelja) te visoku uporabu memorije (radi uporabe interpretera i neskladnog rada sa sakupljačem smeća). Ostala dva

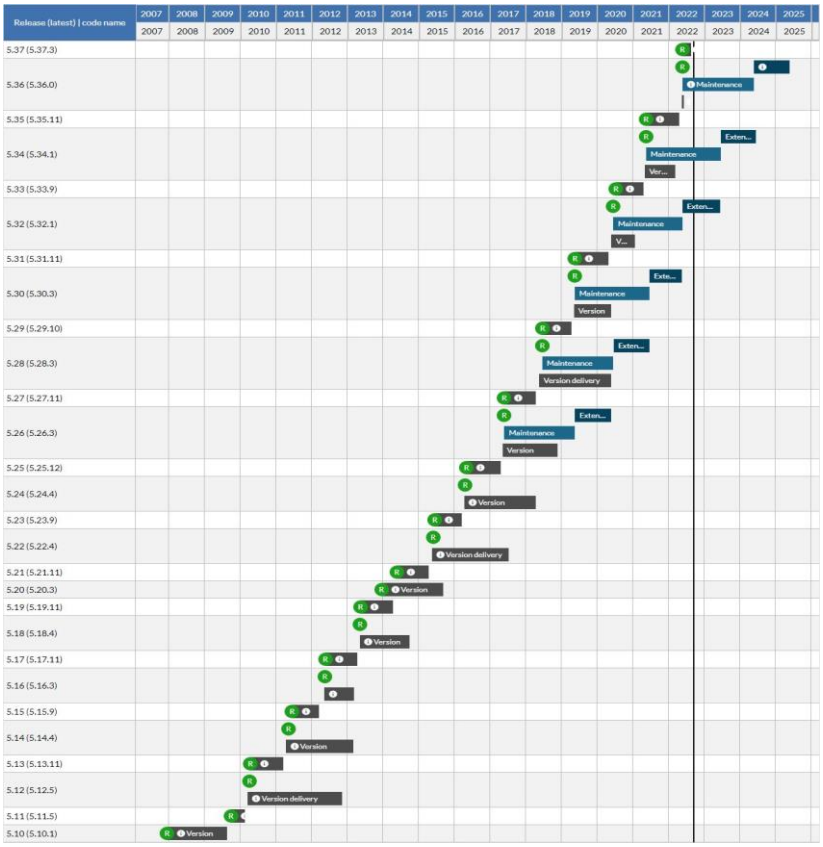
nedostatka znatno su više vezana uz ljudski sektor. Naime Joiy (2021) navodi kako mnogi Python developeri nisu odviše željni programirati u više drugih jezika. Stigma koja prati Python jest da on slovi kao jedan od najlakših programskih jezika, iako je to zaista subjektivno stajalište i ljepota razvoja (programskih) rješenja je u tome da do jednog možete doći na milijun, doslovce, načina. Nadalje, rijetko koje tvrtke će koristiti isključivo Python za izgradnju programskih proizvoda te će se radije osloniti na nešto poput Jave, C++-a ili PHP-a (Ashwin Joy, 2021). Ovo ne znači da Python ne nalazi svoje mjesto u velikim tvrtkama, dapače na osobnom primjeru mogu potvrditi da nije ostao zakinut. Naime tvrtka o koja je posrijedi (zaslužna za obavljanje moje ljetne prakse) koristi Python (Selenium) za automatizirano pokretanje / provjeru testnih slučajeva za slanje raznih notifikacija. Zaključak: ako je sudeći prema svemu do sada rečenom, Python ne samo da ima obećavajuću budućnost, već jednu od najviše obećavajućih, a mi smo ovdje kako bismo u njoj sudjelovali! Isto je vidljivo iz sljedećeg infografa, jasno prikazanog razvoja prijašnjih i budućih verzija Pythona sa svojim dugoročnim podrškama upućujući na dosljednost i kvalitetu tehnologije:



Slika 2: Python verzije i podrška kroz godine (versio.io 9.9.2022.)

2.5.3. Perl

Pušten u široku primjenu krajem prošloga stoljeća, *Perl* pred ostalim značajnijim tehnologijama, jezicima i okvirima pada u peti plan. Razlog tomu nije jedino njegova starost već razni detalji koji ga u usporedbi s aktivnim / atraktivnim gigantima poput Jave i Pythona (s kojim je vječito uspoređivan) čine toliko manje primamljivim, razumljivijim, korištenim. Ubrojimo u to varijablu proteklog vremena i dobivamo recept za propast! Najveće zamjerke s kojima nas je na stranicama *Tecmint* upoznao Saive (2016) jesu Perlova mogućnost da jedno programsko rješenje ostvari na stotine načina. Isprva se može činiti kao prednost, no zapravo je vrlo velika smetnja jer isto dovodi do nepreglednosti te nerazumijevanja za ostale korisnike koji bi koristili potpuno drugačije pristupe. Također mnogo je sporiji, a u prilog mu zasigurno ne ide i njegova ne prenosivost na više platformi što je jedna od ključnih stavki modernih tehnologija i aplikacija (Ravi Saive, 2016). Nažalost Perl nije jedini ovakav slučaj, no onaj je kojega smo istaknuli radi stalnih usporedbi s Pythonom (kojeg smo također obradili). Zaključak: Perl uz znatno snažniju konkurenciju ne drži dovoljno jak korak te uz veliki vremenski period koji je proveo na tržištu nije uspio zadovoljiti sve više rastuće potrebe stoga je njegova popularnost i reputacija u znatnome padu. Vidljivo je to i iz sljedećeg infografa gdje je evidentno kako verzije Perla nema jednako mnogo kao Pythona te verzije nisu ni izbliza dugo opstale kao one .NET-a:



Slika 3: Perl verzije i podrška kroz godine (versio.io 9.9.2022.)

2.5.4. Swing

Nažalost, Swing je jedna od tehnologija čija se pozicija ne razlikuje znatno od one opisane u slučaju Perla. Također ga krasi dugoročna prisutnost na tržištu budući da se na istome pojavio krajem 90ih prošloga stoljeća. Java Swing unazad prošlih petnaestak godina, od pisanja ovoga rada doživjela je što nijedna tehnologija, proizvod ili jezik ne žele doživjeti, a to je krah. Veoma jasne i elaborirane razloge naveo je Ben Wilson za stranice *Vaadin* bloga pod iznimno zanimljivim naslovom *Technical Erosion and Java Swing*. Wilson (2019) spominje najrelevantniju funkcionalnost i potrebu današnjih aplikacija, prenosivost. Naime riječ je o nemogućnosti prijenosa aplikacija izrađenih putem Java Swinga s radnih površina poslovnih računala na neke druge površine poput onih osobnih prijenosnih računala. Ovaj problem usko je vezan uz onaj koji se tiče instaliranja preduvjeta za samo korištenje Swinga koji su u začecima bili dostupni na spomenutim poslovnim računalima. Problem oko prikaza Swing aplikacija na različitim uređajima ovdje ne prestaje, naime Wilson (2019) objašnjava kako s razvijanjem zaslona s visokim DPI-jem, Swing aplikacije postalo je doslovno nemoguće koristiti na Windowsu i Linuxu. Zadnji veliki nedostatak koji je utjecao na neuspjeh Swinga jest vezan uz vještine osoblja koje se njime bavi. Preciznije riječ je o tome da su mnoge tvrtke od svojeg Swing osoblja zahtijevale poznavanje i mogućnost rada u vrlo specifičnim bibliotekama koje su razvile za vlastitu nišu, što naravno nije uvijek bio slučaj te je često stvarao velike probleme za developere koji se izvorno koriste Swingom (Ben Wilson, 2019). Swing će zbog ovih i mnogih drugih razloga doživjet sve veći pad na industriji te će od aplikacija u njemu izrađenih zahtijevati migracije u nešto više obećavajuće, službeno do 2026. Oracle jest naveo kako će upravo do ove godine pružati *support* Swingu, nakon 2026. upitno je što će ga zadesiti. Zaključak: uz znatno snažniju konkurenciju ne drži dovoljno jak korak te uz veliki vremenski period koji je proveo na tržištu nije uspio evoluirati u proizvod za koji su se mnogi korisnici nadali kako bi i dalje bio dovoljno koristan, a time i više relevantan. Prilog ovome ide i dodijeljeni infograf:

Java SE Release	Azul Lifecycle Title	Java SE GA Date	Oracle End of Free Commercial Use	Oracle Java SE	Azul Platform Core		Azul Platform Prime
					Azul Zulu Builds of OpenJDK	Eclipse Temurin	Azul Zulu Prime Builds of OpenJDK
CURRENT OFFERINGS							
Java 6	LTS	Dec 2006	Apr 2013	Dec 2018 (12 yrs)	Dec '18 / Dec '27* (12 yrs) / (21 yrs)*	-	Apr 2017 (11 yrs)
Java 7	LTS	Jul 2011	Apr 2015	July 2022 (11 yrs)	Jul '22 / Dec '27* (11 yrs) / (16 yrs)*	-	Dec 2021 (10.5 yrs)
Java 8	LTS	Mar 2014	Jan 2019	Dec 2030 (16 yrs)	Dec 2030 (16 yrs)	May 2026 (12 yrs)	Dec 2030 (16 yrs)
Java 9	6 months	Sept 2017	Mar 2018	Mar 2018 (6 mo)	Mar 2018 (6 mo)	-	-
Java 10	6 months	Mar 2018	Sept 2018	Sept 2018 (6 mo)	Sept 2018 (6 mo)	-	-
Java 11	LTS	Sept 2018	Mar 2019	Sept 2026 (8 yrs)	Sept 2026 (8 yrs)	Oct 2024 (6 yrs)	Sept 2026 (8 yrs)
Java 12	6 months	Mar 2019	Sept 2019	Sept 2019 (6 mo)	Sept 2019 (6 mo)	-	-
Java 13	MTS	Sept 2019	Mar 2020	Mar 2020 (6 mo)	Mar 2023 (3.5 yrs)	-	Mar 2023 (3.5 yrs)
Java 14	6 months	Mar 2020	Sept 2020	Sept 2020 (6 mo)	Sept 2020 (6 mo)	-	-
Java 15	MTS	Sept 2020	Mar 2021	Mar 2021 (6 mo)	Mar 2023 (2.5 yrs)	-	Mar 2023 (2.5 yrs)
Java 16	6 months	Mar 2021	Sept 2021	Sept 2021 (6 mo)	Sept 2021 (6 mo)	-	-
Java 17	LTS	Sept 2021	Mar 2022	Sept 2029 (8 yrs)	Sept 2029 (8 yrs)	TBD	Sept 2029 (8 yrs)
Java 18	6 months	Mar 2022	Sept 2022	Sept 2022 (6 mo)	Sept 2022 (6 mo)	-	-
FUTURE OFFERINGS (EST.)							
Java 19	MTS	Sept 2022	Mar 2023	Mar 2023 (6 mo)	Mar 2025 (2.5 yrs)	-	Mar 2025 (2.5 yrs)
Java 20	6 months	Mar 2023	Sept 2023	Sept 2023 (6 mo)	Sept 2023 (6 mo)	-	-
Java 21	LTS	Sept 2023	Sept 2026	September 2031 (8 yrs)	September 2031 (8 yrs)	TBD	September 2031 (8 yrs)

Slika 4: Java verzije i podrška kroz godine (Azul.com 2022.)

2.5.5. Rust

Rust se na tržištu pojavio 2010. godine, no par godina kasnije, preciznije 2016. njegova se popularnost lansirala! Naime korisnici najpoznatijeg developerskog foruma „Stack OverFlow“ od navedene 2016. godine do godine pisanja ovoga rada (2022.) izabrali su upravo Rust kao najdraži programski jezik. Također pretražimo li na internetskom pregledniku samo neke od ključnih riječi poput „Rust future“, „Future of Rust“, ili jednostavno „Rust“, upoznati smo s mnoštvom foruma, blogova koji govore upravo o navedenim dobrim stranama *Rusta* i njegovom sve više rastućem uspjehu i popularnosti. Daria (2021) za stranice *Scalac* kao

distinktne prednosti predstavlja sigurnost (u usporedbi s C++) pisanjem takozvanog *unsafe* koda sa zadanim *safe* kodom kao rezervom, brzinu (u usporedbi s Javom), jednostavnost kada je u pitanju pronalaženje i ispravljanje grešaka (u usporedbi s Pythonom). U slučaju da su navedene stavke nedovoljno uvjerljive, vrijedi spomenuti kako je Microsoft sve više počeo koristiti upravo Rust u nekim od svojih infrastruktura (Daria Karasek 7.6.2021.). Zaključak: sudeći po svemu rečenom uz konstantno rastuću publiku *Rust* očekuje jedna od najsvjetlijih budućnosti

Sagledavši različite sudbine i trenutna stanja onih popularnih i nekad popularnih tehnologija, programskih jezika i okvira osvrnimo se još jednom na stolni razvoj programskih rješenja kao takvo u cijelosti! Uz ovako heterogeno, a opet dovoljno ujednačeno tržište stolnih tehnologija i stolnih aplikacija, ne postoji dovoljno eksplicitan i transparentan dokaz za njihov nestanak. Drugim riječima mogli bismo reći kako smo opovrgnuli prethodno predstavljenu stigmu u uvodu ovoga rada! Tehnologije za izradu stolnih programskih proizvoda nastavit će se koristiti, baš kao i njihovi proizvodi, Upendra Jith (2021) za stranice *Bridge-global* navodi u čemu:

a) Poslovnome softveru: velike količine osjetljivih podataka koje neke tvrtke prikupljaju,

b) Softver niske razine: upravljački programi,

c) Poduzetničke aplikacije: ERP, DAS, CAD rješenja,

d) Videoigre.

Upendra Jith (2021). Primijetili smo da je činjenično stanje kako se sve više tehnologija bazira na web-u upravo radi veće i lakše pristupačnosti krajnjim korisnicima, a sukladno tome mnoge stolne aplikacije migriraju na web ili ostaju zaboravljene radnim površinama. Usprkos tome, postoje i one koje su oduvijek bile i bit će stolne aplikacije, ali i tehnologije koje nam tek dolaze. Sasvim je uobičajeno da svaki proizvod, trend, ali i tehnologija imaju svoj životni ciklus, a mi smo nabrojavši pregršt primjera (općenitih i specifičnih) te istraživši budućnost, sadašnjost i prošlost istih dokazali kako postojanost tog životnog ciklusa stolnih aplikacija i tehnologija nisu postali apstrakcija, već su i dalje sveprisutna stvarnost koja pogoni većinu IT industrije. Vrijeme je da isto učinimo na praktičnome primjeru koji je razrađen u drugome dijelu razrade, a izrađen je u jednoj od najpopularnijih novijih tehnologija korištenih za izradu „cross-platform“ aplikacija.

Shodno dispoziciji rada sljedeće potpoglavlje ujedno je i zadnje potpoglavlje teorijskog dijela, a tiče se naravno izabrane tehnologije za razradu praktičnog rješenja problema naloženog ovim završnim radom. Tehnologija izabrana za postizanje ovoga cilja jest *Flutter*.

2.6. Flutter

Flutter sloganom „*Build apps for any screen*“ jasno istaknutim na svojim početnim stranicama tvrtke, na tržište se probio kao jedan od najpopularnijih novih ekosistema za stvaranje aplikacija. „Dijete“ mega korporacije Google, *Flutter* se na tržištu prvi puta pojavio u svibnju 2017. kao *UI framework* za mobilne uređaje („Flutter (software)“, bez dat.). Ono što je tada *Flutter* učinilo drugačijim od drugih, on i danas nalazi kao svojevrsnu prednost i distinkciju zbog koje sve više developera ili na njega prelazi u potpunosti ili planira koristiti u budućnosti do neke mjere. Neke od karakteristika koje „Altexsoft Inc“ navodi na stranicama svojega bloga jesu: otvoreni kod (odnosno *open source*), *cross platform development*, a ono najpoznatije jest korištenje *Dart* programskog jezika kao jezika za pisanje programskih proizvoda i „*Flutter engine*“ u C++-u. Korištenje *Dart* jezika opravdano jest na stranicama Wikipedije gdje je riječ kako se naše aplikacije puštaju kroz „Dart virtualnu mašinu“ što nama povratno omogućuje brže kompajliranje te tzv. „*Hot reload*“. *Hot reload* još jedna funkcionalnosti koju *Flutter* koristi kao prednost nad drugima, a ono nam omogućuje da unosimo promjene u svoj kod za vrijeme kada je aplikacija pokrenuta, uživo vidimo promjene te nema potrebe za ponovnim pokretanjem programa („Flutter (software)“, bez dat.). *Flutter engine* ono je što nam služi za implementiranje *Flutter* biblioteka te za povezivanje s SDK-ovima ostalih platformi.

Iako prvotno osmišljen za izradu aplikacija za mobilne uređaje, *Flutter* se naravno proširuje i na ostale platforme, a to planira nastaviti i u budućnosti. Ovdje je dakako riječ o „*desktop*“ i „*web*“ platformama. Na prvi pogled mnogima bi se učinilo kako *Flutter* i nije najnovija, „*cutting edge*“ tehnologija, no tu bi bili u krivu! Naime *Flutter* je upravo nedavno objavio svoju prvu stabilnu verziju za razvoj stolnih programskih proizvoda u Windows ozračju (samo šest mjeseci od pisanja ovog rada). Točnije riječ je o veljači 2022., kada je i Tim Sneath (2022), direktor UI / UX za *Flutter i Google*, objavio vijesti na mnogim utjecajnim stranicama, kao što su *Medium*“ i „*Linkedin*“ pod naslovom „*Announcing Flutter for Windows*“!

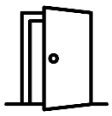
Sagledajmo stoga neke od odlika! Voditelj projekata *Flutter* tima, Sells (2022) u kratkim crtama u svojem prezentacijskom video uratku za prvu stabilnu verziju Fluttera za Windows, pod istoimenim nazivom poseban naglasak stavlja na sljedeće bitne osobine:

Prenosivost: već spomenuta mogućnost izrade „*Cross-platform*“ aplikacija, ne samo u vidu Windows / MacOS, već mobilni uređaji, web aplikacije na web stranicama…

Ljepota: jednostavnost i upotreba mnoštva aktualnih *widgeta*, „*Flutter*“ developerima omogućuje izradu modernih UI-a, što znači da korisnici istih uživaju u raznim navigacijskim obrascima, fontovima, paletama boja…

Brzina: prethodno spomenuto korištenje virtualne mašine *Darta* kao i korištenje samog programskog jezika, *Flutter* čine brzim… bržim i od *Native Reacta*. Na odmet nije i njegova najveća posebnost, jedna jedinstvena baza koda.

Produktivnost: mogućnost *Fluttera* koja se dopala mnogim developerima jest *Hot reload*, naziv koji predstavlja sposobnost da promjene u kod vidimo „*in real time*“ putem aplikacije koja nam je konstantno pokrenuta. Također ovdje možemo ubrojiti i jednu bazu koda.



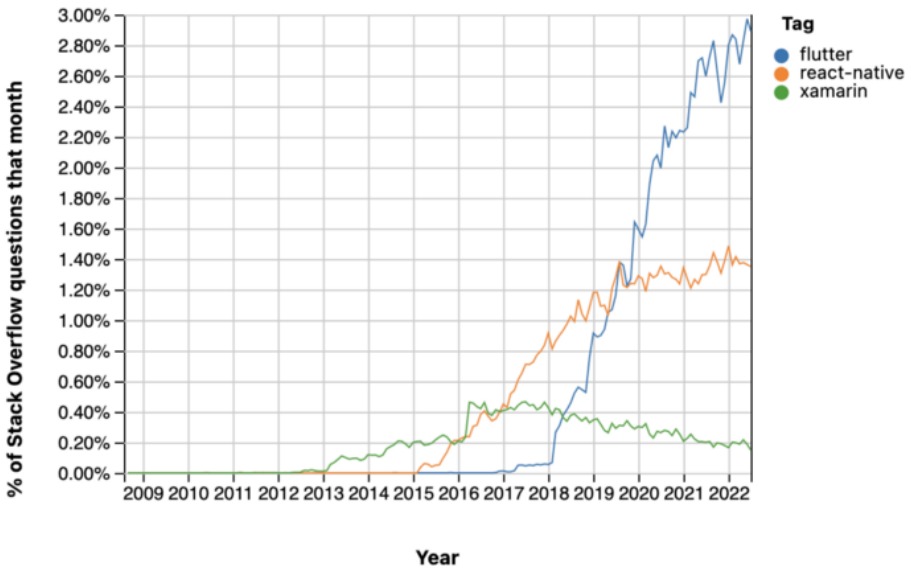
Otvorenost: *Flutter* je dakako besplatan te na izgradnji i uređenju njegova okoliša pomaže i grupno surađuje tisuće ljudi od kojih nisu svi zaposlenici *Google* i *Flutter* tima što pokazuje njegovu rasprostranjenost te predanost publici.

Flutter krase i visoke CPU performanse, a ne smijemo zaboraviti i na Google koji je u pozadini svega kao svojevrsno jamstvo kvalitete!

Nabrojavši sve prednosti, pozitivne aspekte i odlike *Fluttera* koji *Flutter* čine onim što jest, na pamet pada pitanje: „Pa zašto sve fizičke i pravne osobe, mikro, male, srednje i velike firme ne koriste *Flutter* u izradi svojih programskih proizvoda?“ Svakako je više toga posrijedi nego je na prvi pogled vidljivo, stoga je naša dužnost pokazati kako ni u slučaju *Flutter* nije sve crno i bijelo! Upravo neke od nabrojanih prednosti *Fluttera* zapravo se očituju kao jedne od njegovih većih mana, a riječ je o veličini samog programskog proizvoda. Berka (2020) ukratko navodi neke od nesavršenosti *Flutter* aplikacija kao: teške aplikacije (najmanje veličine znaju biti i 4MB, što je više nego one izrađene u *Native Javi*), sljedeća stavka jest sam *Dart* ne spada u grupu najrazvikanijih jezika, mnogi od developera (kao i ja sam) susreću se

s njim prvi puta, pa upravo ta činjenica otežava rast *Flutter* publike s developerske strane, što nije začuđujuće. Ovaj nedostatak za posljedicu ima direktnu povezanost s našim zadnjim, većim nedostatkom, a riječ je o manjoj dostupnosti kvalitetnih izvora informacija, primjera i tutoriala, čemu u prilog podilazi i vrlo kratko vrijeme postojanja *Fluttera* (Maciek Berka, 2020).

Zašto je baš *Flutter* izabran za izradu proizvoda i opovrgavanje spomenute stigme u uvodu ovoga rada te razrješavanje problema koji nam se nameće?! Osobni izbor ove tehnologije proizašao je iz njezina navedenog snažnog i bogatog sučelja, sve više rastuće publike. Nadalje ovaj rad zahtjeva korištenje jedne od novijih tehnologija, a spomenuli smo činjenicu da je *Flutter* na tržištu tek pet godina od čega je ove godine (2022) objavio prvu stabilnu verziju za Windows! Također smatram kako je „Flutter“ pored „Reacta“, „Angulara“, Jave ili Pythona prošao pomalo neopažen stoga ovaj rad može poslužiti kao svojevrsno upoznavanje s dotičnom tehnologijom. Pronašao sam i veliku korelaciju s razlozima zbog kojih biste željeli koristiti *Flutter* koje je naveo Berka (2020) kao što su slučajevi kada želimo brzo razviti aplikaciju, ugodno ali i složenije korisničko sučelje te dobre performanse aplikacije (Maciek Berka, 2020). Zanimljivo je kako je *Flutter* postao najveći konkurent *React Nativeu* koji će i dalje biti izbor mnogih developera kada je cilj izraditi specifično *native* programsko rješenje za jednu platformu, no *Flutter* postaje sve veći „trn u oku“, kako developera tako i React ravojnog tima, a to je poduprto sljedećim grafom popularnosti:



Slika 5: Pregled udjela upita na Stack Overflowu za Flutter, React-native, Xamarin (Stack Overflow Trends, 2022)

No nemojmo stati samo na ovome, usporedimo *Flutter*, a zajedno s njim i *Dart* s nabrojenim konkurentima na tržištu, kako tehnologijama tako i programskim jezicima u sljedećem potpoglavlju.

