

SE201 – Uvod u softversko inženjerstvo

Softver za generaciju i skladištenje putnih naloga

Student: Petar Stošić 5466 Profesor: Jovana Jović Asistent: Bogdan Janković

Niš, 2025

# Sadržaj

[Sadržaj 2](#_Toc210103646)

[Uvod 5](#_Toc210103647)

[Izbor agilne metodologije 6](#_Toc210103648)

[Murov obrazac za viziju proizvoda 7](#_Toc210103649)

[Funkcionalni zahtevi 8](#_Toc210103656)

[Nefunkcionalni zahtevi 9](#_Toc210103657)

[Korisničke priče 10](#_Toc210103658)

[Persone 11](#_Toc210103659)

[Scenariji za persone 12](#_Toc210103665)

[Stilovi softverske arhitekture 14](#_Toc210103666)

[Klase 16](#_Toc210103667)

[Baza podataka 18](#_Toc210103668)

[Ponovno upotrebljive komponente 19](#_Toc210103669)

[Računarstvo u oblaku 21](#_Toc210103670)

[Ranjivosti 23](#_Toc210103688)

[Atributi kvaliteta sistema 24](#_Toc210103689)

[Pouzdano programiranje 25](#_Toc210103690)

[Scenariji testiranja 26](#_Toc210103691)

[xUnit testiranje 27](#_Toc210103692)

[Trello alat 28](#_Toc210103694)

[Git 29](#_Toc210103695)

[Zaključak 30](#_Toc210103696)

[Literatura 31](#_Toc210103697)

# Uvod

Ovaj softver predstavlja jednostavnu, ali praktičnu aplikaciju za upravljanje putnim nalozima u okviru firme ili organizacije. Ideja je da se izbegne ručno popunjavanje obrazaca i komplikacije oko štampanja i arhiviranja, tako što se ceo proces prebaci u digitalni format sa automatskim čuvanjem podataka i eksportom u PDF. Korisnik može da unese osnovne podatke o putovanju, zaposleni, vozilu i troškovima, a sistem zatim generiše čitav dokument spreman za štampu ili digitalnu arhivu.

Ciljna grupa su firme i službe koje često izdaju putne naloge i žele jednostavno rešenje koje ne zahteva nikakvu dodatnu infrastrukturu, servere ili mrežne baze. Aplikacija se pokreće lokalno i sve podatke čuva na korisničkoj mašini.

Glavni korisnički tok se svodi na nekoliko koraka: prvo se u sistem unesu informacije o zaposlenima i vozilima, zatim se kreira novi putni nalog kroz intuitivnu formu, i na kraju se taj nalog može sačuvati i izvesti u obliku preglednog PDF dokumenta. Aplikacija takođe nudi pregled prethodnih naloga i jednostavno filtriranje po datumu putovanja.

Za razvoj aplikacije korišćen je C# uz Windows Forms za korisnički interfejs. Kao baza podataka koristi se LiteDB, što je fajl-bazirana NoSQL baza idealna za ovakve desktop aplikacije, jer omogućava čuvanje podataka bez potrebe za dodatnim serverima ili kompleksnim podešavanjima. Podaci se čuvaju lokalno u .db fajlu, a pristupa im se kroz custom helper klasu LiteDatabaseManager.

Za generisanje PDF dokumenata koristi se QuestPDF, moderna biblioteka za .NET aplikacije koja omogućava vrlo precizno i fleksibilno kreiranje dokumenata. Stilizacija, raspored, margine, fontovi i sve ostalo se definiše kroz kod, što omogućava punu kontrolu nad izgledom finalnog dokumenta. Ova biblioteka omogućava da se kompletan sadržaj dinamički sastavi na osnovu unetih podataka, bez šablona i manuelnog formatiranja.

Aplikacija je modularno podeljena u nekoliko važnih klasa: Employee, Vehicle, TravelOrder, TravelCosts, DataManager, kao i forme AddForm, TravelOrderForm, TrackForm i PdfPreviewForm koje omogućavaju rad sa podacima i vizualizaciju rezultata. Svaki deo ima jasnu ulogu i zaduženje, što olakšava održavanje i budući razvoj sistema.

# Izbor agilne metodologije

Za razvoj desktop aplikacije koja omogućava evidenciju i štampanje putnih naloga, metodološki pristup koji je izabran bio je Kanban. Ova metodologija je najviše odgovarala načinu rada jer se projekat razvijao postepeno, bez striktnih vremenskih ograničenja i bez potrebe za preciznim sprint planiranjem. Nije bilo fiksnog roka, jer se radilo u pauzama između drugih akademskih i poslovnih obaveza.