2.7. *Flutter* protiv ostatka tržišta

Parametri usporedbe o kojima ćemo raspraviti tiču se *Fluttera* kao tehnologije, ali i samog *Darta* programskog jezika u pozadini svega, a razlog tomu je upravo taj što je nešto poput .NET-a tehnologija dok je nešto poput *Rusta* programski jezik. Parametri koje ćemo opisati su sljedeći:

a) Popularnost

b) Dostupnost dokumentacije i ostalih informativnih materijala,

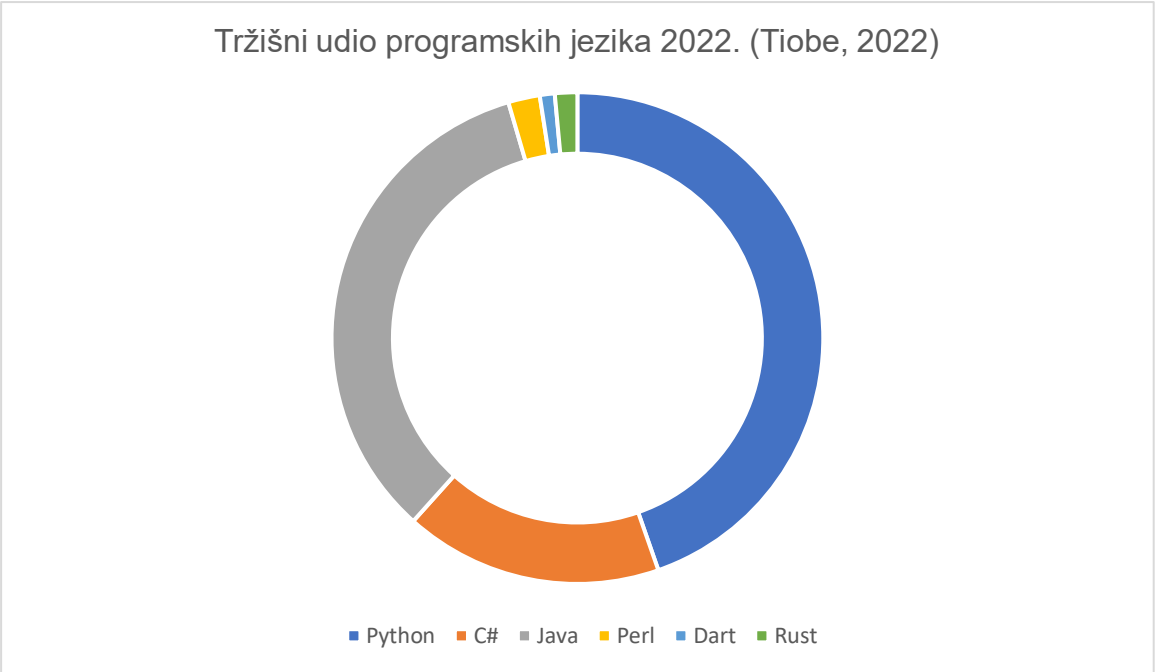
c) Tehnički aspekti (kompleksnost, sigurnost, potrebni resursi, veličina programa)

Popularnost:

.NET svoju popularnost i dalje ostvaruje zahvaljujući strateškom potezu spajanja *.NET Corea* i *.NET Frameworka* koju smo već spomenuli. Gadgil (2022.) za blog stranice *Clariontecha* navodi upravo lagan porast u broju upita na *Stack Overflowu* vezanih uz .NET. Iako porast nije velik, nije ni beznačajan, preciznije riječ je o trećini svih upita na najpopularnijoj i najvećoj zajednici za doprinos rješavanju tehničkih izazova, Stack Overflow (Shalaka Gadgil, 2022). Ipak je. Osobno iskustvo pak dopušta mi da zamijetim kako se C#, odnosno time *Visual Studio*, odnosno time .NET tehnologija podučava u barem četiri sveučilišna i dva veleučilišna studija u Republici Hrvatskoj. Nažalost Python i njegova popularnost ipak su bez premca, stranice *Tiobe* (2022.) tako navode kako je njegov tržišni udio najveći, a iza sebe ostavlja C s na drugom mjestu te spomenuti C# na petome mjestu (Tiobe.com, 2022). Nadalje kada je riječ o Javi, ona svoje mjesto nalazi između C# i Pythona, na sigurnom trećem mjestu dok pak Perl zauzima tek 19. poziciju (Tiobe.com, 2022). Tablica i grafički prikaz (eng. *Pie Chart*) u nastavku precizno prikazuju navedene tržišne udjele:

Tehnologija (naziv)	Tržišni udio (%)
Python	15,74
C#	4,8
Java	11,7
Perl	0,7
Dart	0,34
Rust	0,51

Tablica 1: Tržišni udio tehnologija (Tiobe.com, 2022.)



Slika 6: Tržišni udio programskih jezika (Tiobe, 2022)

Da ipak nije sve tako crno za „naš“ *Dart* pokazuju činjenice koje nalazimo na stranicama *Stack Overflow Insights* (2022.) u objavljenim rezultatima pregleda mnogih developerskih preferenca za 2021. godinu u kojemu je sudjelovalo otprilike 81 000 developera iz cijelog svijeta. Programski jezici sastavljeni pod stavkom „Voljen ili užasavan“ (eng. Loved vs Dreaded) imali su sljedeći raspored (Insights Stackoverflow, 2021.):

Tehnologija (naslov)	Broj „obožavatelja“	Broj „mrzitelja“	Plasman
Python	26 992	12 800	2.
C#	14 241	8743	4.
Java	13 749	15 413	5.
Perl	730	1298	6.
Dart	3166	1799	3.
Rust	5044	755	1.

Tablica 2: Broj obožavatelja i mrzitelja (Insights.Stackoverflow.com, 2021.)

Uviđamo kako popularnosti *Flutteru* i *Dartu* ipak ne nedostaje, a stavka popularnosti usko je vezana uz sljedeću stavku usporedbe, a to je dostupnost dokumentacije i informativnih materijala.

Dostupnost informativnih materijala:

Jednostavnom pretragom nad repozitorijem online tutorijala *Udemy* ustvrdit ćemo dostupnost edukacijskih materijala dok ćemo onom nad *Stack Overflowom* (otprilike) ustvrditi broj relevantnih upita koje korisnici, developeri i oni koji se tako osjećaju mogu koristiti u svrhu grupnog rješavanja pojedinačnih ili grupnih problema. Naravno spomenimo kako svaka od tehnologija i svaki od jezika o kojima se informiramo ima i svoju dokumentaciju koja se izrađuje od strane samog proizvođača stoga tu dokumentaciju, poput *Python docs* ne ubrajamo. Pretražimo li tako *Udemy* (2022) ključnim riječima dobivamo sljedeće rezultate precizno prikazane u tablici i grafički (Udemy.com, 16.9.2022.):

Tehnologija (naziv)	Broj *Udemy* tečajeva	Plasman
Python	10 000	1.
C#	6500	2.
Java	10 000	1.
Perl	100	5.
Dart	650	3.
Rust	200	4.

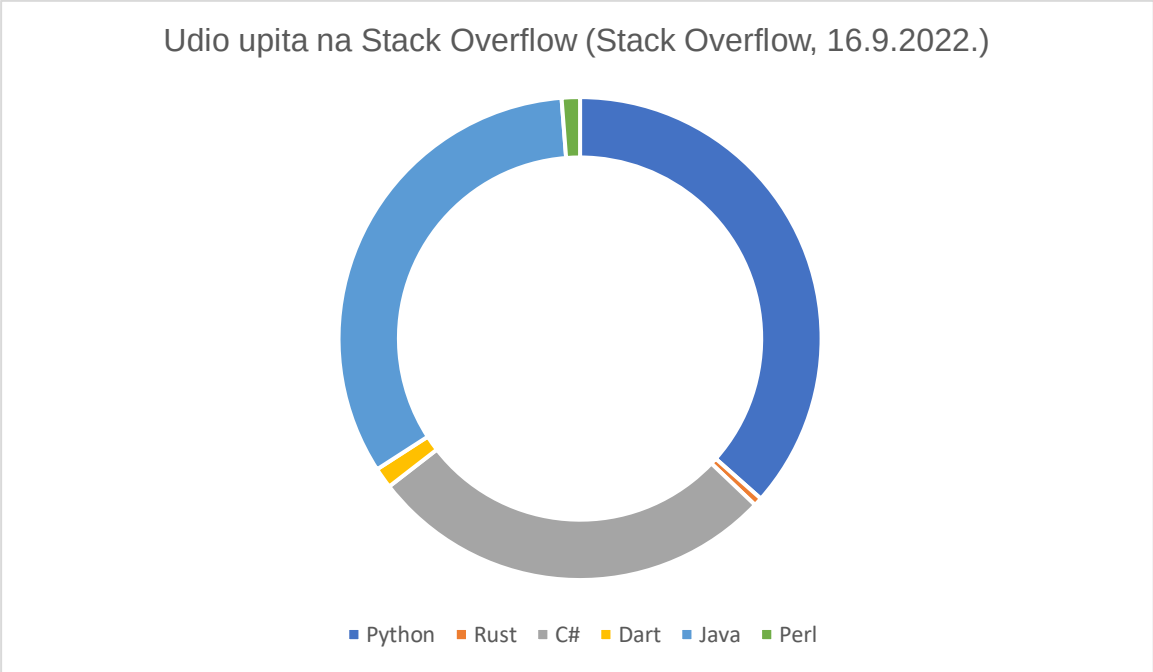
Tablica 3: Broj Udemy tečajeva (Udemy.com, 16.9.2022.)

Pretražimo li pak Stack Overflow (2022) jednakim ključnim riječima možemo vidjeti sljedeće precizno navedene rezultate, ponovno tablično i grafički (Stack Overflow, 16.9.2022.):

Tehnologija (naziv)	Broj *Stack Overflow* upita	Plasman
Python	2 000 000	1.
C#	1 500 000	3.

Java	1 800 000	2.
Perl	67 000	5.
Dart	76 000	4.
Rust	31 000	6.

Tablica 4: Broj Stack Overflow upita (Stackoverflow.com, 16.9.2022.)



Slika 7: Udio upita na Stack Overflow (Stack Overflow, 16.9.2022.)

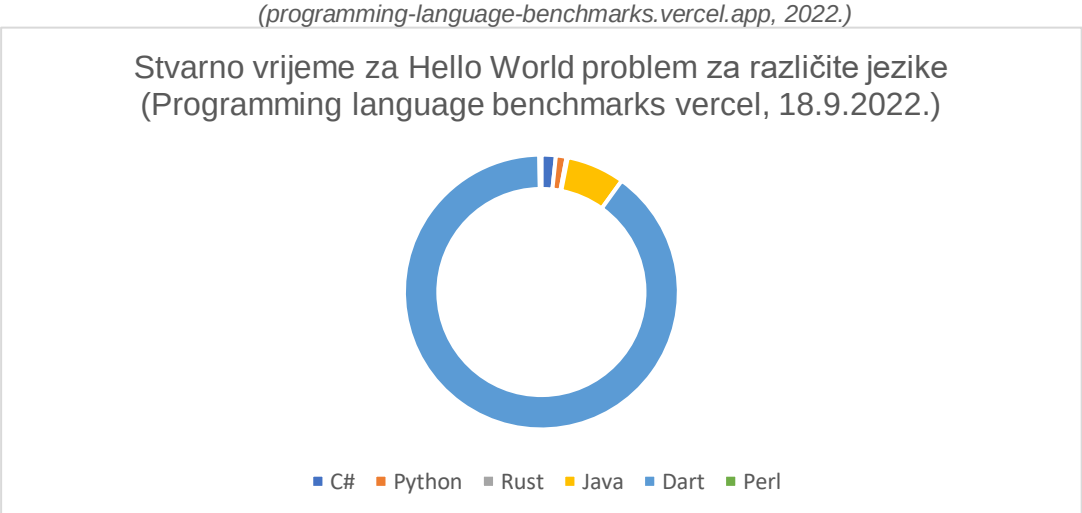
Tehnički aspekti:

Izabrani aspekti odnose se na dvije ključne stvari kada je riječ o performansama i izvođenju programa te korištenju tehnologije. Ključne stvari su tako stvarno vrijeme izvođenja (eng. *Real time performance*) i korištenje memorije (eng. *Peak memory usage*). Dva tehnička aspekta, parametra uzeta su kao predstavnici uspješnosti izvršavanja različitih problema upravo iz razloga njihove relevantnosti kada je u pitanju brzina i kvaliteta izvođenja te korištenje resursa, u ovome slučaju memorije. Svi podaci su preuzeti sa stranice *programming-language-benchmarks.vercel.app* gdje su prikazani mnogi od problema, odnosno programskog koda koji je služio za testiranje, a mi ćemo iskoristiti četiri. Klasični ispis poruke „*Hello World*“ bit će nam prvi parametar usporedbe, a razlog tomu je njegova ustaljenost u programerskom svijetu kao prvog ispisa na ekranu. Stvaranje binarnog stabla bit će naša sljedeća usporedba, razlog tomu je naravno njezino korištenje rekurzija kao složene

i važne strukture podataka što može uvelike dočarati mogućnosti određene tehnologije i jezika u rukovanju s repeticijom koda i matematičkim izrazim. Vrlo zanimljiva usporedba jest ona FASTA format programa (softverskog paketa za usklađivanje sekvenci DNK i proteina) koju smo iskoristili iz razloga što se u njemu koriste veoma opširne baze podataka odnosno pretraživanje i usporedba zapisa istih, što nam ukazuje na mogućnosti tehnologije i jezika kada je riječ o rukovanju s velikim količinama podataka. Zadnja točka usporedbe jest svima poznati broj Pi (π), to jest ispis mnogo znakova u nizu, njih 4000, što će nam dakako još jednom ukazati na brzinu izvođenja programa u određenoj tehnologiji kada je riječ o rukovanju s neprekinutim ispisom. U nastavku tabličnim i grafičkim prikazom, navedene su vrijednosti koje *Programming language benchmarks vercel* (2022) navodi na istoimenim stranicama:

Tehnologija (naslov)	Hello World (ms)	Binary Tree (ms)	Fasta (ms)	4000 mjesta znamenke Pi (ms)	Plasman
		Input: 15	Input: 250 000	Input: 4000	
Python	14	126	332	652	3.
C#	19	153	85	1271	4.
Java	72	167	139	1344	5.
Perl	2,7	2233	458	/	6.
Dart	1007	101	65	427	2.
Rust	1,6	141	27	393	1.

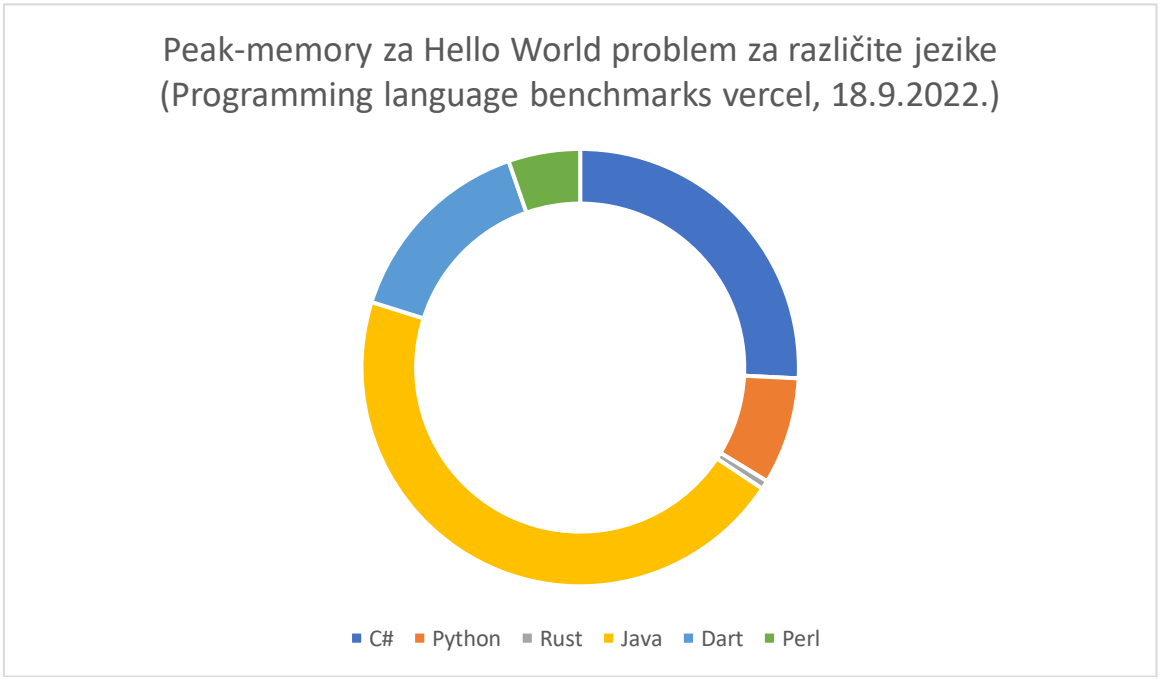
Tablica 5: Stvarno vrijeme izvršavanja programskih problema (programming-language-benchmarks.vercel.app, 2022.)



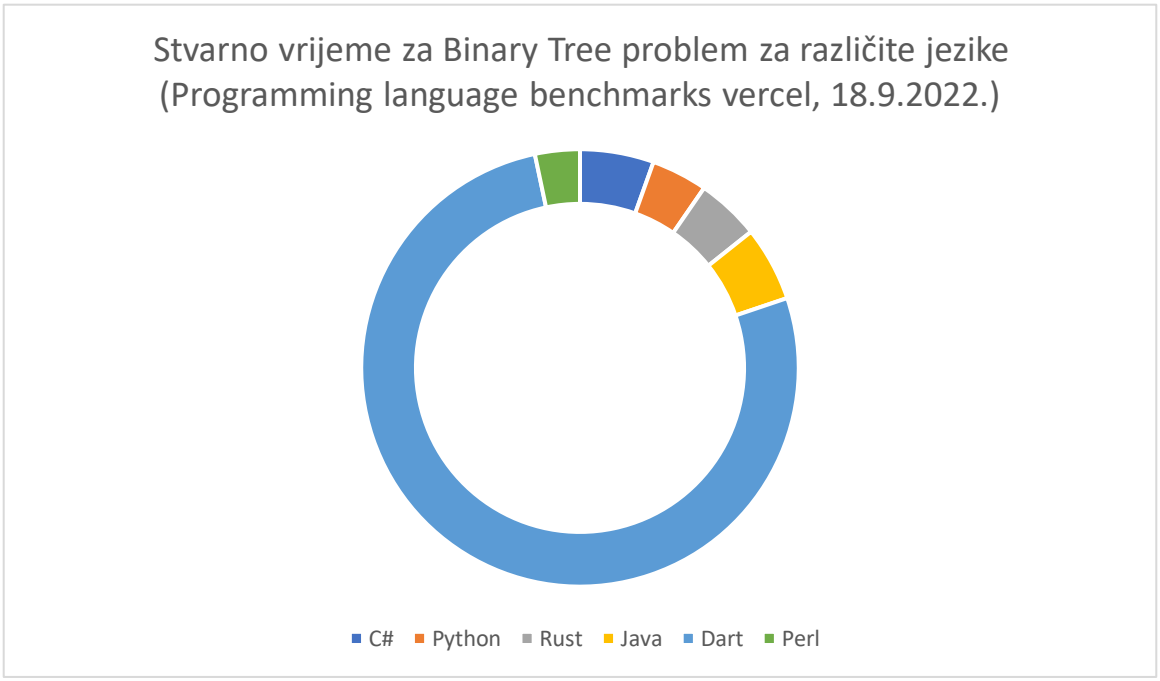
Slika 8: Stvarno vrijeme za Hello World problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)

Tehnologija (naslov)		Hello World (MB)	Binary Tree (MB)	Fasta (MB)	4000 mjesta znamenke Pi (MB)	Plasman
			Input: 15	Input: 250 000	Input: 4000	
Python		7,3	83,2	81,8	84,5	5.
C#		24,3	44,8	36,8	96,5	4.
Java		61	111,2	43	378,3	6.
Perl		5	14,3	6,7	/	2.
Dart		12,2	47,4	14,3	19,9	3.
Rust		0,6	5,7	0,7	2,6	1.

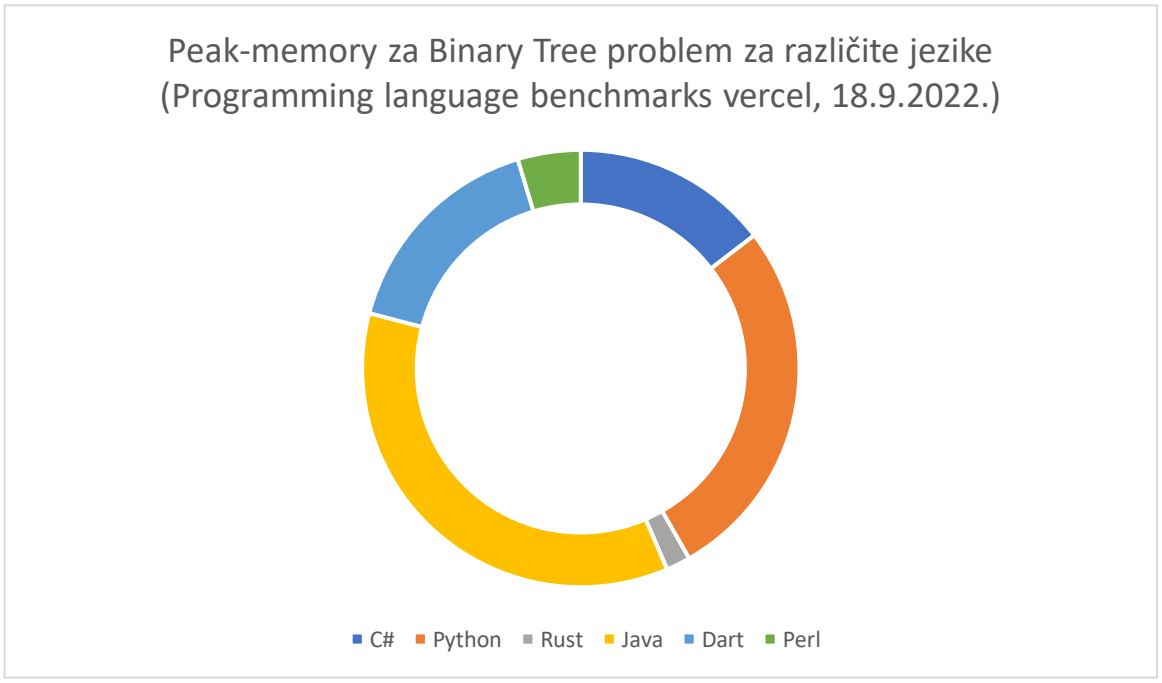
Tablica 6: Ukupno korištenje memorije za izvršavanje programskih problema (programming-language-benchmarks.vercel.app, 2022.)



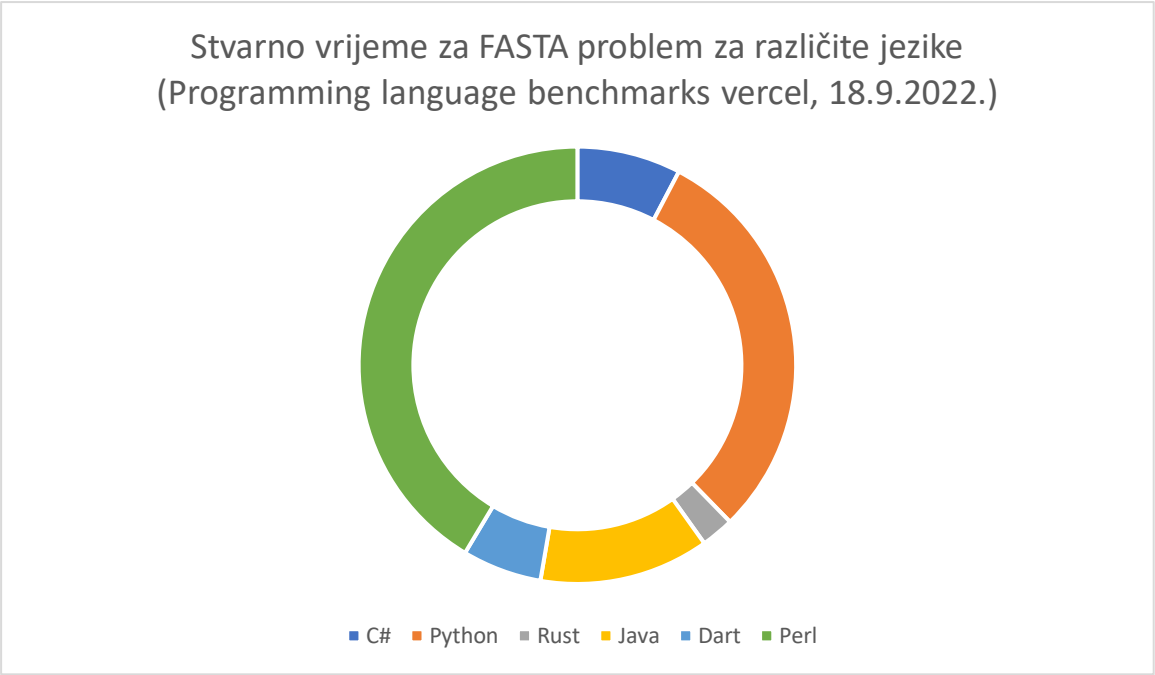
Slika 9: Peak-memory za Hello World problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



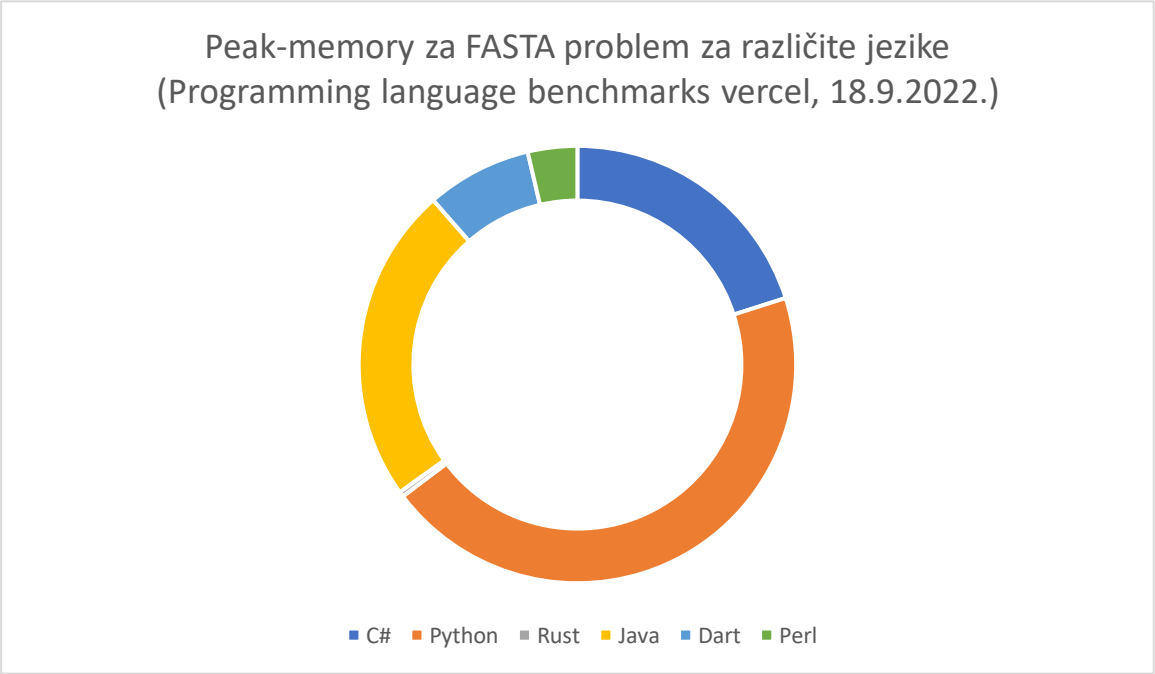
Slika 10: Peak-memory za Binary Tree problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



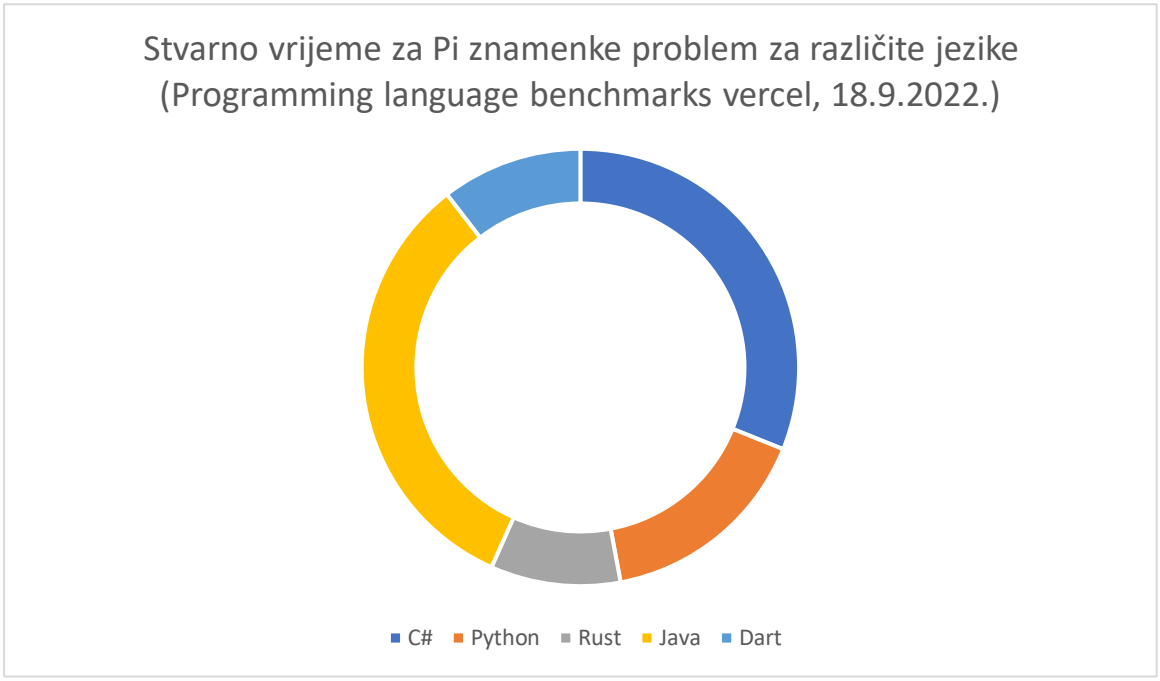
Slika 11: Peak-memory za Binary Tree problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



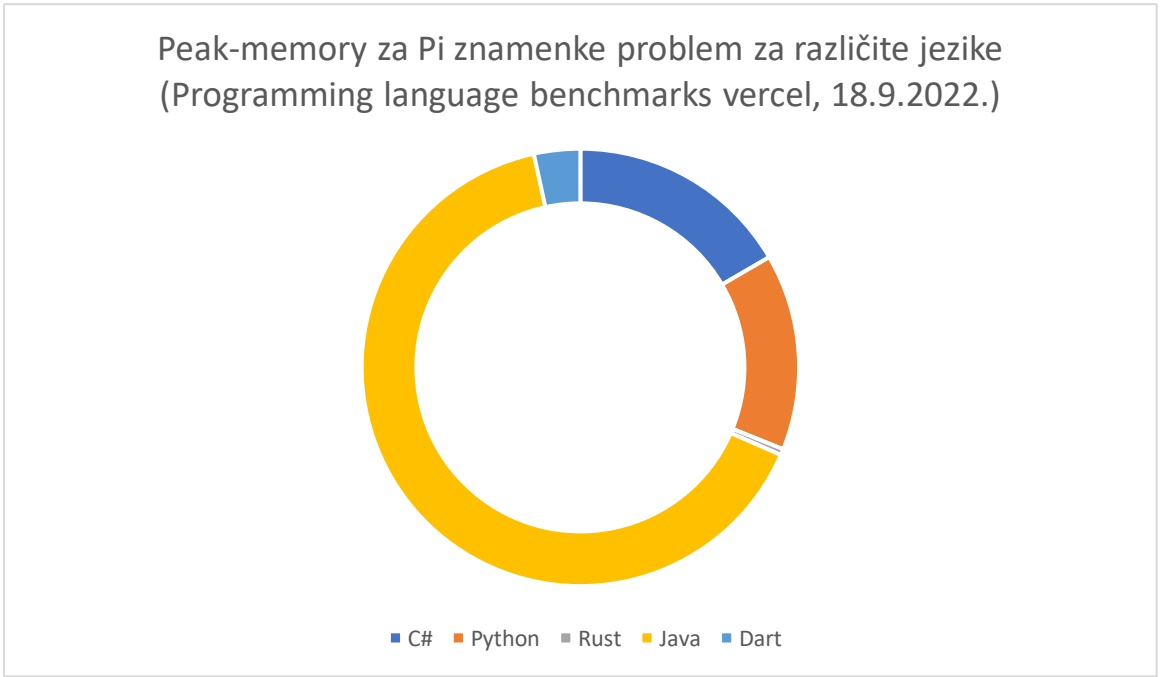
Slika 13: Stvarno vrijeme izvršavanja FASTA programa za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



Slika 14: Peak-memory za FASTA problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



Slika 16: Stvarno vrijeme izvršavanja ispisa znamenki broja Pi za različite jezike (Programming language benchmarks vercel, 18.9.2022.)



Slika 17: Peak-memory za Pi znamenke problem za različite jezike (Programming language benchmarks vercel, 18.9.2022.)

	Broj 1. mjesta	Broj 2. mjesta	Broj 3. mjesta	Broj 4. mjesta	Broj 5. mjesta	Broj 6. mjesta	Ukupni plasman
Python	2	1	1	/	1	/	PRVI
C#	/	1	1	3	/	/	ČETVRTI
Java	1	1	/	/	2	1	PETI
Perl	/	1	/	/	2	2	ŠESTI
Dart	/	1	3	1	/	/	TREĆI
Rust	3	/	/	1	/	1	DRUGI

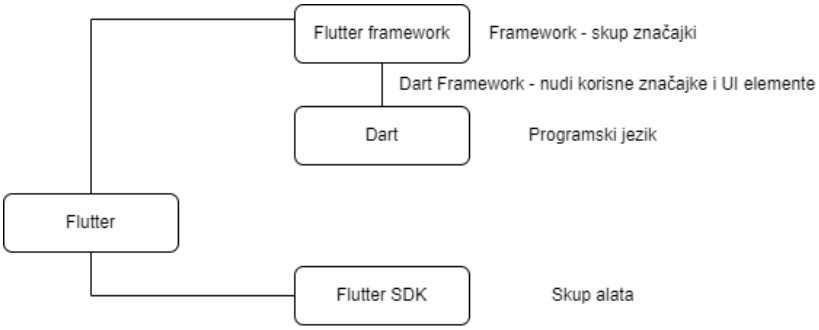
Tablica 7: Ukupni poredak tehnologija (vlastoručna izrada)

Pogledom na posljednju tablicu uviđamo i konačni poredak naših odabranih tehnologija u kojemu prvom mjesto odnosi Python, a zadnje Perl dok Dart stoji na trećemu. Teško je osporiti popularnost poput Pythonove ili one C#, no bližim uvidom u onu Darta i Fluttera (tablica 2 i tablica 3) možemo vidjeti naznake nastanka i vrlo brzog rasta popularnosti ove tehnologije. Jednako tako na odmet nisu ni radne performanse, navedeno stvarno vrijeme izvršavanja programa te korištenje radne memorije, pobliže sagledane na način djelovanja različitih tehnologija nad nekim od standardnih programskih problema, poput navedenog ispisa *Hello World*, broja decimalnih pozicija broja Pi (π), binarnog stabla itd. Ovdje Dart ponovno zauzima uvjerljivo ukupno 3. mjesto.

Uviđamo kako je Flutter sve više rastuća i traženija tehnologija za izradu više platformskih aplikacija te u svojem djelovanju primjenjuje također sve prisutniji Dart koji se svojim performansama uspijeva nositi s najvećim ustaljenim jezicima i tehnologijama. Uzevši statistiku u obzir koja prikazuje kako se u svakom aspektu našeg kratkog, ali pronicljivog istraživanja Flutter / Dart nalaze na „Zlatnoj sredini“ zaključujemo kako ga očekuje svjetla budućnost na tržištu koje je svojevrsno sve veća monopsija u vidu mnoštva tehnologija različitih proizvođača, a nas, krajnjih korisnika kao jedinstvenog kupca. Također je za kraj bitno napomenuti kako je odabir Fluttera dijelom i osoban upravo zbog činjenice kako je riječ o tehnologiji koja je novonastala (eng. *emerging*) te „plodove svog uspjeha tek treba ubrati“, a upravo je to jedan od preduvjeta ovoga rada, izabrati noviju, eng. *not yet mainstream* tehnologiju te je upravo zato Flutter ispravan odabir. ***Enter Flutter*!**

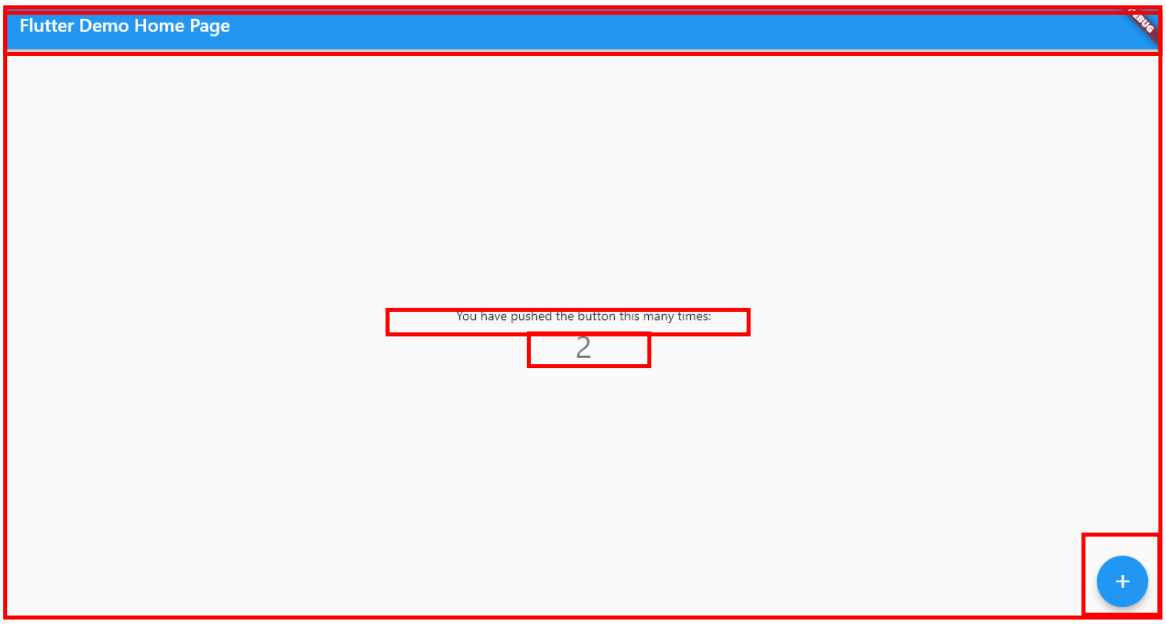
3. Flutter & Dart

Kako smo već napomenuli Flutter jest SDK (eng. *software development kit*) i *framework* / *widget* biblioteka za jedan programski jezik, Dart. Pojednostavljen odnos između dvoje uviđamo na sljedećem dijagramu:

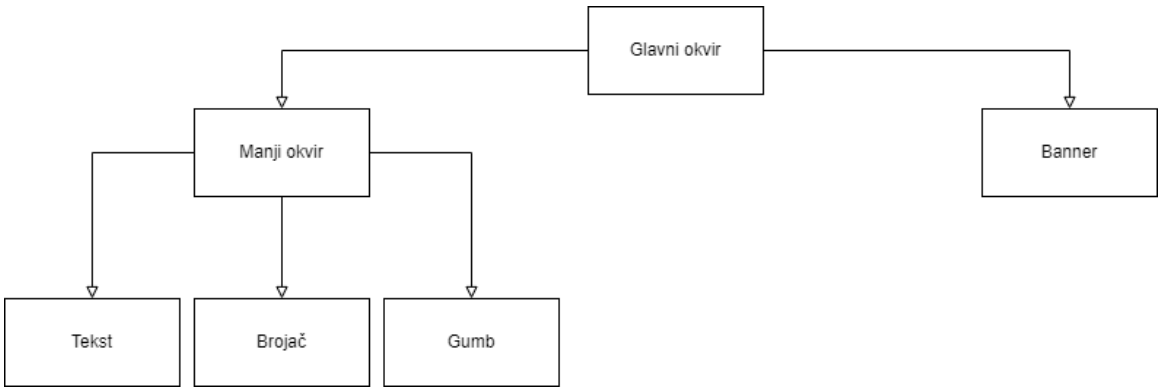


Slika 18: Pojednostavljeni prikaz Flutter & Dart (vlastoručna izrada, 2022.)

Još jedna vrlo bitna napomena prije samog postavljanja okoliša za razradu našeg praktičnog problema jest odnos *widgeta* u Flutteru. Za razliku od primjerice .NET gdje vizualna „povuci i ispusti“ (eng. *drag and drop*) akcija za *widgete* postoji, Flutter to rješava samim kodiranjem i to na način „sve je *widget*“ (eng. *everything's a widget*) odnosno stvaranjem widget drveta. Sve naznačeno crvenim okvirom jest *widget*.



Slika 19: Sve je widget (vlastoručna izrada, 2022.)



Slika 20: Widget drvo (vlastoručna izrada, 2022.)

Nadolazeća poglavlja rezervirana su za samu razradu rješenja na praktični, programski način gdje ćemo na primjeru / ima prezentirati načine na koje iste rješava Flutter. a za to je bitno postaviti okoliš, upoznati se s osnovama Darta, načinom na koji Flutter kompilira kod.

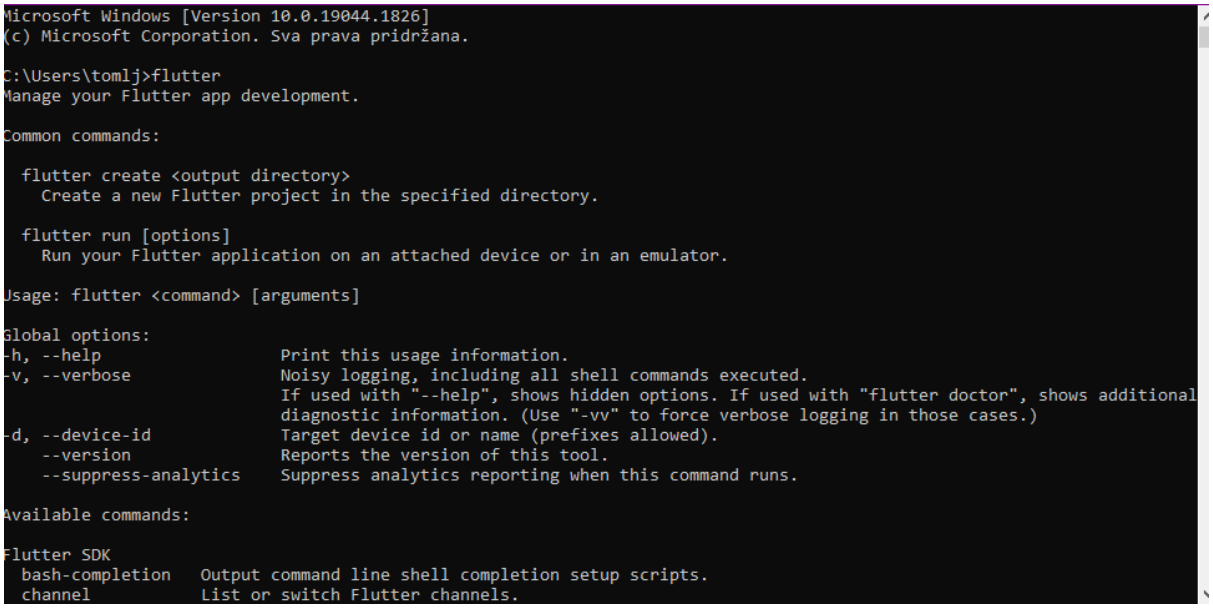
3.1. Postavljanje okoliša

Postavljanje okoliša najveći je, a gotovo i jedini preduvjet za uspješan rad na praktičnom rješenju problema nametnutog ovime radom stoga je iznimno bitno pomno pratiti korake, od kojih je ovdje navedeno njih 12, za što efektivniji učinak. Kako bismo osigurali neometan rad na aplikacijama i unutar okoliša važno je odabrati i podesiti ga. U nastavku ovoga poglavlja vidljivo je kako postavljamo tri različita: Android Studio, Visual Studio te Visual Studio Code ,a razlog tomu je izbjegavanje svih mogućih grešaka koje se mogu prikazati jednom kada naredbom Flutter doctor pregledamo „zdravlje“ okoliša. Također bitno je napomenuti kako nam Android Studio nudi mogućnost prikaza emulatora mobilnog uređaja, što nam pomaže u slučaju da se odlučimo naš proizvod prebaciti na mobilni uređaj. Izbor između ova tri okoliša na nama je samima, a u našem slučaju koristit ćemo Visual Studio Code zbog njegove „lakoće“ i brzine te velike podrške za Flutter u vidu različitih ekstenzija.

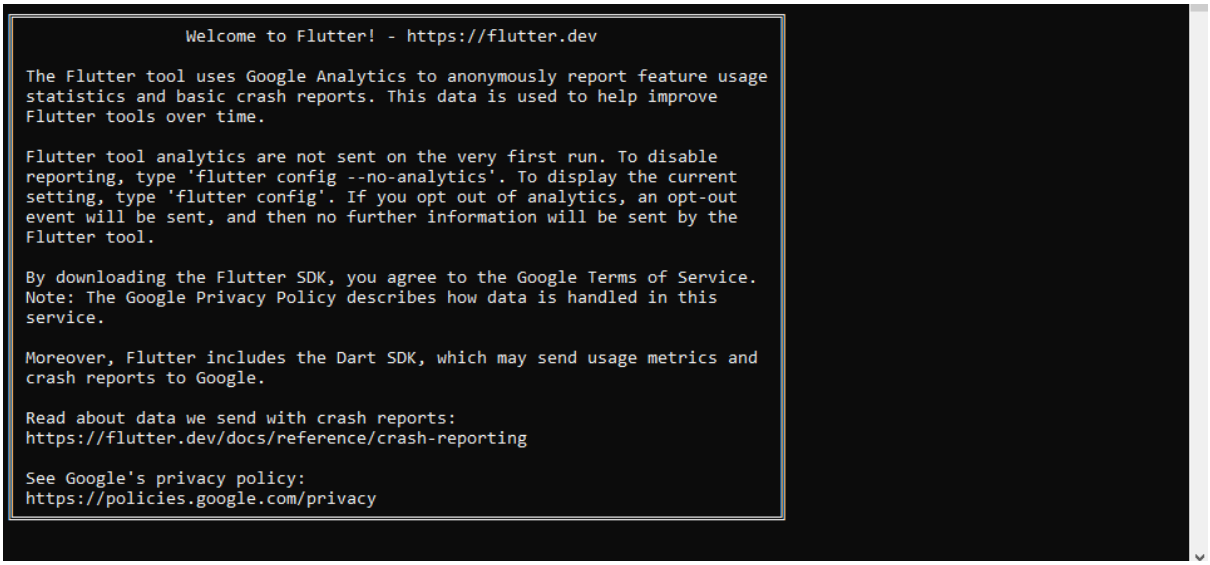
1. korak: Preuzeti Flutter SDK.

Posjetimo službene stranice Fluttera (www.flutter.dev) te navigiramo na povezinicu „*DOCS*“ u gornjem dijelu stranice. Na novootvorenoj stranici biramo *Get started* te nakon toga Windows i zatim poveznicu koja nosi naziv zadnje stabilne verzije nakon čega će automatski uslijediti preuzimanje. Preuzetu .zip mapu ekstrahiramo i pokrećemo Bitno je napomenuti

kako se SDK ne bi trebao ekstrahirati u mapama koje zahtijevaju višu razinu ovlasti (poput C:\program files).



Slika 22: preuzimanje Flutter SDK kroz CMD



Slika 21: uspostava Flutter SDK kroz CMD

2. korak: Preuzeti Git za Widnows

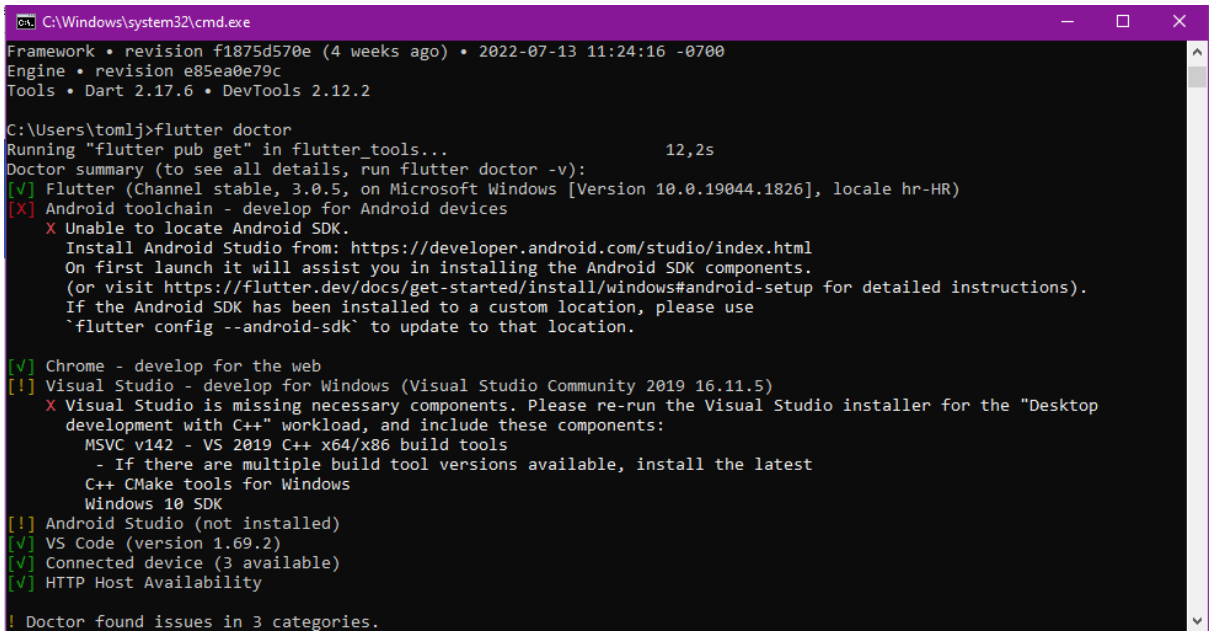
Posjetimo službene stranice Gita (www.git-scm.com) te navigiramo na poveznicu na početnoj stranici „*Download for Windows*“ te instaliramo u zadanoj mapi.