Primena Kanban metodologije se ogledala u organizaciji rada kroz virtuelnu tablu podeljenu na To Do, In Progress i Done. Sve ideje, zadaci i problemi beleženi su kao kartice koje su se pomerale po potrebi. Na početku razvoja tabla je sadržavala osnovne zadatke vezane za kreiranje formi, povezivanje sa bazom i osnovni unos podataka. Tokom rada, zadaci su se nadopunjavali u skladu sa potrebama koje su iskrsavale, npr. dodavanje eksportovanja u PDF, implementacija dropdown izbora za vrstu vozila, logika za automatski obračun troškova itd.

Kanban je omogućio da se radi potpuno asinhrono – bez sastanaka, rola, vremenskih sprintova ili ceremonija. Kada bi se pojavio novi zahtev ili ideja, jednostavno bi se dodala nova kartica i započela implementacija čim za to postoji vreme. Ovo je bilo naročito korisno jer su faze razvoja često bile nelinearne – npr. testiranje generisanja PDF izveštaja se dešavalo paralelno sa refaktorisanjima u bazi podataka ili sa dorađivanjem korisničkog interfejsa.

Testiranje i korekcije su takođe bile integrisane u sam tok. Kada bi neki zadatak prešao u "Done", nije značilo da je završeno zauvek – u nekim slučajevima bi se vraćao u "In Progress" zbog otkrivene greške ili potrebe za dodatnim podešavanjima. Tako se prirodno i fluidno iteriralo bez stvaranja prevelikog tehničkog duga.

# Murov obrazac za viziju proizvoda

## Misija Cilj ovog softverskog rešenja je da pojednostavi i ubrza proces unosa, evidencije i generisanja putnih naloga u organizacijama. Umesto ručnog ispunjavanja obrazaca i Excel tabelisanja, korisnici mogu kroz intuitivan interfejs brzo da popune nalog, izračunaju troškove i izvezu gotov PDF dokument spreman za štampu ili slanje. Time se štedi vreme, smanjuju greške i čuva istorija svih naloga na jednom mestu.

## Vrednosti Kroz projekat se stavlja akcenat na jednostavnost korišćenja, automatizaciju dosadnih zadataka i sigurnost podataka. Aplikacija ne traži nikakvu internet konekciju jer radi lokalno, pa su svi unosi i dokumenti sačuvani na računaru korisnika. Interfejs je pregledan i prilagođen stvarnim potrebama administracije i računovodstva.

## Resursi Projekat je razvijen u C# programskom jeziku uz korišćenje .NET Windows Forms tehnologije. Za rad sa podacima koristi se lokalna baza LiteDB, a za generisanje PDF dokumenata koristi se iTextSharp biblioteka. Radni resursi uključuju IDE Visual Studio, Windows operativni sistem i dokumentaciju sa zvaničnih sajtova korišćenih biblioteka. Razvojni tim je činila jedna osoba, što je dodatno motivisalo korišćenje jednostavne arhitekture i modularnog pristupa.

## Interesne grupe

## Glavne interesne grupe ovog projekta su zaposleni koji putuju u ime firme i administracija koja obrađuje njihove naloge. Zaposleni imaju potrebu da brzo popune podatke o putu bez gubljenja vremena na komplikovane formulare, dok administracija mora da ima jasan i konzistentan izveštaj za arhivu i obračun troškova. Pored njih, i finansijski sektor firme ima koristi jer dobija automatski obračun troškova na osnovu kilometraže i tipa vozila. Posmatrano šire, sistem može koristiti i menadžmentu kao podrška za analizu učestalosti i cene poslovnih putovanja u okviru organizacije.

## Potrebni resursi

1. Tehnički resursi: Windows operativni sistem, .NET Framework, C# okruženje (npr. Visual Studio), LiteDB za lokalnu bazu podataka, i iText za generisanje PDF dokumenata
2. Ljudski resursi: Programer koji razvija i održava aplikaciju, eventualno osoba koja piše dokumentaciju i testira sve funkcionalnosti
3. Vremenski resursi: Fleksibilno vreme za razvoj i testiranje, uz iterativno dodavanje novih funkcija na osnovu povratnih informacija
4. Ostalo: Dokumentacija zakonskih propisa koji regulišu izgled i sadržaj putnih naloga u Srbiji, kao i primeri realnih naloga za testiranje i validaciju formata

# Funkcionalni zahtevi

Funkcionalni zahtevi predstavljaju konkretne zadatke i mogućnosti koje softver mora da omogući korisniku. Oni opisuju šta sistem treba da radi u različitim situacijama i interakcijama. U kontekstu aplikacije za evidenciju putnih naloga, to su svi koraci koje korisnik može da preduzme u vezi sa unosom, čuvanjem i obradom naloga.