3. korak: Ažurirati putanju

Ažuriranje putanje ostvaruje se na način da preko tražilice sustava se pozicioniramo na „Svojstva sustava“, a zatim na „Varijable okruženja“. Ondje ćemo pod korisničkim varijablama označiti „*Path*“ te odabrati „Uređivanje“. Odabrati „Novo“ i kopirati putanju našeg prethodno preuzete i raspakirane Flutter mape (npr. C:\src\flutter\bin).

4. Korak: CMD Flutter doktorCmd

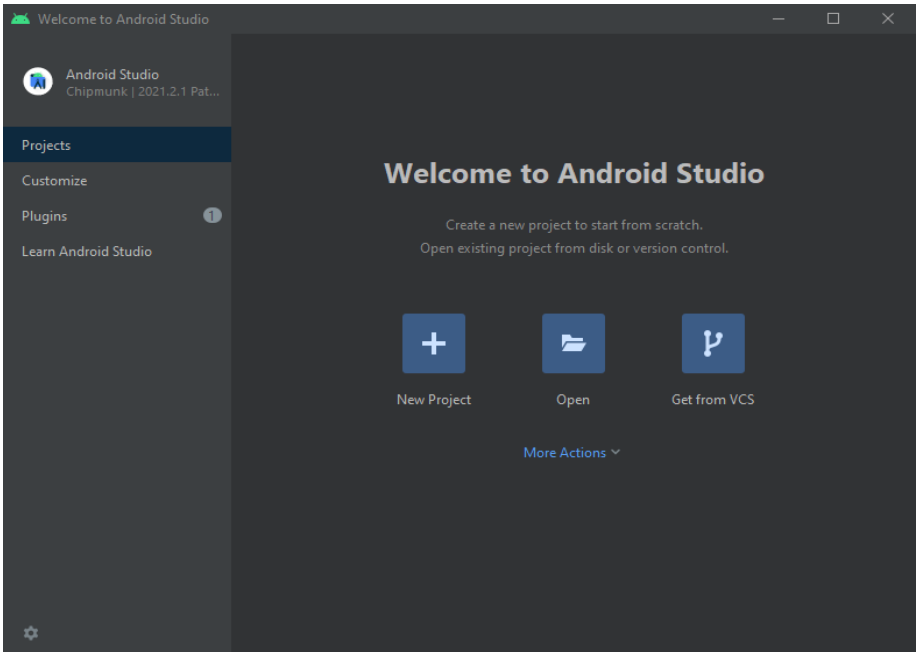
Pristupimo li CMD-u i upišemo „flutter“ dobit ćemo ispis korisnih (flutter) naredbi s prefiksom, a jedna od njih jest „flutter doctor“. Riječ je o naredbi za analizu koja nas povratno informira o stanju našeg sustava odnosno spremnosti na rad s Flutterom. Upisujući naredbu prvi puta dobit ćemo nekolicinu grešaka i upozorenja koje je naravno najbolje sve ukloniti kako bismo uživali u nesmetanom radu. Sljedeći koraci navode rješenja!



Slika 23: Flutter Doctor u CMD

5. korak: Preuzeti i instalirati Android studio

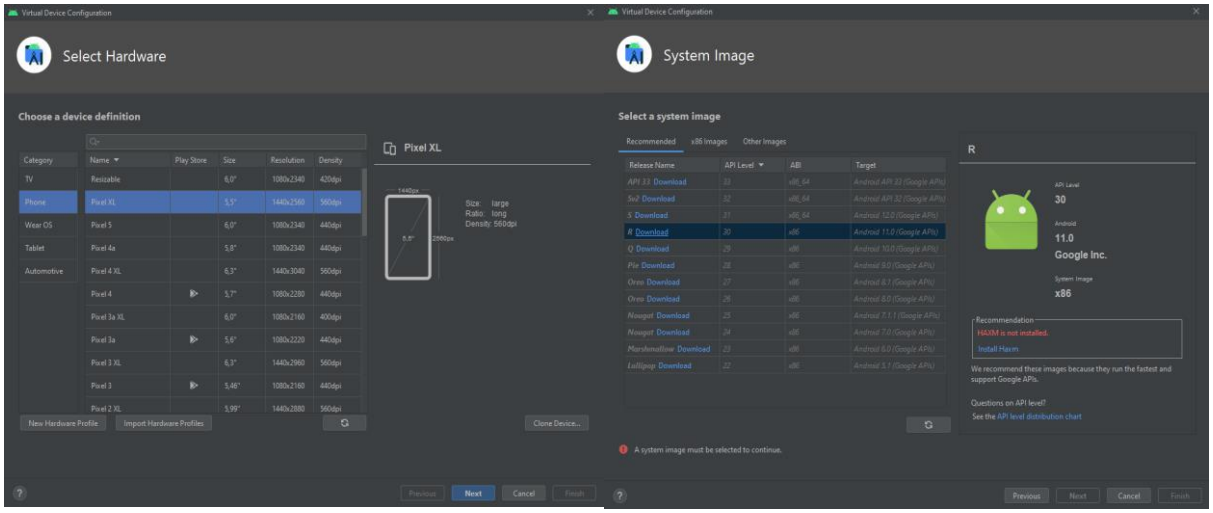
Posjetimo li stranicu (https://developer.android.com/studio) možemo i preuzeti istoimeni softver te instalirati. Tijekom instalacije vodit će nas čarobnjak te je bitno slijediti postavljene naredbe, ali napomenimo kako pod *SDK Components Setup* moramo odabrati (eng. tick) *Android SDK*, *API 30*, *Android Virtual Device* te ako smo u mogućnosti *Performance*. Jednom instaliranom studiju pristupamo i na početnom zaslonu odabiremo Configure zatim *SDK Manager*. Ondje je potrebno dodatno preuzeti *Android SDK command line tools*.



Slika 24: Android Studio

6. korak: Stvoriti virtualni mobilni uređaj (emulacija)

Ponovno se pozicioniramo na *Configure* unutar Android Studia, ali ovaj puta odabiremo *AVD manager* gdje ćemo, ako pratimo sve navedene korake s lakoćom izraditi svoj vlastiti Android pametni telefon kojeg onda pokrećemo i vidimo na svome ekranu.



Slika 25: Android Studio stvaranje emulatora

7. korak: Prihvatiti licence

Otvorimo li ponovno naš CMD i upišemo naredbu „flutter doctor“ vidjet ćemo sada kako još uvijek moramo prihvatiti „*Android & Google licenses*“, a to činimo na vrlo jednostavan način, upisom „flutter doctor –android-licenses“ te na svaku ponuđenu licencu odgovaramo s y (označava eng. *yes*).

8. korak: Instalirati / ažurirati Visual Studio

Ukoliko ne posjedujemo Visual Studio isti moramo preuzeti i instalirati (https://visualstudio.microsoft.com/downloads/). Jednom u našem posjedovanju najbitnije je kroz njega preuzeti Desktop development with C++ ukoliko se nije preuzeo zajedno s ostalim komponentama.

9. korak: Preuzeti i Instalirati Visual Studio Code

Isto možemo napraviti putem (https://code.visualstudio.com/).

10. korak: Kreirati projekt

U CMD-u se pozicionirati na željenu mapu (gdje želimo stvoriti svoj projekt(e)) te upisati flutter create **ime_projekta** (od posebnih znakova dopuštena je samo donja povlaka „_“).

11. korak: Otvoriti projekt

Otvaramo Visual Studio Code te putem *File* i *Open Folder* izabrati cijeli folder koji smo kreirali u prethodnome koraku.

12. korak: Instalirati ekstenzije

Pozicionirani smo unutar Visual Studio Codea te pritiskom na *View*, a zatim *Extensions* dobivamo uvid u sve ekstenzije te pomoću pretrage odabiremo „Flutter“ ekstenziju, u koju je Dart ekstenzija automatski uključena. Također preporučujem / koristim ekstenzije: H*ighlight matching tag*, *Prettier – code formatter*, *Indent-rainbow*, *Material icon theme*, *Git base* i *Git*.

3.2. Dart, osnovna načela

Kao posljednje teorijski informativno poglavlje u dijelu praktične razrade problema završnog rada nameće nam se samo upoznavanje s osnovama Dart programskog jezika kao što je deklariranje varijabli, komentari, korištenje aritmetičkih operatora klasa i objekata.

Komentari : u Dartu uspostavljaju se pisanjem dviju kosih crta unaprijed (***//***)

Varijable : kao i većini programskih jezika služe za „pospremanje“ vrijednosti na memorijskoj lokaciji. Zadana vrijednost varijabli kao i uvijek jest *null*.

```
var nekoIme = "Pozdrav!";
//ili
String nekoDrugoIme;
nekoDrugoIme = „Pozdrav 2!";
```

Kod 1: deklariranje varijabli u Dartu

Vidljiva razlika između dva pristupa deklariranju tipa varijable je u samom instanciranju. Naime dobra Dart praksa nalaže da se varijabla označava s **var**, kao svojim tipom podataka, ako se istoj odmah dodjeljuje vrijednost. Riječ je o svojstvu koje se naziva **zaključivanje tipa** (eng. *type inference*) a ono Dartu omogućuje da uz pomoć desne strane izjave zaključi o kojem je tipu podataka riječ, iako je s lijeve strane iste izjave navedeno **var**. Primijetimo još jednu zanimljivost, a to je da se u Dartu tekstna varijabla označava tipom podataka **String**, pisano velikim početnim slovom.

Operatori :

Operator	Naziv	Primjer	x = 6, y = 4
+	Zbrajanje	x+y = 10	
-	Oduzimanje	x-y = 2	
*	Množenje	x * y = 24	
/	Dijeljenje	x / y = 1,5	

%	Modul	x % y = 2
~/	Dijeljenje (int)	x ~/ y = 1
++	Inkrement	++x = 7
-	Dekrement	-- y = 3

Tablica 8: Primjer operatora u Dartu (vlastoručna izrada, 2022.)

Objektno orijentirani program :

Osvrnimo se i na klase kao jednu od sastavnih komponenata Darta:

```
//Kreiranje klase u Dartu (vrlo je slično drugim popularnim programskim jezicima)
class imeKlase{
	tipPodataka imeVarijable = vrijednost;
}

//Kreiranje objekta klase (na četiri načina gdje je zadnji ujedno i najčešći / najprimjereniji)
var imeVarijable = imeKlase();
//ili
var imeVarijable = new imeKlase();
//ili
imeKlase imeVarijable = imeKlase();
//ili
imeKlase imeVarijable = new imeKlase();

//Kreiranje konstruktora klase
class imeKlase{
	imeKlase(){

	}
}
```

Kod 2: Klasa i objekt unutar Darta

Dart ne zahtjeva da se pri pozivanju klase koristi ključna riječ ***new***.

Snažno tipkanje (eng. *strong typing*) :

Dart je Statički tipiziran, odnosno tipovi varijabli poznati su u vrijeme kompajliranja i provodi se provjera, međutim Dart podržava i nešto što nazivamo zaključivanje tipa što nam omogućuje izostavljanje eksplicitne deklaracije tipa kada se isti može zaključiti iz konteksta.

```
void main() {
  String poruka = 'Bok, Dart!';
  int broj = 42;

  // Type error:
  // message = 42;

  print(poruka);
  print(broj);
}
```

Kod 3: Snažno tipkanje unutar Darta

Asinkrono programiranje:

Dart ima ugrađenu podršku za asinkrono programiranje što uvelike olakšava rukovanje zadacima poput korisničkih interakcija, I/O operacija. Ovdje su ključne riječi poput **async** i **await**.

```
import 'dart:async';

void fetchData() {
  print('Fetching data...');

  Future.delayed(Duration(seconds: 2), () {
    print('Data fetched!');
  });
}
```

```
void main() {
  print('Start');
  fetchData();
  print('End');
```

Kod 4: Asinkrono programiranje unutar Darta (vlastoručna izrada, 2023.)

```
}
```

Biblioteke i paketi:

Biblioteke kao zbirke i paketi kao jedinice distribucije koje sadrže jednu ili više biblioteka omogućuju nam uvoz mnogih funkcionalnosti i modula koje pospješuju i olakšavaju rukovanje kako Dartom tako i Flutterom.

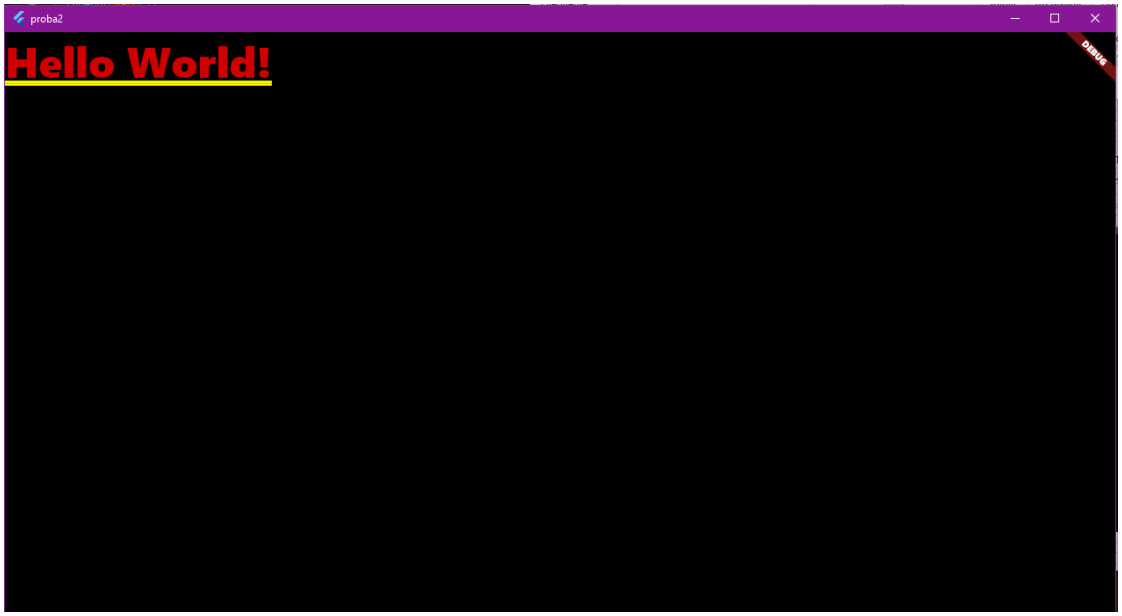
```
import 'dart:math';


void main() {
  var randomNumber = Random().nextInt(10);
  print(randomNumber);
}
```

Kod 5: Uvoz biblioteke unutar Darta (vlastoručna izrada, 2023.)

Naravno da ovo nisu sve mogućnosti niti načela Darta kao programskog jezika niti Fluttera kao njegove tehnologije, a više o istima otkrit ćemo u sljedećim poglavljima gdje ćemo se pozabaviti izradom praktičnih primjera.

3.3. *Hello World*

Kraće poglavlje koje će nam uvelike pomoći u razumijevanju osnovnih principa stvaranja najjednostavnije aplikacije, ujedno i apeksa kada je riječ o upoznavanju s novim programskim jezikom ili tehnologijom, a to jest jednostavni ispis riječi *Hello World* koji izgleda ovako:



Slika 26: Hello World (vlastoručna izrada, 2023.)

Vidljivo je kako aplikacija ne privlači svojim izgledom, no to je nešto što ćemo u kasnijim poglavljima uvelike razraditi. Pred nama se konačno nalazi vlastoručni kod koji preostaje razlučiti u svrhu poimanja osnovnih principa korištenih u stvaranju ove aplikacije.

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(home: Text('Hello World!'));
  }
}
```

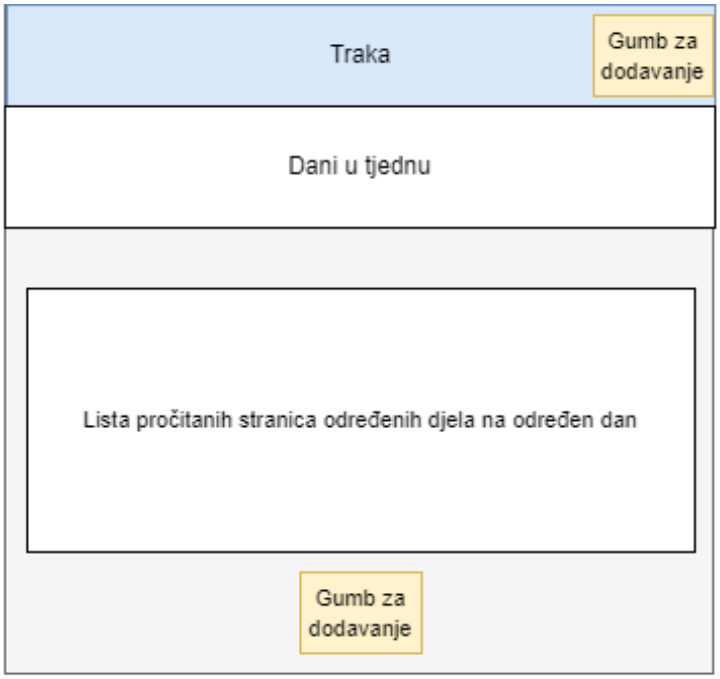
Kod 6: Hello World (vlastoručna izrada, 2023.)

Jednom pozicionirani u novostvoreni Flutter projekt u našem postavljenom VS Codeu, kojeg smo uspostavili s ostatkom potrebnog okoliša (poglavlje 3.1. Postavljanje okoliša) otvaramo **main.dart** editor unutar mape **lib**. Ova stranica trenutno je ispunjena programskim kodom koji je rezultirao prvom početnom aplikacijom (Slika 17: Sve je Widget) što smo naravno izbrisali i zamijenili našim prethodno navedenim kodom.

Načelo „Sve je *widget*“ govori nam kako bismo za uspostavu početnog ekrana morali stvoriti *Widget*. Upravo to činimo stvaranjem nove klase proizvoljnog imena, u ovom slučaju *MyApp* (Kod 4: Ispis Hello World, linija 7) koja će uz pomoć nasljeđivanja (eng. *inheritance*) poprimiti sva svojstva temeljne klase koja se naziva *StatelessWidget*, a to činimo ključnom riječju *extends*. Ova temeljna klasa nalazi se unutar mape **material.dart** kojoj pristupamo uvođenjem (eng. *import*) (Kod 4: Ispis Hello World, linija 1). Uvođenje ne bi bilo moguće bez postojanja zavisnosti (eng. *dependency*) prema Flutteru, a postojeće možemo vidjeti i uređivati u **pubspec.yaml**. Potreba za ovim postupkom jest uvođenje već postojećih Flutter funkcionalnosti te samim time olakšan i dosljedan rad. Vidljivo je kako se u našoj klasi nalazi još ponešto, a riječ je o metodi naziva sagradi (eng. *build*). Ova metoda želi sagraditi nešto, rečeno programerskom terminologijom ona želi vratiti nešto, a riječ o Widgetu, stoga je isto navedeno kao njezin tip podataka. Kako bi gradnja bila moguća, za istu su nam potrebni materijali, u ovom slučaju to su argumenti tipa *BuidContext* dosljednog imena *context* (Kod 4: Ispis Hello World, linija 9). Naveli smo kako metoda želi odnosno mora vratiti nešto, zamislimo da je to rezultat gradnje, a rezultat jest *widget* zvan *MaterialApp*. Jednako kao i *BuildContext*, *MaterialApp* jest dio **material.dart** mape, no primjećujemo kako je ona ujedno i klasa prema tome što sadrži (imenovani) argument *home*. Napomenimo kako je on samo jedan od pregršt argumenata kojima možemo pristupiti preko ove klase znamo li ime argumenta. Argumentu ne dodajemo statičnu vrijednost već novi *widget* naziva *Text* (Kod 4: Ispis Hello World, linija 10). Uviđamo kako je naravno i ovaj *widget* zasebna klasa koja sadrži riječi ' *Hello World* '. Kako bismo sve pokrenuli našu novoizrađenu klasu moramo pozvati u *main*, a to i činimo s ključnom riječju *runApp* (Kod 4: Ispis Hello World, linija 4). Za objasniti je ostala jedino ključna riječ *@override*, a njezina funkcija jest nadjačavati *build* metodu *StatelessWidgeta* koja već postoji s onom koju mi uvodimo. Iako će se to uvijek dogoditi, dobra je praksa koristiti ovu ključnu riječ.

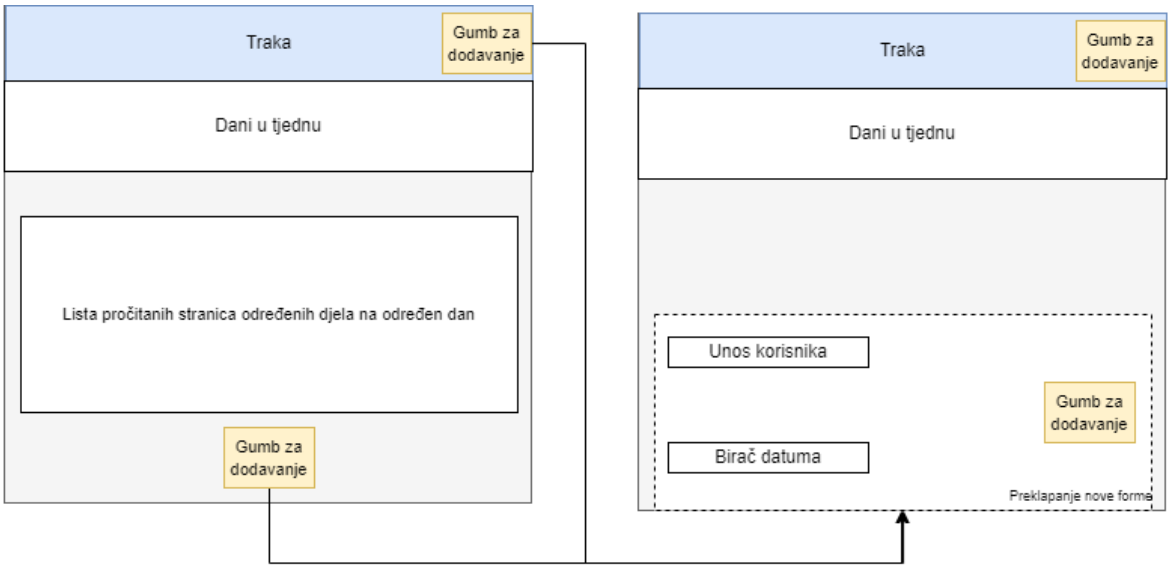
3.4. Razrada jednostavnog praktičnog primjera

Kao i svaku aplikaciju, započnimo idejnom razradom rješenja na način uvođenja modela. Model samog zaslona poslužit će nam za jasniju predodžbu željenog rasporeda widgeta i uređenja prikaza, bolje poznato kao UI (eng. *user interface*) dok će nam dijagram akcije poslužiti za jasnije poimanje interakcije nas kao korisnika (eng. *end user*) i same aplikacije (njezinih mogućnosti). Početna ideja sastoji se od banera i podignutog gumba u gornjem dijelu ekrana, nakon čega slijedi grafikon koji prikazuje napredak u danima u tjednu. Naposljetku nalazimo prikaz unesenih zapisa te još jedan podignuti gumb za dodavanje novog zapisa.



Slika 27: Prikaz widgeta prve aplikacije (vlastoručna izrada, 2023.)

Jednom stisnuti gumb potaknut će novi ekran koji će preklopiti dio ekrana aplikacije s mogućnošću unošenja novog zapisa na način upisa teksta (npr. naslov djela i autor), vremena provedenog čitanja (npr. u minutama) te naravno odabir dana u tjednu kada se akcija dogodila.



Slika 28: Nova forma (vlastoručna izrada, 2023.)

Prije negoli započnemo sa samom daljnjom razradom modela, pojmova i koda napomenimo kako se još uvijek nismo susreli s većinom mogućnosti i widgeta koje nam nudi Flutter jednako kao i pravilima Dart programskog jezika. Zbog toga ćemo posebnu pažnju posvetiti dijelovima rješenja koja su za nas nova, odnosno nisu spomenuta i / ili obrađena u prethodnim poglavljima (poput poglavlja 3.3. *Hello World*).

Stvaranje aplikacije započinjemo na poznat način, otvaranjem cmd naredbenog retka i pozicioniranjem na željeni prostor (mapu) za naš nadolazeći projekt, nakon čega upisujemo `flutter create` *`proizvoljno_ime_projekta`*. Radnju slijedimo pokretanjem našeg VS Code programa te odabirom File -> Open folder -> proizvoljno_ime_projekta otvaramo cjelokupnu mapu u kojoj će biti sadržan naš projekt. Razlog tomu jest naravno taj što uz sam programski kod koji ćemo početi pisati u **main.dart** ovdje se učitavaju i sve naše zavisnosti (eng. *dependencies*) u spomenutom **pubspec.yaml**, biblioteke za Android, iOs, Linux, Web i naravno Windows kao i potrebnu metadata. Poput prethodne, i ovu aplikaciju započinjemo stvaranjem nove datoteke u mapi lib koju ćemo naravno nazvati main.dart, ondje će se nalaziti početni i najveći dio našeg rješenja.

Započnimo uvođenjem biblioteke materijala potrebne za sav daljnji rad u kojoj su sadržani naši najvažniji objekti, odnosno widgeti, i njihove funkcije, odnosno metode te stvaranjem prazne main funkcije za pokretanje našeg rješenja.

Nastavljamo stvaranjem nove klase tipa ***MaterialApp*** proizvoljnog imena koja je središnjica aplikacije jer će Flutteru dati do znanja da želimo koristiti sve njezine komponente (eng. *material components*) i slijediti upravo takvu temu, a ne primjerice ***Cupertino***. Nadalje, susrećemo se s novim widgetom naziva ***Scaffold***, a riječ je o widgetu koji uvijek dolazi poslije *MaterialApp* te nam kao svojevrsni kontejner vršne razine, zapravo omogućuje uporabu svih ostalih „vidljivih" widgeta poput gumbova (eng. *elevated button*), trake (eng. *app bar*), tekstova. Ponovimo kako ovaj widget, kao i svaki drugi, zapravo jest dio konteksta koji nam služi za gradnju koju nam omogućuje metoda eng. ***build*** tipa ***widget***, a parametar koji ta metoda prima jest ***context*** tipa ***BuildContext***. *BuildContext* različit je za svaki od widgeta u widget drvcu upravo iz razloga što je svaki od widgeta različit. Widget možemo zamisliti kao svojevrsni recept (eng. *blueprint*) za nešto što se tek treba prikazati na samome ekranu, no on sam po sebi ne zna gdje bi se morao stvoriti, stoga mu u pomoć uskače navedeni argument koji će tu poziciju pronaći prolazeći kroz spomenuto drvce widgeta (eng. *widget tree*) i uvidjeti relacije prema ostalim widgetima kako bi u poziciji prema njima izgradio željeni.

Uvodimo novi folder naziva **model** unutar kojeg postavljamo djelo.dart, nazvano prema klasi ondje smještenoj. Riječ je dakle o običnoj klasi ali ne i o widgetu, a ista će primiti parametre koji za aplikaciju o vođenju navika čitanja imaju smisla, kao što su naslov i autor djela, te datum i vrijeme utrošeno na čitanje. Ovdje moramo stvoriti i konstruktor kojem ćemo proslijediti imenovane argumente. Ključna riječ eng. *required* sadržana u biblioteci *foundaion.dart* označava da se unos ovih argumenata ne smije propustiti.

```
import 'package:flutter/foundation.dart';

class Djelo {
  final String id;
  final String autor;
  final String naslov;
  final double vrijemeCitanja;
  final DateTime danCitanja;

  Djelo(
      {required this.id,
      required this.autor,
      required this.naslov,
      required this.vrijemeCitanja,
      required this.danCitanja});
}
```

Kod 7: Pomoćna klasa djela (vlastoručna izrada, 2023.)

Stvorimo prvi widget izdvojen u vlastitoj datoteci koji će inicijalizirati prethodno stvorenu klasu u obliku liste. Naizgled ova izjava nema smisla, hoće li biti riječ o widgetu ili listi? Odgovor upravo leži u tome da ćemo uspostaviti listu widgeta (npr. kartice) s informacijama i podacima

koje želimo proslijediti, a proslijediti ćemo upravo one koji se nalaze u stvorenoj klasi. Naravno kako bismo mogli koristiti objekte nekog druge datoteke moramo ju uvesti sada već poznatom naredbom *import*. Kako bismo ostvarili navedeno mapiranje, koristimo metodu imena *map* i mapiramo našu listu naziva **unosi** na widgete. Metoda za argument prima funkciju koju smo nazvali djelo te mora nešto vratiti, naravno riječ je o widgetu, u ovom slučaju to jest kartica. Metoda će iterativno izvesti funkciju nad svakim objektom na kojeg. Ostatak koda daje nam do znanja kako je pretežito riječ o uređivanju prikaza samih widgeta gdje bih izdvojio funkciju ***ListView***, koja widget automatski čini pomični (eng. *scrollable*). Također postaje jasnije da jednom kada želimo pristupiti jednom od argumenata klase **Djelo** sve što nam je činiti jest imenovati funkciju na koju je mapirana lista klase (u ovom slučaju to je **djelo**) i stavljanjem točke dobivamo popis mogućih argumenata.

```
class ListaDjela extends StatelessWidget {
  @override
  final List<Djelo> unosi;
  final Function obrisiUnos;
  ListaDjela(this.unosi, this.obrisiUnos);
  Widget build(BuildContext context) {
    return Container(
      height: 500,
      child: unosi.isEmpty
          ? Column(
              children: <Widget>[
                Text('Danas još ništa nije pročitano!'),
                Flexible(
                  child: Image.asset(
                    'assets/images/waiting.png',
                    //fit: BoxFit.cover,
                  ),
                ),
              ],
            )
          : ListView.builder(
              itemBuilder: (ctx, index) {
                return Flexible(
                  fit: FlexFit.tight,
                  child: Card(
                    child: Wrap(
                      children: <Widget>[
                        Container(
                          margin: EdgeInsets.symmetric(
                              horizontal: 50, vertical: 20),
                          padding: EdgeInsets.all(15),
                          width: 200,
                          //width: double.infinity,
                          decoration: BoxDecoration(
                            border: Border.all(color: Colors.indigo,
width: 2),
                          ),
                          child: Text(
```

```
                        'Naslov: ' '\n${unosi[index].naslov}',
                        style: TextStyle(
                            fontWeight: FontWeight.bold,
                            //color: Colors.purple,
                            fontSize: 18),
                      ),
                    ),

                    //mainAxisAlignment: MainAxisAlignment.end,
                    //children: <Widget>[
                    Container(
                      margin: EdgeInsets.symmetric(
                          horizontal: 100, vertical: 20),
                      padding: EdgeInsets.all(15),
                      width: 200,
                      //width: double.infinity,
                      decoration: BoxDecoration(
                        border: Border.all(color: Colors.indigo,
width: 2),
                      ),
                      child: Text('Autor:\n' '${unosi[index].au-
tor}',
                          style:
                              TextStyle(fontSize: 16, color: Co-
lors.black)),
                    ),
                    Container(
                      margin: EdgeInsets.symmetric(
                          horizontal: 100, vertical: 20),
                      padding: EdgeInsets.all(15),
                      width: 200,
                      //width: double.infinity,
                      decoration: BoxDecoration(
                        border: Border.all(color: Colors.indigo,
width: 2),
                      ),
                      child: Text(
                        'Datum:\n'
                        '${DateFormat('dd.MM.yyy.    HH:mm').for-
mat(unosi[index].danCitanja)}',
                        style:   TextStyle(color:   Colors.black,
fontSize: 16),
                      ),
                    ),
                    Container(
                      margin: EdgeInsets.symmetric(
                          horizontal: 100, vertical: 20),
                      padding: EdgeInsets.all(15),
                      width: 200,
                      //width: double.infinity,
                      decoration: BoxDecoration(
                        border: Border.all(color: Colors.indigo,
width: 2),
                      ),
                      child: Text(
                        'Vrijeme:\n'
```

```
                                '${unosi[index].vrijemeCitanja.toStrin-
gAsFixed(2)}'
                                ' min',
                                style: TextStyle(color: Colors.black,
fontSize: 16),
                              ),
                            ),
                            Container(
                              child: IconButton(
                                icon: Icon(Icons.delete),
                                onPressed: () => obrisiUnos(unosi[in-
dex].id),
                                color: Theme.of(context).errorColor,
                              ),
                            ),
                          ],

                          //],
                        ),
                      ),
                    );
                  },
                  //itemCount: unosi.length),
              ),
    );
  }
}
```

Kod 8: Lista kartica s djelima

Naravno kako bismo išta mogli prikazati moramo nešto i unesti stoga slijedi potpuno novi dio u našem kodu a tiče se korisničkog unosa (eng. *user input*). Kako bismo ovo ostvarili stvorio sam još jednu datoteku pod mapom *widget* naziva **novi_unos**. Započinjemo s uvođenjem tri polja za upis teksta, widget pod nazivom ***TextField*** koji će se naravno nalaziti u *Column* pošto znamo da je to jedan od widgeta koji može sadržavati *children*, a ne *child*. Taj column obgrljen jest kontejnerom zbog lakše manipulacije i uređivanja. Još jedna novina jest dodavanje takozvanih kontrolera. Kontroleri „osluškuju“ (eng. *listener mechanism*), u ovom slučaju naš odnosno korisnikov unos u tekstualna polja, a instanciraju se na jednostavan način dodjeljivanja nekoj proizvoljnoj varijabli bez tipa podataka i pozivanja imenovanim argumentom ***controller*** unutar *TextField* klase. Osim ovoga, veoma je bitno napomenuti i postojanje gumba koji će nam poslužiti za upis stavki jednom kada je isti pritisnut. Jedan od njegovih imenovanih argumenata jest ***onPressed*** koji kao što ime govori želi napraviti nešto jednom kada je pritisnut i kao argument očekuje funkciju stoga mu mi prosljeđujemo onu prethodno definirano **dodajDjelo** i unutar zagrada definiramo što očekujemo da jest upisano. Ostatak koda također kao i u prethodnom odjeljku uvelike služi za uspostavljanje stila, oblika i rasporeda na samome zaslonu.

```
class NoviUnos extends StatefulWidget {
  final Function dodajDjelo;
  NoviUnos(this.dodajDjelo);

  @override
  State<NoviUnos> createState() => _NoviUnosState();
}

class _NoviUnosState extends State<NoviUnos> {
  @override
  final naslovController = TextEditingController();
  final autorController = TextEditingController();
  final vrijemeCitanjaController = TextEditingController();
  DateTime? _izabraniDatum;
  void _unesiPodatke() {
    final uneseniNaslov = naslovController.text;
    final uneseniAutor = autorController.text;
    final unesenoVrijeme = double.parse(vrijemeCitanjaControl-
ler.text);
    if (uneseniNaslov.isEmpty ||
        uneseniAutor.isEmpty ||
        unesenoVrijeme <= 0 ||
        _izabraniDatum == null) {
      return;
    } else {
      widget.dodajDjelo(
          uneseniNaslov, uneseniAutor, unesenoVrijeme, _izabraniDa-
tum);
    }
    Navigator.of(context).pop();
  }

  void _pokaziKalendar() {
    showDatePicker(
            context: context,
            initialDate: DateTime.now(),
            firstDate: DateTime(2023),
            lastDate: DateTime.now())
        .then((zeljeniDatum) {
      if (zeljeniDatum == null) {
        return;
      }
      setState(() {
        _izabraniDatum = zeljeniDatum;
      });
    });
  }

  Widget build(BuildContext context) {
    return Card(
      elevation: 6,
      child: Container(
        margin: EdgeInsets.all(10),
        //padding: EdgeInsets.all(6),
        child: Column(
```

```
            crossAxisAlignment: CrossAxisAlignment.end,
            children: <Widget>[
              TextField(
                style: TextStyle(fontSize: 16),
                decoration: InputDecoration(labelText: 'Naslov'),
                controller: naslovController,
              ),
              TextField(
                style: TextStyle(fontSize: 16),
                decoration: InputDecoration(labelText: 'Autor'),
                controller: autorController,
              ),
              TextField(
                style: TextStyle(fontSize: 16),
                decoration: InputDecoration(labelText: 'Vrijeme ci-
tanja (min)'),
                controller: vrijemeCitanjaController,
                keyboardType: TextInputType.number,
              ),
              Container(
                height: 80,
                child: Row(
                  children: <Widget>[
                    Padding(
                      padding: const EdgeInsets.all(10.0),
                      child: Text(
                        _izabraniDatum == null
                            ? 'Nije izabran datum'
                            : DateFormat('dd.MM.yyy').format(_izab-
raniDatum!),
                        style: TextStyle(fontSize: 14),
                      ),
                    ),
                    Padding(
                      padding: const EdgeInsets.all(20.0),
                      child: TextButton(
                        child: Text(
                          'Izaberite datum',
                          style: TextStyle(
                              color: Theme.of(context).accentColor,
                              fontWeight: FontWeight.bold,
                              fontSize: 15),
                        ),
                        onPressed: _pokaziKalendar,
                      ),
                    ),
                  ],
                ),
              ),
              Container(
                decoration: BoxDecoration(
                  boxShadow: [
                    BoxShadow(
                      //color: Colors.grey.withOpacity(0.5),
                      spreadRadius: 1,
                      blurRadius: 3,
```

```
                        offset: Offset(0, 2), // changes position of
shadow
                      ),
                    ],
                  ),
                  child: ElevatedButton(
                    onPressed: () => _unesiPodatke(),
                    child: Text(
                      'Dodaj',
                      style: TextStyle(fontSize: 16),
                    ),
                    style: ButtonStyle(
                        foregroundColor: MaterialStateProperty.all(Co-
lors.white),
                        backgroundColor:
                            MaterialStateProperty.all(Colors.purple)),
                  ),
                )
              ]),
      ),
    );
  }
}
```

Kod 9: Novi unos korisnika (vlastoručna izrada, 2023.)

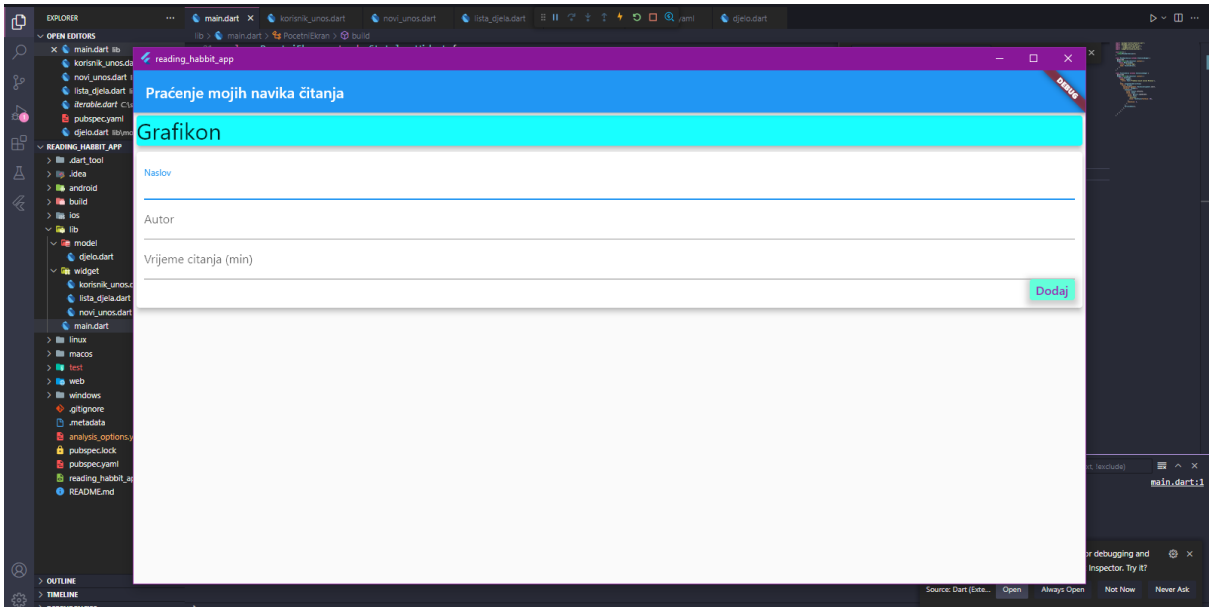
Gumb jest upravo ono što nas dovodi do zadnjeg i najkompleksnijeg, iako linijama koda najkraćeg, widgeta. Nalazi se u svojoj datoteci kao i ostali te nosi ime **korisnik_unos**. Primijetimo kako do sada nismo imali ispisa na ekranu, a razlog tomu je naravno što ni jedan od ovih widgeta, objekata, funkcija nismo pozivali unutar same ***main.dart*** datoteke. Potreba za ovim widgetom dolazi iz razloga što će on objediniti oba prethodno razrađena *StatelessWidgeta* te kao jedinstveni *StatefulWidget* biti pozvan u glavnom programu gdje ponovno želimo zadržati samo *StatelessWidget*. Ovime ne samo da se dobiva na smanjenom obujmu glavne datoteke već su widgeti kao cjeline podijeljeni te rad s njima olakšan. Započinjemo stvaranjem prazne liste koja će biti popunjavana tako kako ćemo mi potvrđivati unos upisom zadanih parametara koji su vidljivi u ***void*** funkciji **_dodajNovoDjelo** a podudaraju se s prethodno odabranim vrijednostima pozivanim u funkciji gumba (naslov, autor, vrijeme čitanja). Ovdje smo također dodali vrijednosti (argumente) koje prosljeđujemo onima klase **Djelo**, vidljivo u **naslov: naslovDjela** koji su oboje imenovani argumenti. Kako je riječ o *StatefulWidgetu* za kojeg sada znamo da želi unesti promjenu na zaslonu koja će odmah biti vidljiva (odnosno želi pozvati metodu build svaki put kada se učini promjena) moramo biti precizni u navođenju same promjene i onoga što se mijenja. Ovo je naravno izraženo metodom posebnom za ovaj widget ***setState*** koja za argument prima funkciju. U našemu primjeru riječ je o tome da svojoj listi, pritiskom gumba, želimo dodati novi zapis i moći ga vidjet, pa to i činimo dodavanjem linijom **_korisnikDjela.add(novoDjelo)**. Pomoću

spomenute metode za izgradnju vraćamo dva konstruktora **NoviUnos** čiji argument jest pokazivač na funkciju sadržanu unutar iste datoteke i **ListaDjela** čiji argument jest pokazivač na funkciju u drugoj datoteci.

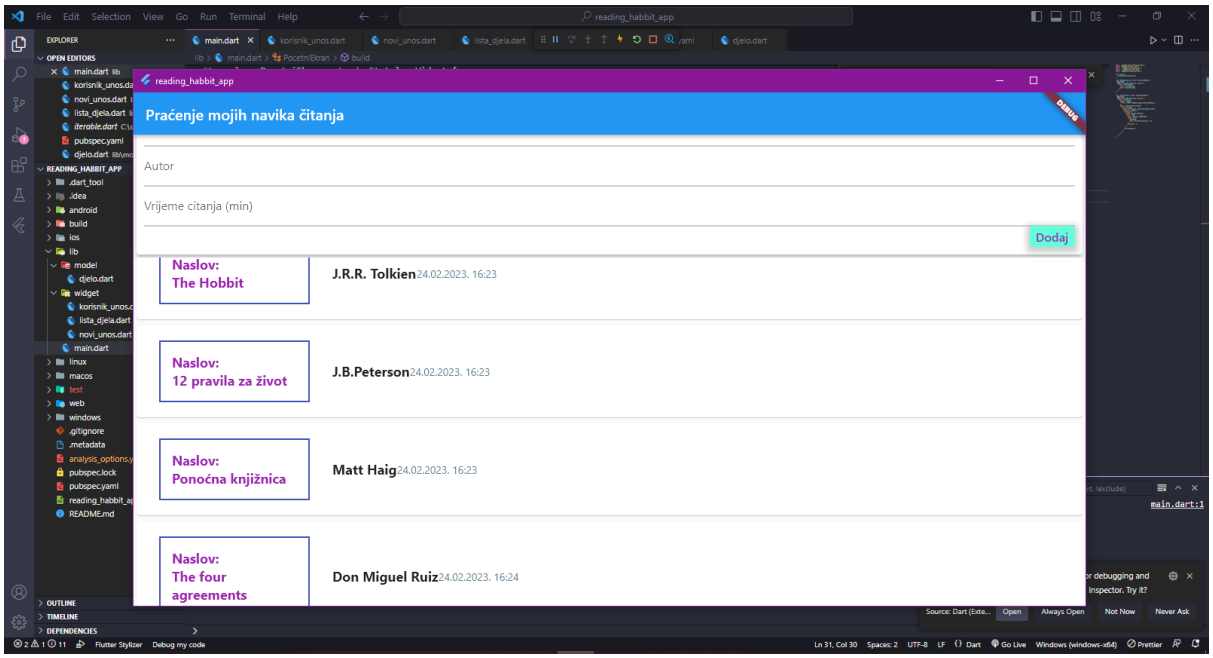
```
class KorisnikUnos extends StatefulWidget {
  @override
  State<KorisnikUnos> createState() => _KorisnikUnos();
}

class _KorisnikUnos extends State<KorisnikUnos> {
  @override
  final List<Djelo> _korisnikDjela = [];

  void _dodajNovoDjelo(String idDjela, String naslovDjela, String a-
utorDjela,
      double vrijemeCitanjaDjela) {
    final novoDjelo = Djelo(
      id: idDjela,
      naslov: naslovDjela,
      autor: autorDjela,
      vrijemeCitanja: vrijemeCitanjaDjela,
      danCitanja: DateTime.now(),
    );
    setState(() {
      _korisnikDjela.add(novoDjelo);
    });
  }

  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        NoviUnos(_dodajNovoDjelo),
        ListaDjela(_korisnikDjela),
      ],
    );
  }
}
```

Kod 10: Pomoćni widget za podizanje stanja (vlastoručna izrada, 2023.)

Našem glavnom programu konačno možemo proslijediti upravo razrađeni widget nakon čega naše rješenje izgleda ovako:



Slika 29: Aplikacija - jednostavni primjer (vlastoručna izrada, 2023.)



Slika 30: Aplikacija (unos) - jednostavni primjer (vlastoručna izrada, 2023.)

Usporedimo li prvotni crtež koji je poslužio kao model ove aplikacije i trenutno stanje rješenja, lako uviđamo manjak jedne od ključnih funkcionalnosti, grafikona. Za grafikon smo prvo kreirali novi widget file u kojem smo stvorili *stateless* widget koji će se sastojati od jedne kartice. Kartica će zatim imati redak koji će unutar sebe imati stupac koji će pak unutar sebe imati kontejner. Logiku stvaranja i popunjavanja ovog widgeta zasnivamo na dvije stavke, dan čitanja (datum) i vrijeme utrošeno na čitanje (minute). Za ovo će nam poslužiti **lista mapa**. Prisjetimo se, mape sadrže ***string*** i ***objekt*** te vrijednostima možemo pridodavati identifikatore

(zamislimo ključeve). Listu ćemo ostvariti stvaranjem ***getter*** metode proizvoljnog naziva koja gdje klasa ***List*** mora sadržavati konstruktor ***generate*** koji prima dva argumenta (dužina i funkcija). Funkcija će se izvršiti nad svakim objektom liste koji smo deklarirali ključnom riječi **indeks**. Unutar ove funkcije želimo vratiti dvije spomenute stavke (dan čitanja i vrijeme utrošeno na čitanje).

Grafikon možemo stvoriti jedino stvaranjem nove liste koja bi sadržavala navedene dvije stavke, stoga stvaramo novu listu proizvoljnog imena i stvaramo konstruktor. Kako je riječ o sedmodnevnom tjednu bitno nam je prepoznati o kojem danu je riječ kada unosimo podatke (npr. dan unošenja jest srijeda dok je djelo čitano prekjučer, u ponedjeljak). Ponovno ćemo iskoristiti ***DateTime.now*** od kojeg ćemo oduzeti spomenuti indeks uz pomoć metode naziva **oduzmi** (eng. ***subtract***). Metoda može sadržavati objekt **duljine** (eng. ***Duration***) u čijem konstruktoru možemo odrediti duljinu unošenjem imenovanog argumenta **dani** (eng. ***days***): indeks. Ispod svakog stupca navest ćemo skraćenicu dana u tjednu (na eng.).

Daljnjim korištenjem ***for*** petlje za svaki od objekata liste provjeravamo je li se dogodio na dan unošenja. Kako bismo bismo popunili odnosno obojali određeni stupac treba nam vrijednost, a ona će naravno biti izražena kao suma ukupno utrošenog vremena na čitanje pa je prikladno i nazovimo **sumaVremena** i zadajmo početno stanje 0.00.

```
class Grafikon extends StatelessWidget {
  @override
  final List<Djelo> nedavnoCitano;
  Grafikon(this.nedavnoCitano);
  List<Map<String, Object>> get grupiranNedavnoCitano {
    return List.generate(7, (index) {
      final dan = DateTime.now().subtract(
        Duration(days: index),
      );
      var sumaVremena = 0.00;
      for (var i = 0; i < nedavnoCitano.length; i++) {
        if (nedavnoCitano[i].danCitanja.day == dan.day &&
            nedavnoCitano[i].danCitanja.month == dan.month &&
            nedavnoCitano[i].danCitanja.year == dan.year) {
          sumaVremena += nedavnoCitano[i].vrijemeCitanja;
        }
```

```
    }
    return {
      'Dan': DateFormat.EEEE().format(dan).substring(0, 1),
      'Vrijeme': sumaVremena
    };
  }).reversed.toList();
}

double get maxProcitano {
  return grupiranNedavnoCitano.fold(0.0, (suma, item) {
    return suma + (item['Vrijeme'] as double);
  });
}

Widget build(BuildContext context) {
  return Card(
    elevation: 6,
    margin: EdgeInsets.all(20),
    child: Container(
      padding: EdgeInsets.all(5),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: grupiranNedavnoCitano.map((data) {
          return Flexible(
            fit: FlexFit.tight,
            child: GrafikonStupac(
                (data['Dan'] as String),
                (data['Vrijeme'] as double),
                maxProcitano == 0.0
                    ? 0.0
                    : (data['Vrijeme'] as double) / maxProcitano),
          );
        }).toList(),
      ),
```

```
      ),
    );
  }
}
```

Kod 11: Grafikon (vlastoručna izrada, 2023.)

U mainu dograbimo Grafikon(). ***Getter*** funkcija nam je potrebna i ovdje kako bismo prikazali sva pročitana djela unazad sedam dana, ne više.Za uspostavu ove restrikcije poslužit će nam ključna riječ gdje (eng. ***where***).

Prelazimo na stvaranje samih vidljivih stupaca te naravno kreiramo novu datoteku naziva **grafikon_stupac.dart** te uvodimo konstruktor istog naziva kao naš stateless widget s tri argumenta, oznaka (eng. ***label***) koja će označavati kraticu dana, **provedenoVrijeme** koje će biti prikazano iznad stupca i **postotakUkupnogVremena** koji će u usporedbi s provedenim vremenom čitanja drugih dana moći obojati stupac obzirom na to koliko je manje ili više pročitano. Unutar stupca dodajemo obični tekst kao prvo dijete, a služit će za izražavanje provedenog vremena čitanja. Drugo dijete, stog (eng. ***stack***) uvodimo po prvi puta, a poslužit će nam upravo onome čemu sama riječ asocira, a to je slaganje widgeta jednih na druge kako bi se dobio dojam dubine. Unutar samog stoga dodajemo dijete koje će poslužiti kao kontejner koji će bojati kontejner na kojemu leži, FractionallySizedBox koji prima argument heightFactor a to će naravno biti naš spomenuti postotakVremenaCitanja.

```
class GrafikonStupac extends StatelessWidget {
  @override
  final String label;
  final double provedenoVrijeme;
  final double postotakUkupnogVremena;
  GrafikonStupac(
      this.label, this.provedenoVrijeme,
this.postotakUkupnogVremena);
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        Text(provedenoVrijeme.toStringAsFixed(0)),
```

```
        SizedBox(
          height: 5,
        ),
        Container(
          height: 100,
          width: 25,
          child: Stack(
            children: <Widget>[
              Container(
                decoration: BoxDecoration(
                  border: Border.all(color: Colors.grey, width:
2.0),
                  borderRadius: BorderRadius.circular(10),
                  color: Color.fromRGBO(220, 220, 220, 1),
                ),
              ),
              FractionallySizedBox(
                  heightFactor: postotakUkupnogVremena,
                  child: Container(
                    decoration: BoxDecoration(
                      color: Theme.of(context).accentColor,
                      borderRadius: BorderRadius.circular(10),
                    ),
                  )),
            ],
          ),
        ),
        SizedBox(
          height: 5,
        ),
        Text(label)
      ],
    );  }}
```

Kod 12: Stupac grafikona (vlastoručna izrada, 2023.)

Zamislimo da je danas, na dan pisanja ili čitanja ovog paragrafa srijeda i želimo unesti djelo koje smo čitali u ponedjeljak. Trenutno to nije ostvarivo stoga kako bi sam grafikon u potpunosti imao smisla preostaje nam dodati mogućnost biranja datuma (unazad sedam dana od današnjeg dana). Buduća (eng. ***future***) klasa omogućuje nam da kreiramo objekt u budućnosti, a na primjeru našeg kalendara to se očituje na funkcionalnosti prikaza, ali ne i odabiru i pospremanju vrijednosti datuma. Stoga ovu klasu uvodimo dodavanjem ***.then()*** na kraju naše novostvorene funkcije proizvoljnog imena u kojoj je također sadržana krucijalna metoda ***showDatePicker***. Metoda može primiti više imenovanih argumenata, a oni koji su nama poslužili jesu kontekst (eng. ***context***), inicijalni datum(eng. ***initialDate***), prvi mogući datum (eng. ***firstDate***) i zadnji mogući datum (eng. ***lastDate***). Ukoliko je datum izabran mijenjamo stanje na način pohrane izabrane vrijednosti u prethodno definiranu varijablu, a u suprotnom ne vraćamo ništa. Spremljeni datum sada moramo iskoristiti, a to ćemo napraviti na jednostavan način dodavanja ovog argumenta našoj funkciji **dodajDjelo**, te konstruktoru u mainu navesti da traži i očekuje upravo ovaj argument tipa ***DateTime***.

```
void _pokaziKalendar() {
    showDatePicker(
            context: context,
            initialDate: DateTime.now(),
            firstDate: DateTime(2023),
            lastDate: DateTime.now())
        .then((zeljeniDatum) {
      if (zeljeniDatum == null) {
        return;
      }
      setState(() {
        _izabraniDatum = zeljeniDatum;
      });
    });
  }
```

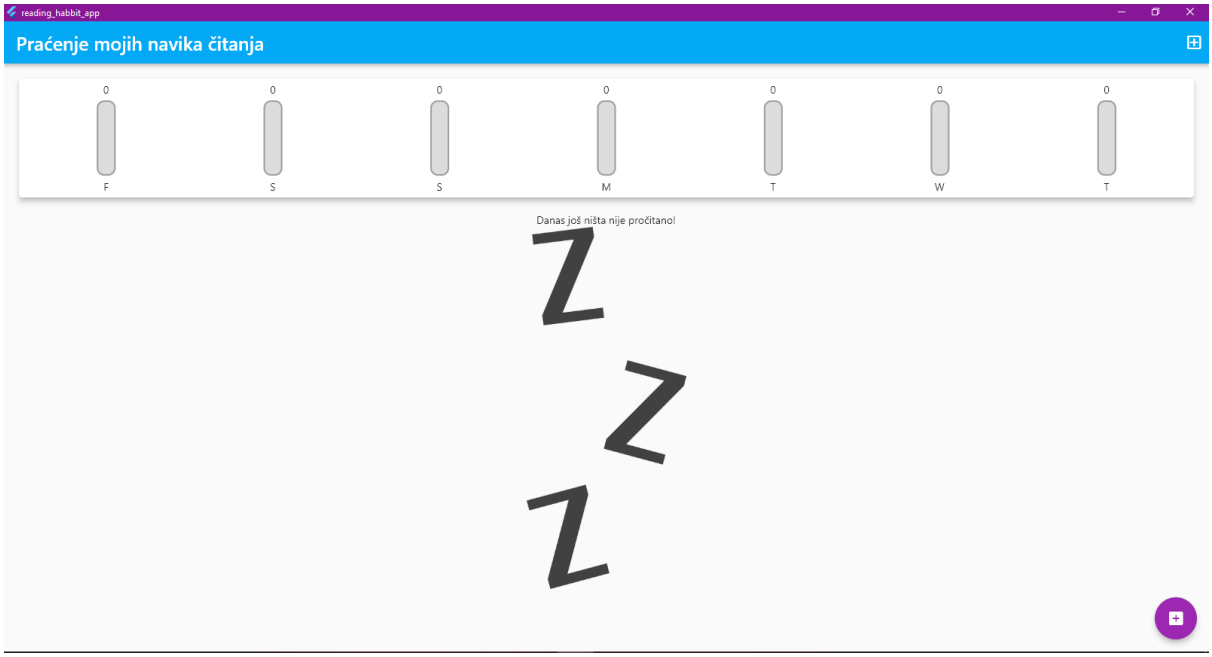
Kod 13: Prikaz kalendara za odabir (vlastoručna izrada, 2023.)

Uviđamo kako se na sučelju nalazi detalj koji nismo spomenuli, a to jest brisanje samog unosa i ažuriranje sučelja (liste zapisa i popunjenosti grafikona) na temelju te akcije. Brisanje

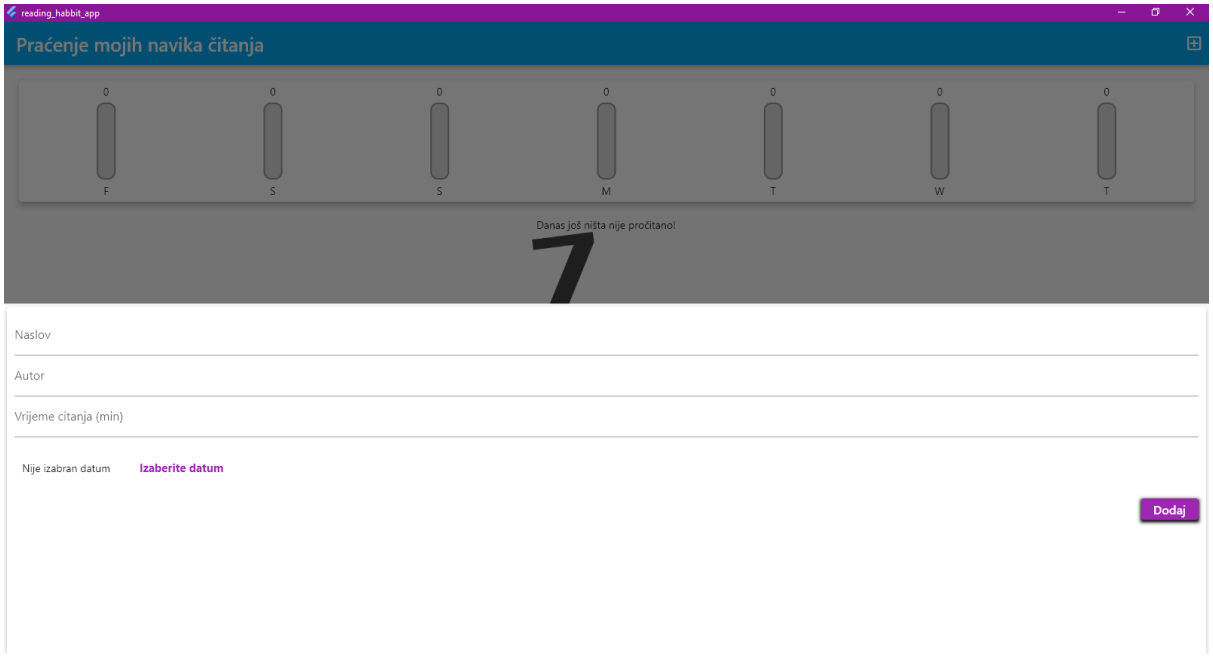
ćemo ostvariti na jednostavan način, započevši od pridodavanja identifikatora našim djelima. Identifikator će nam poslužiti za fluidniji pristup unesenom djelu iz različitih widgeta odakle moramo isto ukloniti. U našoj odvojenoj klasi djelo.dart unosimo novi argument, **String id**. Unutar main datoteke stvaramo naravno novu funkciju koja za argument prima jedan, novi identifikator. Ponovno ćemo iskoristiti ***setState*** koji će nam poslužiti da istovremeno postavimo novo stanje widgeta gdje pri pozivanju liste **korisnikDjela** ovoga puta koristimo funkciju **.**removeWhere. Ovime dajemo do znanja da želimo iz liste obrisati upravo onaj zapis koji označavamo pritiskom na gumb za brisanje. U metodi **dodajNovoDjelo** unutar main datoteke dodajemo imenovani argument **id**. Na kraju svega u datoteci lista_djela dodajemo novu funkciju naziva **obrisiUnos** jer u **ListaDjela** sada očekujemo ne samo funkciju unosa već i brisanja te ovime napokon završavamo razradu našeg jednostavnog praktičnog primjera.

```
void _izbrisiUnos(String id) {
    setState(() {
      _korisnikDjela.removeWhere((tx) => tx.id == id);
    });
  }
```

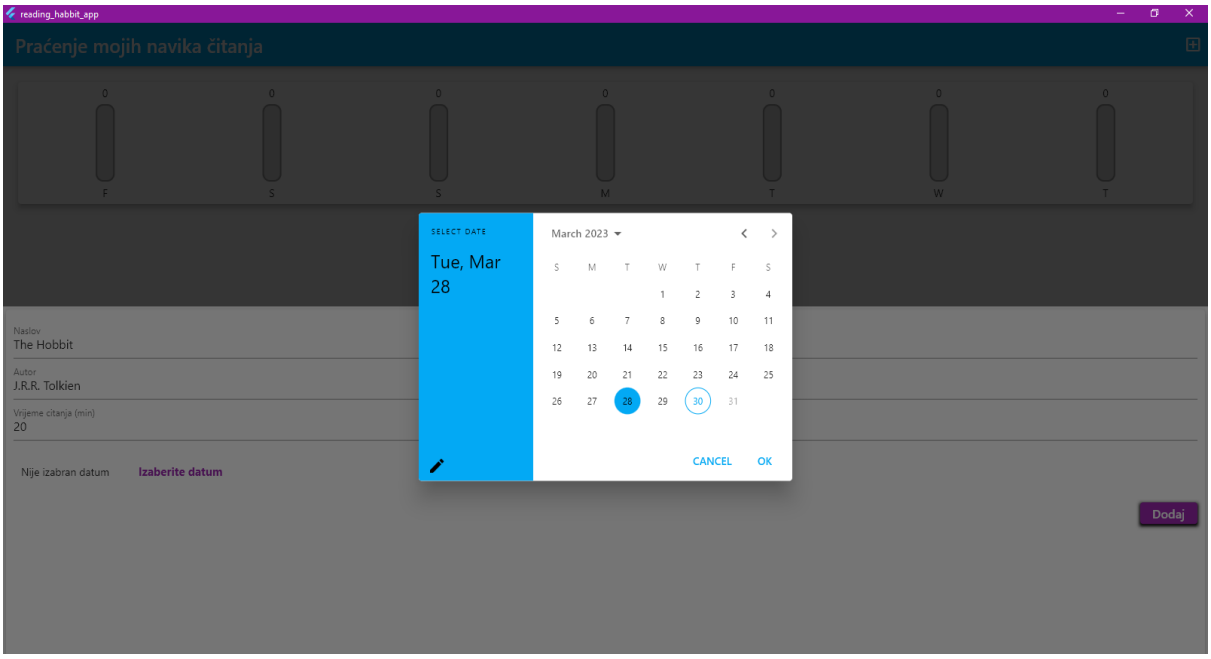
Kod 14: Brisanje djela (vlastoručna izrada, 2023.)



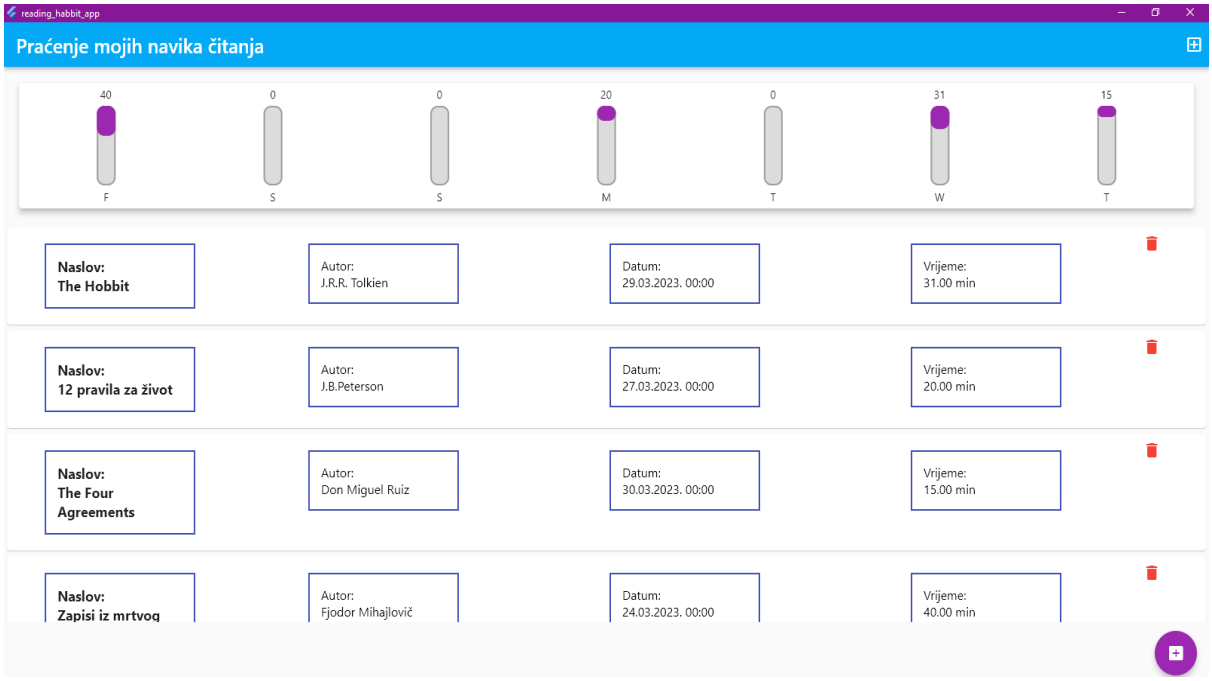
Slika 31: Jednostavni primjer - početni ekran (vlastoručna izrada, 2023.)



Slika 33: Jednostavni primjer- probni unos djela (vlastoručna izrada, 2023.)



Slika 32: Jednostavni primjer - odabir datuma (vlastoručna izrada, 2023.)

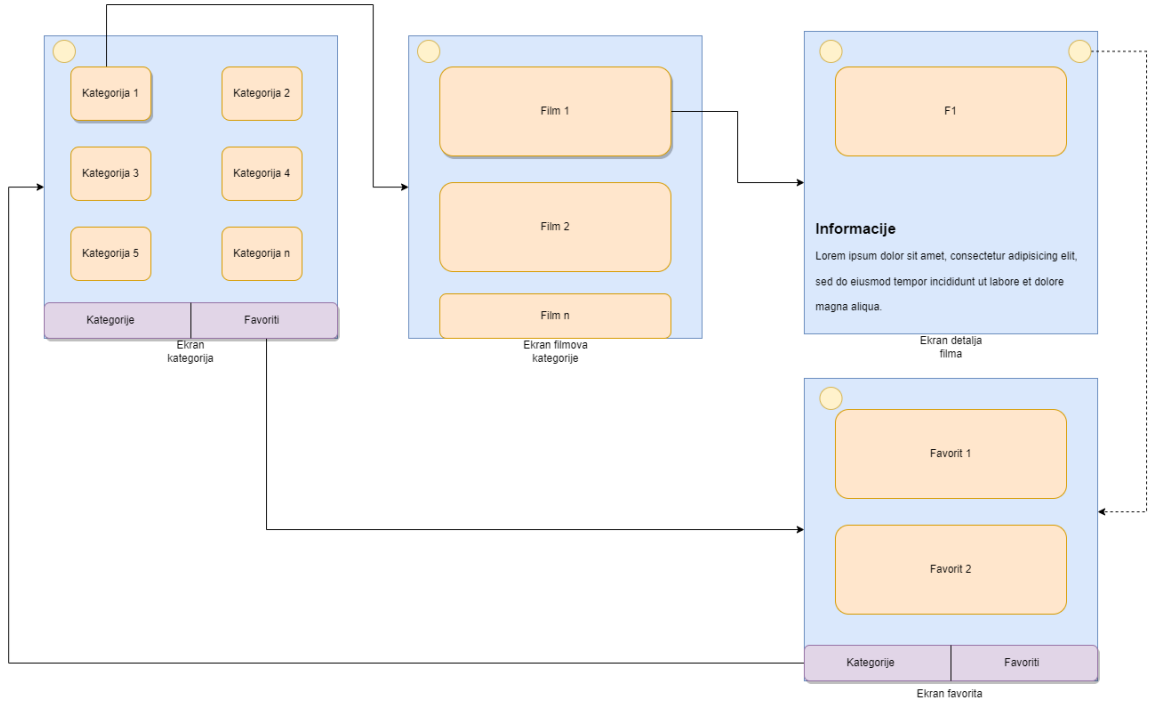


Slika 34: Jednostavni primjer - konačni prikaz (vlastoručna izrada, 2023.)

Uz manje promjene u izgledu i ponašanju samog rješenja došli smo do kraja našeg jedinstvenog stolnog programskog proizvoda u potpunosti proizvedenim uz pomoć tehnologije Flutter i programskog jezika Darta.

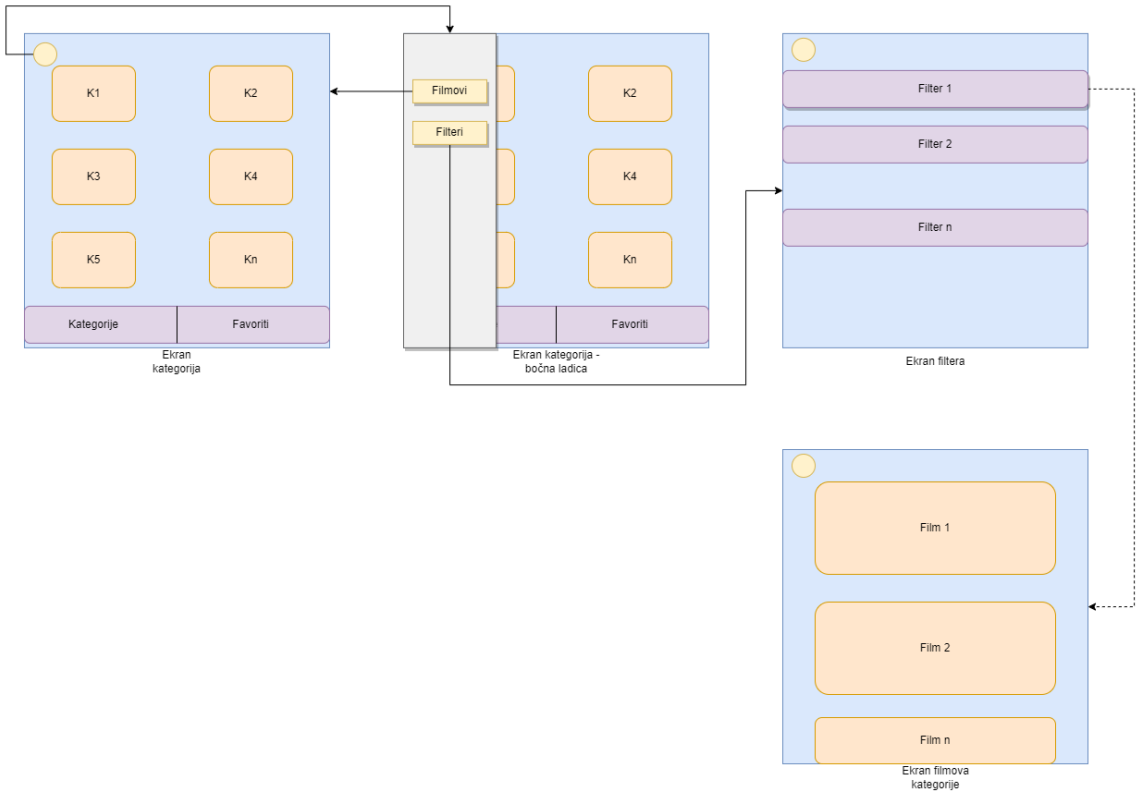
4. Razrada kompleksnog praktičnog primjera

Baš kao razradu prethodnog primjera i ovu započinjemo svojevrsnim planiranjem koje se bazira na raščlambi ekrana i elemenata na istima te njihove međusobne interakcije. Ostvarivanje najlakše postižemo crtanjem stoga prvi crtež u nastavku prikazuje osnovnu zamisao ekrana i prelaska s odabirom kategorije filma na sve filmove koji pripadaju istoj, a zatim pritiskom na željeni film dolazimo na detaljniji prikaz informacija. Uočimo kako su interakcije između navedenih ekrana označene kao akcijski tokovi (puna strelica) dok je jedini informacijski tok (isprekidana strelica) onaj koji postoji između ekrana detalja filma i ekrana favorita, a riječ je naravno o mogućnosti dodavanja filma u favorite pritiskom na gumbić.



Slika 35: Odnos ekrana u kompleksnom primjeru (vlastoručna izrada, 2023.)

Nadalje kao krajnji korisnik u mnogim aplikacijama sličnima ovoj želimo imati određeni način filtriranja i/ili sortiranja stoga je ta funkcionalnost uključena u vidu pritiska na gumbić na početnom zaslonu (u ovom slučaju ekranu kategorija referenciramo se kao početnom zaslonu) koji tada otvara tzv. „bočnu ladicu“. Isti zamišljeni postupak vidljiv je na sljedećoj slici:



Slika 36: Odnos ekrana u kompleksnom primjeru 2 (vlastoručna izrada, 2023.)

U ovome odnosu također postoji jedan informacijski tok, a on je na relaciji ekran filmova određene kategorije i ekrana filtera. Riječ je o tome da jednom primijenjeni filter, primjerice želimo vidjeti samo HollyWoodske filmove, odrazit će se na prikaz istih pod određenom kategorijom te svi filmovi koji na sebi nemaju svojstvo HollyWood bit će izbačeni.

Budući kako smo u prethodnom primjeru, a i prethodnim poglavljima uspostavili naš cjelokupni okoliš i radno okruženje, za početak rada na novome potrebno je jedino otvoriti *VS Code* te u terminalu upisati ***flutter create naziv_aplikacije***. Novostvorena mapa se otvara u zadanom sučelju te započinjemo s radom!

Počinjemo stvaranjem prve datoteke, ujedno i početne, a to je naravno ekran kategorija pod nazivome ekran_kategorije.dart. Ondje ćemo stvoriti istoimeni *Stateless Widget*, prisjetimo se, ovi *widgeti* su statični i ne grade se iznova tijekom trajanja aplikacije za razliku od *Stateful Widgeta* koji su naravno dinamični i svoje stanje mijenjaju za vrijeme izvođenja aplikacije. Unosimo do sada još ne viđeni widget, ***GridView***. Ovaj widget omogućuje nam prikaz widgeta u obliku rešetke, vrlo popularni prikaz korišten u svim internetskim trgovinama za prikaz proizvoda. GridView od nas očekuje jedan parametar, ***gridDelegate*** kojem pak prosljeđujemo novi widget podugačkog imena,

SliverGridDelegateWithFixedCrossAxisCount. Razlomimo widget na pojmove, ***Sliver*** označava mogućnost skrolanja widgeta, ***Grid Delegate*** označava da će nam sadržaj biti prikazan u obliku rešetke, a ***With Fixed Cross Axis Count*** označava da će taj sadržaj zauzimati njemu sav mogući dodijeljeni prostor na ekranu. Sljedeći argumenti proslijeđeni ovom widgetu tiču se organizacije sadržaja na samome ekranu.

```
child: GridView(
        padding: EdgeInsets.all(25),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 2,
            childAspectRatio: 3 / 2,
            crossAxisSpacing: 22,
            mainAxisSpacing: 30),
```

Kod 15: Prikaz rešetke (vlastoručna izrada, 2023.)

Kako bismo uopće razlučili koje kategorije želimo imati te iste prikazali moramo ih definirati, stoga to činimo stvaranjem nove datoteke jednostavnog naziva kategorija.dart u kojoj ćemo imati samo jednu jednostavnu klasu. Klasa Kategorija sadržava **naslov** i **boju** te uvijek prisutni **id**.

```
class Kategorija {
  @required
  final String? id;
  @required
  final String? naslov;
  @required
  final Color? boja;
  const Kategorija({this.id, this.naslov, this.boja});
}
```

Kod 16: Klasa Kategorija (vlastoručna izrada, 2023.)

Nova datoteka naziva ***dummy.dart*** poslužit će nam za sadržavanje cjelokupnog sadržaja svih kategorija, a kasnije i filmova.

```
const DUMMY_KATEGORIJE = const [
  Kategorija(
    id: 'c1',
```

```
    naslov: 'Triler',
    boja: Colors.purple,
  ),
  Kategorija(
    id: 'c2',
    naslov: 'Akcija',
    boja: Colors.red,
  ),
```

Kod 17: Poziv konstruktora Kategorija (vlastoručna izrada, 2023.)

Kako bismo kod održali što urednijim primjenjujemo dobru praksu *outsorcanja* kompleksnijih widgeta u zasebnu datoteku, u ovom slučaju riječ je o stavki kategorije, odnosno **kategorija_stavka.dart** datoteci koja će se pobrinuti o vizualnoj reprezentaciji prethodno definiranih kategorija. Primarni widget koji nam je poslužio u tome jest ***InkWell***, koji nam dopušta korištenje slušača te ga je moguće detaljno uređivati. Ovi slušači omogućuju nam prosljeđivanje željene funkcije, u ovome slučaju ona će nam poslužiti da na dodir (argument onTap) ovog widgeta, navigiramo na novu stranicu.

Dotaknimo se po prvi put naše main.dart datoteke. U njoj želimo definirati naš početni ekran, ono što ćemo kao korisnici ove aplikacije vidjeti prvo pri pokretanju stoga argumentu ***home*** dodjeljujemo klasu **EkranKategorije**.

```
void main() => runApp(
      ProviderScope(
        child: MyApp(),
      ),
    );

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'VidFilmovi',
      theme: ThemeData(
        primarySwatch: Colors.blue,
```

```
      ),
      home: EkranKategorije(),
    );
  }
}
```

Kod 18: Main kompleksnog primjera (vlastoručna izrada, 2023.)

Kategorije sada želimo i popuniti te povezati s određenim filmovima, pa na jednak način započinjemo, a to jest izradom nove datoteke **filmovi_ekran.dart**. Prije negoli započnemo s njegovom razradom, vratimo se ekranu kategorija odakle započinje naše kretanje. Ondje stvaramo novu funkciju **izaberiKategoriju**. Uvodimo novi widget, ***Navigator***. Preko ovog widgeta možemo pristupiti njegovim metodama, dvjema mehanizmima, a oni su ***push***, koji nam omogućuje da novu stranicu stavimo na vrh i nju prikazujemo, ili ***pop***, koji nam omogućuje da vraćanje na stranicu koja je prethodila onoj na kojoj se nalazimo. U našem slučaju stoga koristimo ***Navigator.push*** u kojemu instanciramo novu klasu ***MaterialPageRoute*** u čijem konstruktoru builder parametra prosljeđujemo onu stranicu na koju želimo navigirati, u ovome slučaju to je ekran filmova stoga isti pozivamo upisivanjem **FilmoviEkran**.

```
void _izaberiKategoriju(BuildContext context, Kategorija kategorija)
{
    final filterKategorija = widget.dostupniFilmovi!
        .where((film) => film.kategorije!.contains(kategorija.id))
        .toList();
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (ctx) => FilmoviEkran(
          naslov: kategorija.naslov!,
          filmovi: filterKategorija,
        ),
      ),
    );
  }
```

Kod 19: Funkcija odabira kategorije (vlastoručna izrada, 2023.)

Naravno sada je potrebno i prikazati naše filmove pod određenom kategorijom. Baš kao i za kategorije, izradu filmova započet ćemo izradom modela. Novu **film.dart** datoteku smjestit ćemo unutar novostvorene mape ***models***, u istu smo prenesli kategorija.dart datoteku. Model će sadržavati jednu klasu i konstruktor klase, kompleksniji od one u kategoriji. Kreirat ćemo objekte poput **id**, **naslova**, **redatelja**, **trajanja**, **slike** itd. Za filtraciju filmova poslužit će nam booleani poput **jeHollyWood** ili **jeEuropski**, a dodajemo i jedan enum naziva **Tezina** čija uloga je korisniku dati do znanja može li se opustiti uz film ili je on pak „teže probavljiv" i potrebno ga je aktivno gledati.

```
enum Tezina { Lak, Osrednji, Tezak }

class Film {
  @required
  final String? id;
  @required
  final List<String>? kategorije;
  @required
  final String? naslov;
  @required
  final int? trajanje;
  @required
  final String? redatelj;
  @required
  final int? godina;
  @required
  final List<String>? uloge;
  @required
  final Tezina? tezina;
  @required
  final bool? jeHollyWood;
  @required
  final bool? jeEuropski;
  @required
  final String? slikaUrl;
```

```
  @required
  final String? opis;
  const Film(
      {this.id,
      this.kategorije,
      this.naslov,
      this.trajanje,
      this.redatelj,
      this.godina,
      this.uloge,
      this.tezina,
      this.jeHollyWood,
      this.jeEuropski,
      this.slikaUrl,
      this.opis});
}
```

Kod 20: Klasa filma (vlastoručna izrada, 2023.)

Nakon ovoga vraćamo se u **dummy.dart** gdje na isti princip unosimo kao i kategorije, sada unosimo informacije o našim željenim filmovima.

```
const DUMMY_MOVIES = const [
  Film(
      id: '1',
      kategorije: ['c1', 'c3'],
      naslov: 'Kad jaganjci utihnu',
      trajanje: 118,
      redatelj: 'Jonathan Demme',
      godina: 1991,
        uloge: ['Jodie Foster', 'Anthony Hopkins,', 'Lawrence A.
Bonney'],
      tezina: Tezina.Tezak,
      jeHollyWood: true,
      jeEuropski: false,
      slikaUrl:
```

```
          'https://s3.amazonaws.com/static.rogerebert.com/uploads/blog_post/primary_image/features/timeless-horror-the-25th-anniversary-of-the-silence-of-the-lambs/silence-of-the-lambs.jpg',
      opis:
              'Mladi kadet FBI-a mora dobiti pomoć zatvorenog i manipulativnog ubojice kanibala kako bi uhvatio još jednog serijskog ubojicu, luđaka koji dere kožu svojim žrtvama.'),
```

Kod 21: Primjer unosa filma (vlastoručna izrada, 2023.)

Nadalje stvaramo našu sljedeću stavku, baš kao i kategorija stavka, **film_stavka.dart** pobrinut će se o vizualnoj reprezentaciji prethodno definiranih filmova. Način na koji to ostvarujemo jest uvođenjem ***Stack*** widgeta koji nam omogućuje da postavimo sliku, a na nju željene informacije koje će u ovome slučaju biti godina prikazivanja filma, trajanje filma te težina filma. Novitet koji predstavljamo pri pozivanju slika jest ***FadeInImage*** klasa koja, kako ime nalaže, služi tome da se na zaslonu aplikacije ne vidi okvir slike u kojemu će se ona tek nakon nekog vremena pojaviti te također omogućuje rezerviranje mjesta (eng. *placeholder*) mjesta na kojemu bi se slika trebala pokazati.

```
child: FadeInImage(
        placeholder: MemoryImage(kTransparentImage),
        image: NetworkImage(film.slikaUrl!),
        fit: BoxFit.cover,
        width: double.infinity,
        height: 800,
       ),
```

Slika 37: FadeInImage (vlastoručna izrada, 2023.)

Kako bismo kod održali čitljivijim, stvorit ćemo novi widget **film_stavka_meta.dart**. On će nam poslužiti za prikaz ikone i teksta kako iste ne bismo morali iznova kopirati unutar stavke filma već jednostavno pozvali ovaj widget te tako smanjili naš kod i redundanciju.

```
class FilmStavkaMeta extends StatelessWidget {
  const FilmStavkaMeta({super.key, this.ikona, this.label});
  final IconData? ikona;
  final String? label;
  @override
  Widget build(BuildContext context) {
```

```
    return Row(
      children: [
        Icon(
          ikona,
          size: 36,
          color: Colors.white,
        ),
        SizedBox(width: 8),
        Text(
          label!,
          style: TextStyle(color: Colors.white, fontSize: 24),
        )
      ],
    );
  }
}
```

Kod 22: Meta podaci filma (vlastoručna izrada, 2023.)

Vrijeme je da svaki od filmova učinimo interaktivnim, odnosno klikom na željeni film na ekranu svih filmova određene kategorije navigiramo na novi ekran, ekran detalja o istome. Kao i uvijek za početak nam je potreban novi ekran stoga kreiramo **film_detalji_ekran.dart**.

Pošto uviđamo postojanje već više ekrana, ali i widgeta, naše datoteke premještamo u za to odgovarajuće novostvorene mape. Mapa modeli (eng. ***models***) sadrži dvije datoteke s klasama, mapa ekrani (eng. ***screens***) sadrži datoteke ekran kategorija, ekran filmova, ekran detalja filma te mapa widgeti (eng. ***widgets***) sadrži datoteke stavka kategorije, stavka filma i meta stavke filma.

Vraćamo se na naš widget film_stavka odakle bi naša interakcija trebala započeti. Spomenuli smo kako *InkWell* widgetu možemo dodati slušače, stoga mu dodajemo već poznati imenovani argument ***onTap***, a njemu ćemo sada proslijediti novostvorenu funkciju **onOdabirFilm** koja je tipa **Film**. Jednom kada smo to učinili moramo samome ekranu reći da želimo navigirati na novi, stoga to i činimo u datoteci **filmovi_ekran**. Saznali smo kako za navigaciju na novi ekran koristimo ključne riječi Navigator.push stoga jednakim principom ovdje stvaramo novu funkciju:

```
void odabirFilma(BuildContext context, Film film) {
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (ctx) => FilmDetaljEkran(
          film: film,
        ),
      ),
    );
  }
```

Kod 23: Funkcija navigacije na detalje filma (vlastoručna izrada, 2023.)

Funkciju za prelazak na novi ekran proslijedit ćemo kao parametar funkciji onOdabirFilma koja se sada nalazi unutar **IF izjave** koja ujedno služi za provjeru je li se na našemu ekranu uopće pokazao koji od filmova.

```
if (filmovi.isNotEmpty) {
      sadrzaj = GridView.builder(
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
          crossAxisSpacing: 10.0,
          mainAxisSpacing: 10.0,
        ),
        itemCount: filmovi.length,
        itemBuilder: (ctx, index) => FilmStavka(
          film: filmovi[index],
          onOdabirFilm: (film) {
            odabirFilma(context, film);
          },
        ),
      );
    }
```

Kod 24: Dodjela navigacije (vlastoručna izrada, 2023.)

Jednom uspostavljena osnovna navigacija među ovim ekranima dopušta nam da započnemo s radom na kartici na dnu aplikacije koja bi trebala prikazivati i omogućiti odabir između kategorija, koje smo uspješno riješili, i favorita, ekrana kojeg tek trebamo izraditi. Baš kao i sve do sada, započinjemo stvaranjem nove datoteke ekrana, no ovoga puta umjesto

logičnog naziva koji bi trebao uslijediti, a to jest favoriti_ekran, zapravo izrađujemo **tabovi_ekran.dart**. Naime ideja iza ovoga jest da ćemo u stvarnosti imati jedan ekran koji će mijenjati svoje stanje obzirom na to što je na njemu izabrano, kategorije ili favoriti. Promjena stanja odmah ukazuje na korištenje ***StatefulWidget***. Uvodimo još jednu novinu, a to je widget ***bottomNavigationBar***. Kako ime nalaže riječ je o traci koja se nalazi na dnu ekrana aplikacije te osim uvijek prisutne mogućnosti uređivanja, ona može zaprimiti funkciju aktiviranu na dodir i stavke (eng. ***items***). Naše stavke bit će spomenute kategorije i favoriti stoga listi stavki uz pomoć ***BottomNavigationBarItem*** dodjeljujemo ikonu i naziv.

```
return Scaffold(
      appBar: AppBar(
        title: Text(aktivnaStranicaNaslov),
      ),
      drawer: MainLadica(
        onIzaberiEkran: _postaviEkran,
      ),
      body: aktivnaStranica,
      bottomNavigationBar: BottomNavigationBar(
        onTap: _odaberiStranicu,
        currentIndex: _odabranaStranicaIndex,
        items: [
            BottomNavigationBarItem(icon: Icon(Icons.movie), label: 
'Kategorije'),
             BottomNavigationBarItem(icon: Icon(Icons.star), label: 
'Favoriti'),
        ],
      ),
    );
```

Kod 25: Donja traka stavki (vlastoručna izrada, 2023.)

Uočavamo jedan nerazriješeni argument u prethodnome kodu, ***currentIndex***. On se trenutno poziva na varijablu odabranaStranicaIndex, a istoj je dodijeljena vrijednost nule. Ukoliko dođe do promjene indeksa, stranica se prebacuje, odnosno jedna se zatvara, a druga podiže. Ova izjava označava postojanje promjene stanja, koje ujedno mijenja i indeks, što je vidljivo u sljedećem isječku:

```
void _odaberiStranicu(int index) {
```

```
    setState(() {
      _odabranaStranicaIndex = index;
    }); }
```

Kod 26: Promjena indeksa stranice (vlastoručna izrada, 2023.)

Kako bi ekran favorita bio popunjen, moramo naravno stvoriti mogućnost dodavanja filmova u favorite, a isto će biti ostvareno na ekranu detalja filma. Za početak smo u gornjoj naslovnoj traci na ekranu detalja filma dodali ikonu zvjezdice, klikom na istu film će biti dodan ili izbačen iz favoriziranih filmova. Naše rješenje započinje uvođenjem davatelja usluga (eng. *providers*), a to činimo preuzimanjem vanjskog paketa pod nazivom ***Riverpod***, a isti instaliramo unutar samog terminala upisom ***flutter pub add flutter_riverpod***. Slijedimo princip organizacije našeg koda odnosno naših datoteka unutar mapa stoga stvaramo novu naziva ***providers***. Ondje ćemo za početak stvoriti datoteku koja će nam poslužiti u manipuliranja stanja ekrana filtera, datoteku **favoriti_provider.dart**. Upoznati smo kako stvaranjem novih klasa možemo proširivati postojeće, to je najbolje vidljivo u stvaranju *Stateful* ili *Stateless* widgeta, pa tako i naša nova klasa **FavoritiFilmoviNotifier** proširuje već postojeću klasu ***StateNotifier*** s kojom se prvi puta susrećemo. Ova klasa koristi se za upravljanje stanjima i obavještavanje u promjeni stanja te je različita od klase ***Provider*** koju ćemo kasnije uvesti. Unutar klase dodajemo jednu jedinu metodu koja će služiti za zamjenu filmova unutar prazne liste koju smo postavili kao početnu vrijednost. Vršimo provjeru liste sadrži li ona već film u sebi odnosno je li film već favoriziran na način da pristupimo stanju uz metodu **sadrži** (eng. *contains*). Ako film već je favoriziran mi mijenjamo stanje u kojemu id filma neće biti isti što će značiti da se isti više neće nalaziti u favoritima, a inače kopiramo trenutno stanje te u njega dodajemo naš novo favorizirani film.

```
class FavoritiFilmoviNotifier extends StateNotifier<List<Film>> {
  FavoritiFilmoviNotifier() : super([]);
  bool favoritiFilmoviStatus(Film film) {
    final filmJeFavorit = state.contains(film);
    if (filmJeFavorit) {
      state = state.where((film2) => film2.id != film.id).toList();
      return false;
    } else {
      state = [...state, film];
      return true;
    }
  }
```

```
}
```

Kod 27: Notifier Favorita (vlastoručna izrada, 2023.)

Uspješno riješeni prikaz kategorija, filmova pojedine kategorije, detalja pojedinog filma te mogućnosti dodavanja željenog u favorite i njihov prikaz doveli su nas do zadnjeg ekrana, ekrana filtera. Prisjetimo li se naših crteža s početka ovoga poglavlja uviđamo kako ekranu filtera pristupamo preko jednog polovičnog ekrana odnosno tzv. bočne ladice, pa naš rad zapravo započinje ondje. Pogodnost je što je ovaj widget zapravo u potpunosti ugrađen u Flutteru. Kako bismo kod održali čišćim i čitljivim našu ladicu izgradit ćemo u samostalnoj datoteci, **ladica.dart**. Widget koji će se ondje nalaziti naziva je ***Drawer***, a njemu možemo dodijeliti djecu ***DrawerHeader***, ono što će biti prikazano na vrhu ladice, ***ListTile***, ono što ćemo prikazivati u recima ispod zaglavlja. Primjer jedne pločice popisa vidljiv je u nastavku:

```
ListTile(
            tileColor: Colors.grey.withOpacity(0.4),
            leading: Icon(
              Icons.slideshow_rounded,
              size: 30,
              color: Colors.grey.shade800,
            ),
            title: Text(
              'Filmovi',
                style: TextStyle(color: Colors.blueAccent, fontSize:
30),
            ),
            onTap: () {
              onIzaberiEkran('Filmovi');
            },
          ),
```

Kod 28: Pločica filmova u bočnoj ladici (vlastoručna izrada, 2023.)

Kako bismo uspjeli navigirati na novi ekran jednom kada pritisnemo na pločicu unutar ladice moram uvesti i funkciju koja će to ispravno odrađivati za nas. Jednom pritisnuta pločica filmova mora nas odvesti na ekran kategorija, a pločica filtera naravno na ekran filtera. U kodu iznad vidljiv je već poziv te funkcije u dijelu ***onTap*** a funkcija jest **onIzaberiEkran** koja zahtjeva jedan parametar, u ovom slučaju to su **Filmovi**, što bi značilo da pritiskom na ovu pločicu

dolazimo na ekran s prikazom svih kategorija svih filmova. No ovu smo funkciju definirali unutar druge datoteke, a riječ je o ekranu tabova. Ondje naša **MainLadica** zahtjeva funkciju uz ostale parametre, poput tijela, a isto je vidljivo u prethodno navedenom kodu (Kod 23 Donja traka stavi) da je riječ o funkciji **postaviEkran** koja je proslijđena kao argument.

```
void _postaviEkran(String identifikator) async {
    Navigator.of(context).pop();
    if (identifikator == 'Filteri') {
       final rezultat = await Navigator.of(context).push<Map<Filter,
bool>>(
         MaterialPageRoute(
           builder: (ctx) => FilteriEkran(),
         ),
       );
    }
  }
```

Kod 29: Funkcija postave ekrana (vlastoručna izrada, 2023.)

Nova ključna riječ upada u oko, a riječ je o ***async***. Koristimo je pri izradi ove funkcije kako bismo izbjegli smrzavanje ekrana pri promjenama. Početkom funkcije gasimo ekran, no ne bilo koji već ekran filtera. Naime naše početno i željeno stanje jest ono prikaza svih kategorija, stoga taj ekran možemo smatrati svojevrsnom podlogom na koju se postavljaju (eng. *push*) ostali ekrani, a kada iste želimo maknuti, radimo ***pop*** te nas ta akcija zpr. nazad do kategorija. No ova radnja nas trenutno ne vodi nikamo, ekran filtera još nije izrađen, stoga stvaramo novu datoteku naziva **filteri_ekran.dart**. Ovdje još jednom uvodimo u potpunosti novi widget ***SwitchListTile***, a on je verzija widgeta prethodno korištenog u bočnoj ladici, a riječ je o ***ListTile***. No za razliku od njega naš novouvedeni widget omogućava i prebacivanje između stanja u vidu pokrenuto i ugašeno (eng. *toggle on and off*) što će nam u ovome slučaju odlično poslužiti obzirom da filtere želimo moći gasiti i paliti. Uviđamo kako bi najsmislenije rješenje za ovaj postupak bilo uvođenje svojevrsnog booleana koji je **istinit** kada je filter **upaljen** i naravno **false** kada je **ugašen**. Kako ćemo i ove parametre morati proslijediti na ostale ekrane nailazimo na savršeno podneblje za uvođenje novog davatelja usluga, **filteri_provider.dart**. Ondje stvaramo enum koji će sadržavati vrijednosti za raspoznaju filmova poput onih HollyWoodskih ili pak onih Europskih.

```
enum Filter { jeHollyWood, jeEuropski }
```

Kod 30: Enum Filter (vlastoručna izrada, 2023.)

Stvaramo novu klasu, **FavoritiFilmoviNotifier** koja ponovno proširuje ***StateNotifier***. Klasa želi bar jedan proslijeđeni argument, a mi ćemo još jednom koristiti mapu odnosno spoj enuma Filter i ključa tipa bool. Preciznije to znači kako vrijednosti te mape mogu biti **jeHollyWood** ***true*** / ***false*** i **jeEuropski** ***true*** / ***false***, a to je upravo kombinacija logike koju želimo. Usputno ćemo instancirati početne vrijednosti ovih filtera tako da ne budu aplicirani, tj. neka su vrijednosti *false*, nakon čega ćemo stvoriti metodu koje nam služi za manipulaciju nad filterima. **PostaviFilter** kao argument želi upravo enum **Fitler** i bool **jeAktivan** na temelju čega će odrediti je li zadani filter aktiviran ili nije. Stanje (eng. *state*) sada postavljamo u trenutno postojeće stanje kopiranjem, odnosno korištenje tzv. **operatora širenja** (eng. *spread operator*) te u kopiranom novom stanju mijenjamo vrijednost filtera.

```
class FilteriNotifier extends StateNotifier<Map<Filter, bool>> {
  FilteriNotifier()
      : super({Filter.jeHollyWood: false, Filter.jeEuropski: false});
  void postaviFiltere(Map<Filter, bool> izabraniFilteri) {
    state = izabraniFilteri;
  }

  void postaviFilter(Filter filter, bool jeAktivan) {
    state = {
      ...state,
      filter: jeAktivan,
    };
  }
}
```

Kod 31: Notifier za manipulaciju stanja filtera (vlastoručna izrada, 2023.)

Logika provjere i promjene koji od filmova zadovoljavaju postavljene filtere također se nalazi u ovoj datoteci odakle joj se može pristupiti globalno, umjesto da je smještena unutar ekrana filtera.

```
final filteriProvider =
    StateNotifierProvider<FilteriNotifier, Map<Filter, bool>>(
        (ref) => FilteriNotifier());
```

```
final filtriraniFilmoviProvider = Provider((ref) {
  final filmovi = ref.watch(filmoviProvider);
  final aktivniFilteri = ref.watch(filteriProvider);
  return filmovi.where((film) {
    if (aktivniFilteri[Filter.jeHollyWood]! && !film.jeHollyWood!) {
      return false;
    }
    if (aktivniFilteri[Filter.jeEuropski]! && !film.jeEuropski!) {
      return false;
    }
    return true;
  }).toList();
});
```

Kod 32: Provider filmova izabranih na temelju apliciranih filtera (vlastoručna izrada, 2023.)

Možda se čini kao mnogo bespotrebnog posla, no ovaj pristup koristi se u svim kompleksnijim Flutter aplikacijama te kao svojevrsni primjer dobre prakse primijenili smo ga u našem slučaju. Vratimo se nazad na naš ekran filtera gdje sada umjesto prvotno postavljenog ***StatelessWidget*** isti moramo zamjeniti tzv ***ConsumerWidget***. Ovaj widget poseban je prema svojoj namjeni koja je da konzumira promjene od davatelja usluga odnosno ***Provider*** widgeta. Naš novi widget osluškivat će davatelja usluga te u slučaju da načuje promjenu u podacima automatski će se samostalno izgraditi i ne samo to… do sada smo uvijek izgrađivali cjelokupno naše drvce widgeta za pojedini ekran, dok uz ovaj widget možemo specificirati samo određeni dio na kojemu želimo da se ponovna izgradnja učini, pa se tako vraćamo na ***SwitcListTile***.

```
 SwitchListTile(
              activeColor: Colors.purple,
              value: aktivniFilteri[Filter.jeHollyWood]!,
              onChanged: (isChecked) {
                ref
                    .read(filteriProvider.notifier)
                    .postaviFilter(Filter.jeHollyWood, isChecked);
              },
              title: Text(
                'HollyWood',
```

```
                  style: TextStyle(color: Colors.black, fontSize: 25),
                ),
                subtitle: Text('-prikaži samo HollyWoodske filmove',
                      style: TextStyle(color: Colors.black, fontSize:
14)),
            ),
```

Kod 33: SwitchListTile sa referencom na provider i metodu manipulacije (vlastoručna izrada, 2023.)

Uviđamo neobjašnjeni argument, a to jest ***onChanged*** koji provjerava je li nešto provjereno (eng. *is checked*), a u ovome slučaju riječ je o argumentu **ref** (referenci) preko kojega možemo pristupiti metodama poput **čitanja** (eng. *read*) pomoću čega čitamo promjene unutar našeg davatelja usluga **filteriProvider** te na temelju njega aktiviramo metodu **postaviFilter**.

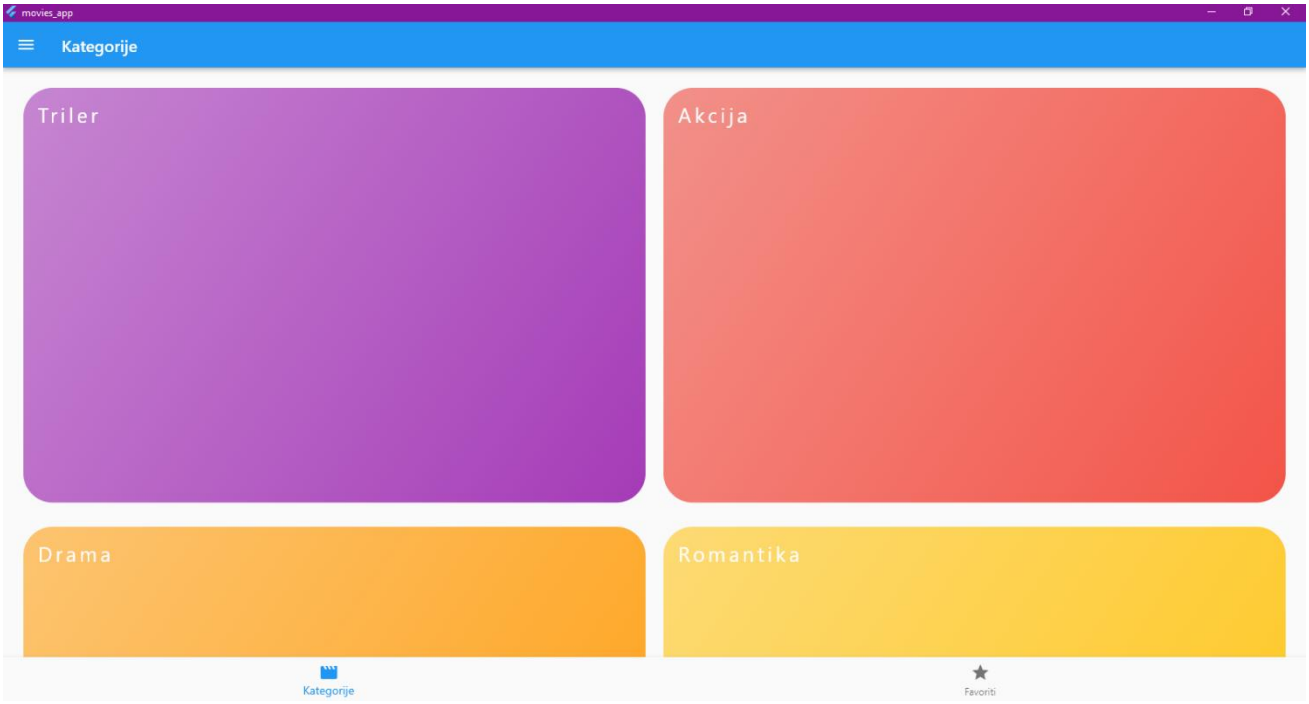
Posljednja preinaka koja je vezana uz uspostavu filtera jest ona koja se odvija na ekranu tabova, a riječ je o pretvaranju ***StatefulWidget*** u ***ConsumerStatefulWidget***.

Sav naš rad mogao bi biti uzalud ako ne učinimo ključnu stavku, a to jest da u našoj **main** datoteci obgrlimo **app** widget s **ProviderScope**, kako bismo korištenje providera učinili mogućim.

```
void main() => runApp(
      ProviderScope(
        child: MyApp(),
      ),
    );
```

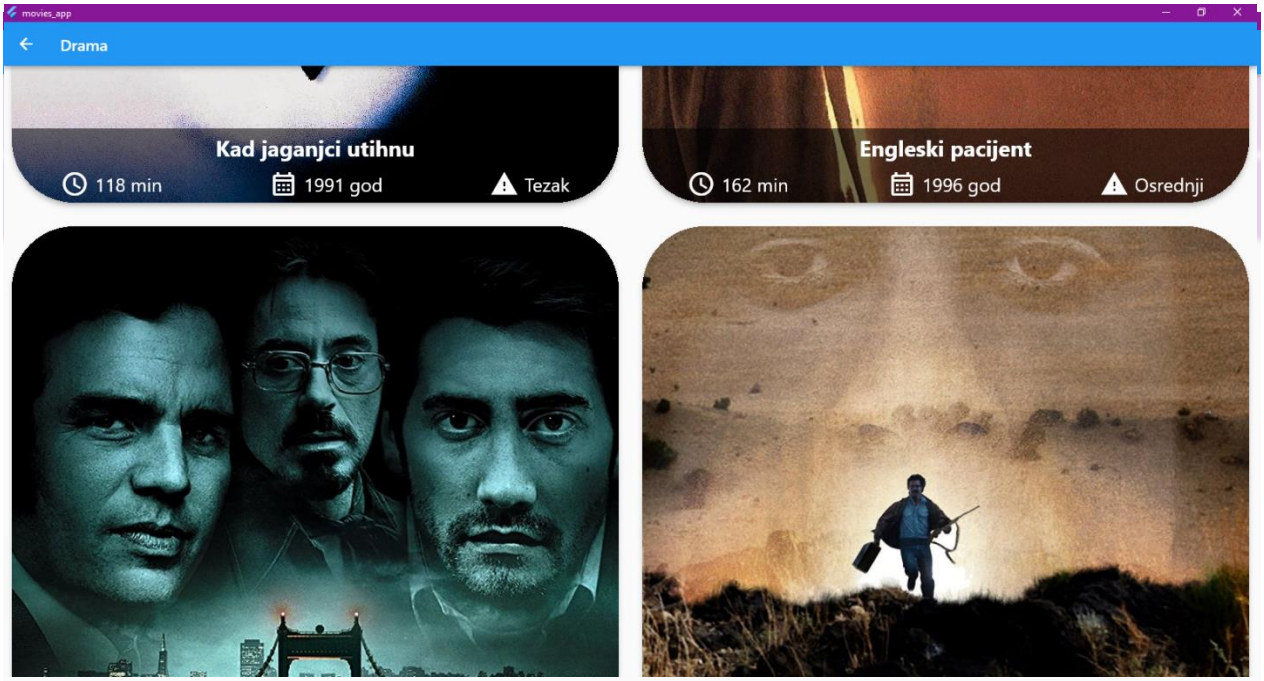
Kod 34: Provider main (vlastoručna izrada, 2023.)

Nakon mnogo rada, vremena i novih widgeta i informacija, ovaj smo kompleksni primjer priveli kraju jednako kao i cjelokupni rad. Pokazali smo još jednom osebujnost i učinkovitost Flutter tehnologije da uz pomoć Dart programskog jezika isporuči kompleksno programsko rješenje, ovoga puta u vidu preglednika filmova prema kategorijama uz pregled detalja o istima, mogućnost favoriziranja i primjene filtera. Krajnji rezultat našega rada:

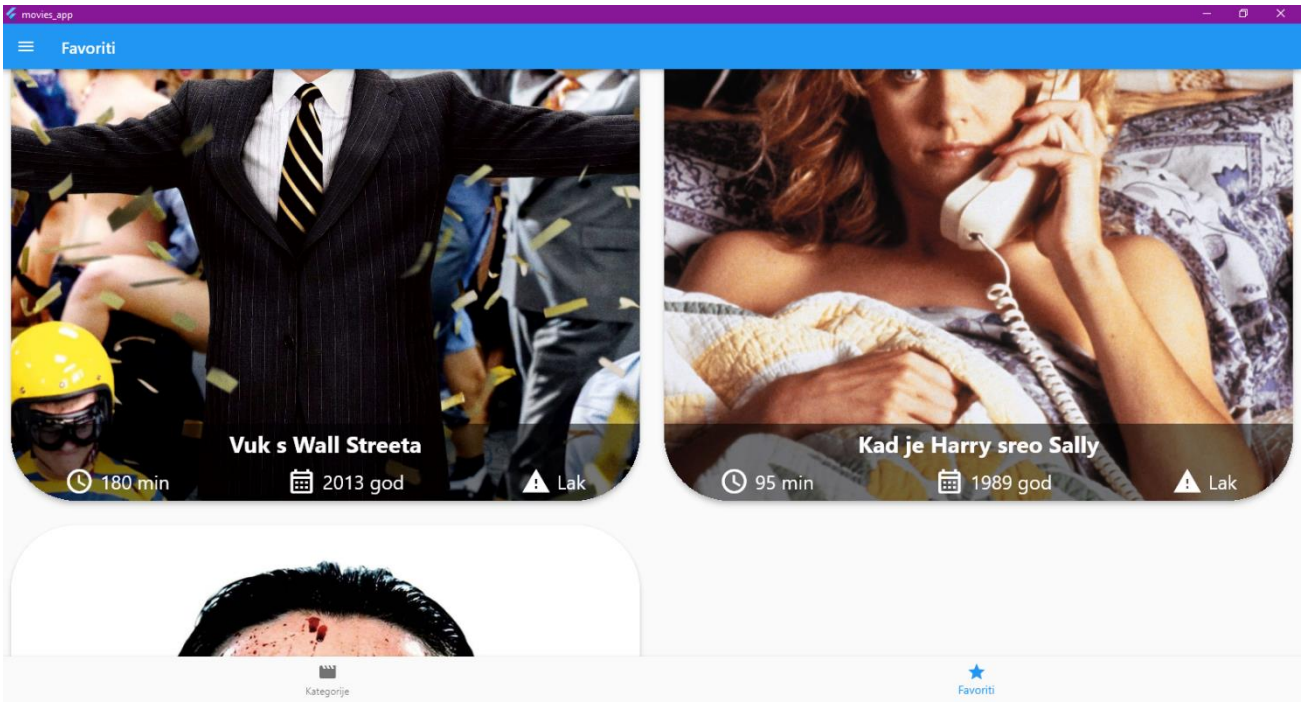


Slika 38:Aplikacija filmova - Ekran kategorija (vlstoručna izrada, 2023.)

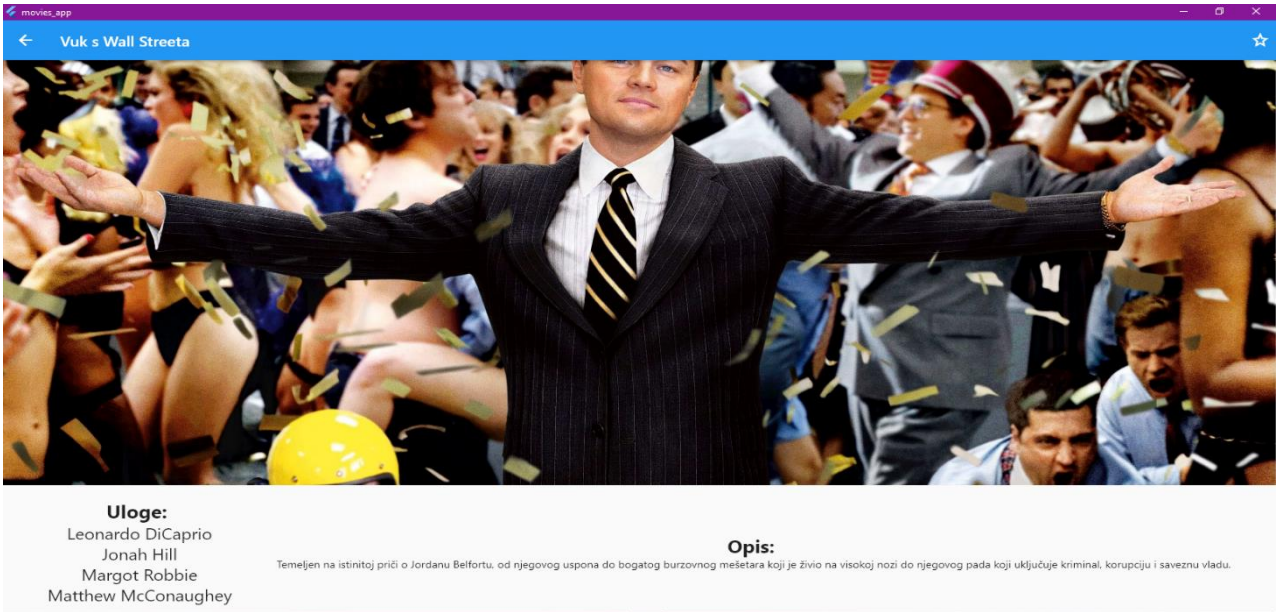
Slika 39:Aplikacija filmova - Bočna ladica (vlastoručna izrada, 2023.)



Slika 42: Aplikacija filmova - Ekran filmova kategorije (vlastoručna izrada, 2023.)



Slika 40: Aplikacija filmova - Ekran favoriziranih filmova (vlastoručna izrada, 2023.)



Slika 43: Aplikacija filmova - Ekran detalja filma (vlastoručna izrada, 2023.)

Logičkim slijedom, razradu problematike ovoga završnog rada priveli smo samome kraju te je jedino preostalo sljedećim poglavljem donesti zaključak.

5. Zaključak

Uzevši u obzir svo istraživanje, ono teorijsko i praktično, zaključujem kako stolnim programskim proizvodima nije došao kraj, a niti se isti nazire, ne njima niti tehnologijama korištenima u njihovoj izradi, a time i poslovima odnosno zaposlenicima koji stoje iza izrade, kako tehnologija tako i samih aplikacija. Znači li to da su stolni programski proizvodi i njihove tehnologije jedino rješenje te nezamjenjivi? Apsolutno ne, no upravo to i jest poenta ovoga rada, a riječ je o tome kako primjerice internetski programski proizvodi služe kao zamjena ili nadomjestak stolnima, a isto vrijedi i obrnuto. Stolni programski proizvodi nikada neće biti iskorijenjeni, niti bi smjeli, njihova izrada, primjena i održavanje mijenja se dakako obzirom na potrebe tržišta.

Internetski ili mobilni programski proizvodi i tehnologije korištene u njihovoj izradi nešto su čime se svatko od nas susreće sve više, no njihovo učestalo korištenje ne opovrgava mnoge od kvaliteta i prednosti koje posjeduju stolni programski proizvodi. Želimo sigurnost? Biramo stolne aplikacije! Želimo li moći aplikaciju pokretati s različitih uređaja brzo i jednostavno? Biramo internetske aplikacije! Zahtjeva li naša aplikacija povezanost s internetom? Vjerojatno je riječ o internetskoj aplikaciji. Zahtjeva li naša aplikacija preuzimanje i instalaciju određenih paketa? Vjerojatno je riječ o stolnoj aplikaciji. Pogledamo li pak tehnologije koje možemo koristiti pri izradi uvidjet ćemo kako je .NET bolji u nekim aspektima od Pythona, dok je Python brži u nekim slučajevima od Fluttera, a Flutter je pak bolji u nekim aspektima od .NET-a. Stoga ovaj primjer nam ukazuje na nužnu i ne slučajnu komplementarnost i međusobnu nadopunu, kako različitih tehnologija tako i različitih okoliša.

Upravo komplementacija jest riječ koja najbolje opisuje trenutne tehnološke trendove u kojima tehnologije za razvoj stolnih programskih proizvoda igraju veoma značajnu ulogu, bilo da je riječ o programskim proizvodima za pametne kuće, automobilske industrije, umjetnu inteligenciju ili pak videoigre. Svi ovi programski proizvodi komplimentiraju ili će komplimentirati naše živote u budućnosti stoga je bitno ostati u koraku s trendovima, a tehnologije za izradu stolnih programskih proizvoda to i čine, jedna od takvih jest Flutter.

Flutter kao tehnologija koja je primarno osvrnuta prema izradi programskih proizvoda za mobilne uređaje, u veljači 2022. godine na tržište je lansirala svoj prvu stabilnu verziju za izradu stolnih programskih proizvoda, ovdje spadaju i Windows i Linux i MacOS. U svijetu punom vizualnih podražaja i brige o izgledu, aplikacije i nas samih, Flutter uz vjernog pomoćnika programskog jezika Darta težište stavlja upravo na mogućnosti uređivanja svakog djelića našeg proizvoda na principu „sve je widget“.

Kroz dva izrađena stolna programska proizvoda u Flutteru opravdali smo naše prethodno uspostavljene teze i prikupljene informacije. Flutter se ispostavio ne samo kao nova tehnologija koja ima mogućnost izrade stolnih programskih proizvoda, već su isti lako prenosivi na ostale spomenute okoliše poput web preglednika i mobilnog uređaja. Osim toga Flutter kao takav veoma dobro konkurira ostalim već ustaljenim tehnologijama te ga developeri odobravaju, što nam zapravo ukazuje na dva zaključka: prostor za stvaranje i korištenje potpuno novih tehnologija vezanih uz stolne programske proizvode, postoji, a samim time i naš developerski svijet pozdravlja takve pothvate te međusobno grade zajednicu u kojoj se može učiti i rasti. Također Flutter se pokazao veoma zabavno iskustvo izrade programskog proizvoda što nikada nije na odmet, stoga bih osobno preporučio svim developerima i onima koji se tako osjećaju da Flutteru daju šansu i izrade jednostavni proizvod za svoju radinost.

Kada se sve zbroji i oduzme, tehnologije za razvoj stolnih programskih proizvoda jednako kao i same proizvode očekuje jedna zaista uzbudljiva budućnost, nimalo jednostavna, prepuna novina u svim relevantnim područjima koje nadopunjuju i ispunjavaju naše živote. Upravo zbog toga smatram kako je ovaj rad ostvario svoju svrhu te opravdao bivstvo stolnih programskih proizvoda kroz krucijalne aspekte definiranja, usporedbe, realnog osvrta na prednosti i mane, pronalaska jedne potpuno nove tehnologije te učenja iste i uspješne eksperimentalne izrade programskih proizvoda te u svojoj objektivnosti i opširnosti nudi djelić povijesti i djelić budućnosti, što doprinosi važnosti ove teme koja je nadam se i Vas potakla na promišljanje. Mislim, dakle jesam (lat. *Cogito ergo sum*)!

Popis literature

Ali, S.H., Ayad, H., Al Rubaie, M. T. (2022). *Fifth Industrial Revolution (New Perspectives). International Journal of Business, Management, and Economics*. Preuzeto 18.6.2022. s http://journal.jis-institute.org/index.php/ijbmer/article/view/694/518

Altexsoft (4.8.2022). *The Good and The Bad of Flutter App Development*. Preuzeto 18.8.2022.s https://www.altexsoft.com/blog/engineering/pros-and-cons-of-flutter-app-development/

American Heritage - Dictionary of the English Language, Fifth Edition (2016). „*Abacus*.“. Preuzeto 14.6.2022. sa https://www.thefreedictionary.com/abacus

Ashwin Joy (2022) 5 main disatvantages of Python Programming Language. Preuzeto 4.8.2022 s https://pythonistaplanet.com/disadvantages-of-python/

Cambridge dictionary (bez dat.). *Web app*. Preuzeto 25.6.2022. s https://dictionary.cambridge.org/dictionary/english/web-app

Ben Wilson (2019). *Technical Erosion and Java Swing*. Preuzeto 23.8.2022. s https://vaadin.com/blog/technical-erosion-and-java-swing

Cebu Web Maker (24.10.2018.). *Do You Know About The Importance Of Desktop Application Development*? Preuzeto 8.7.2022. s https://cebuwebmaker.com/information-technology/do-you-know-about-the-importance-of-desktop-application-development/

Daria Karasek, (2021.). *5 reason Why Rust Is The Future*. [Blog post]. Preuzeto 22.8.2022. s https://scalac.io/blog/5-reasons-why-rust-is-the-future-rust-functional-programming/

Flutter (software) (bez dat.) u Wikipedia. Preuzeto 19.8.2022. s https://en.wikipedia.org/wiki/Flutter_(software)

Hrvatska enciklopedija [EH], *Aplikacija*. Preuzeto 25.6.2022. s https://enciklopedija.hr/natuknica.aspx?id=3306

iFour Techno Lab (23.12.2021.). *Is .NET Framework dead and what is its future?* Preuzeto

28.7.2022. s https://www.ifourtechnolab.com/blog/is-net-framework-dead-and-what-is-its-future-

Jane Marsh (9.2.2021.). *Best Apps for Tracking Home Energy*. Preuzeto 19.7.2022. s https://datafloq.com/read/best-apps-tracking-home-energy-usage/

Jason Fernando (2020). *Assembly Language Definition*. Preuzeto 15.6.2022. s https://www.investopedia.com/terms/a/assembly-language.asp

Jerome Rault (2020). *History of software development.* Preuzeto 15.6.2022. s https://www.laneways.agency/history-of-software-development/

Maciek Berka (2020). *Flutter Pros & Cons – Should YouUse It In Your Project*. Preuzeto 19.8.2022. s https://invotech.co/blog/flutter-pros-cons-should-you-use-it-in-your-project/

Merriam-Webster. (n.d.). Software engineering. In *Merriam-Webster.com dictionary*. Preuzeto 18.8.2022 s https://www.merriam-webster.com/dictionary/software%20engineering

Nadezhda Mal (1.7.2022.). *Web App vs. Desktop App: What Is the Difference*. Preuzeto 5.7.2022. s https://www.qulix.com/about/web-app-vs-desktop-app/

Ravi Saive (2016). *The Truth About Python and Pearl – Features, Pros and Cons Discussed*. Preuzeto (20.8.2022.) s https://www.tecmint.com/the-truth-of-python-and-perl-features-pros-and-cons-discussed/

Regenesys Business School (2020). *The Fifth Industrial Revolution (5IR) and how it will change the business landscape*. Preuzeto 18.6.2022. s https://www.regenesys.net/reginsights/the-fifth-industrial-revolution-5ir/

Sells, Chris (10.2.2022.). *Flutter Update: Windows* [Video file]. Preuzeto 20.8.2022. s https://www.youtube.com/watch?v=g-0B_Vfc9qM&t=604s&ab_channel=Flutter

Shalaka Gadgil (2022). *What are the Top .NET Framework Trends in 2022*? Preuzeto 18.9.2022. s https://www.clariontech.com/blog/top-.net-framework-trends-in-2022

Stack Overflow Insights (2021). *2021 Developer Survey*. Preuzeto 18.9.2022. s

https://insights.stackoverflow.com/survey/2021

Strahonja, V. (2017.). Programsko inženjerstvo – od krize softvera, do znanosti i struke. Programsko inženjerstvo [Moodle]. Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin.

Technobrains (14.2.2022.). *Future of .NET Framework and More on its Recent Updates.* Preuzeto 2.8.2022. s https://technobrains.io/future-of-net-framework/

The Truth of Python and Perl – Features, Pros and Cons Discussed (2016). Preuzeto 22.8.2022. s https://www.tecmint.com/the-truth-of-python-and-perl-features-pros-and-cons-discussed/

Udemy (2022). *C#*. Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=c%23

Udemy (2022). *Python.* Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=python

Udemy (2022). *Java.* Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=java

Udemy (2022). *Rust.* Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=rust

Udemy (2022). *Perl*. Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=perl

Udemy (2022). *Dart* Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=dart

Udemy (2022). *Flutter*. Preuzeto 18.9.2022. s https://www.udemy.com/courses/search/?src=ukw&q=flutter

Upendra Jith. (2021). *Is There a Future for Desktop Development*. [Blog post]. Preuzeto 22.8.2022. s https://www.bridge-global.com/blog/future-of-desktop-development/

Popis slika

Popis crteža

Popis programskih kodova

Popis tablica

Prilozi

U nastavku su priložene poveznice na moj GitHub repozitorij u kojemu se nalaze dva navedena praktična primjera ovoga rada:

Glavni repozitorij:

https://github.com/VidTomljenovic/Flutter

Aplikacija evidencije čitanja:

https://github.com/VidTomljenovic/Flutter/tree/EvidencijaCitanja

izvršna aplikacija nalazi se unutar mape Build -> Windows -> runner -> Release -> ime_aplikacije.exe

Aplikacija pregleda filmova:

https://github.com/VidTomljenovic/Flutter/tree/PregledFilmova

izvršna aplikacija nalazi se unutar mape Build -> Windows -> runner -> Release -> ime_aplikacije.exe