Na osnovu funkcionalnosti implementiranih u ovom projektu, ovo su funkcionalni zahtevi:

1. **Unos i uređivanje putnih naloga**  
   Korisnik mora imati mogućnost da kreira novi putni nalog, kao i da naknadno izmeni podatke za već kreirani nalog (npr. destinaciju, vozilo, svrhu puta, broj dana).
2. **Dodavanje i čuvanje zaposlenih i vozila**  
   Sistem omogućava dodavanje novih zaposlenih i službenih vozila u lokalnu bazu, kao i njihovo čuvanje za kasniju upotrebu u više naloga.
3. **Izračunavanje ukupnih troškova puta**  
   Na osnovu unetih podataka, aplikacija automatski izračunava dnevnice, troškove goriva i ukupne troškove puta, u skladu sa vrstom vozila i pređenom kilometražom.
4. **Generisanje putnog naloga u PDF formatu**  
   Sistem omogućava izvoz ispunjenog putnog naloga kao .pdf fajl, koji je formatiran po standardnom obrascu i spreman za štampu ili arhivu.
5. **Pregled i pretraga postojećih naloga**  
   Korisnik može da pregleda listu svih prethodno kreiranih putnih naloga, da ih filtrira po datumu ili zaposlenom i da otvori nalog radi pregleda ili ponovnog izvoza.
6. **Praćenje statusa i istorije naloga**  
   Svaki nalog čuva svoj status i datum kreiranja, tako da sistem omogućava praćenje istorije unosa i vraćanje na prethodne naloge kad god je to potrebno.
7. **Automatska popuna podataka na osnovu prethodnih unosa**  
   Kada korisnik unese deo podataka (npr. ime zaposlenog ili registarsku oznaku), sistem automatski nudi prethodno sačuvane unose radi bržeg popunjavanja forme i smanjenja grešaka.
8. **Validacija unetih podataka pre čuvanja**  
   Sistem proverava da li su svi obavezni podaci popunjeni (npr. destinacija, zaposleni, vozilo, datumi), i sprečava čuvanje naloga ukoliko nešto nedostaje ili je pogrešno formatirano.
9. **Podesiva vrednost dnevnice i cena goriva**  
   Aplikacija omogućava korisniku da ručno unese ili izmeni vrednosti dnevnica i cene goriva, u slučaju promene propisa ili internih pravila firme.
10. **Čuvanje PDF fajla na željenu lokaciju**  
    Korisnik može da odabere gde želi da sačuva generisani PDF dokument na svom računaru, ili da ga direktno otvori u pregledniku radi štampe bez dodatnog traženja.

# Nefunkcionalni zahtevi

Nefunkcionalni zahtevi opisuju osobine sistema koje nisu vezane direktno za ponašanje funkcija, već za to kako sistem funkcioniše. To su zahtevi koji se odnose na performanse, stabilnost, bezbednost, skalabilnost i slične osobine koje utiču na kvalitet korisničkog iskustva i održavanje softvera.

Za ovu aplikaciju, sledeći nefunkcionalni zahtevi su izdvojeni kao najbitniji:

1. **Performanse sistema**  
   Aplikacija mora da reaguje na korisničke akcije (kao što su unos podataka ili generisanje PDF izveštaja) u roku kraćem od jedne sekunde na prosečnom računaru (npr. Intel i5, 8GB RAM).
2. **Pouzdanost i stabilnost**  
   Sistem mora da bude stabilan u radu i da ne dođe do gubitka podataka prilikom nepredviđenog gašenja aplikacije. Svi podaci se trajno čuvaju u lokalnoj bazi.
3. **Bezbednost korisničkih podataka**  
   Pristup bazi podataka i generisanim dokumentima mora biti moguć samo kroz samu aplikaciju. Direktan pristup fajlovima i bazama sa strane korisnika mora biti onemogućen kroz UI aplikacije.
4. **Prenosivost i jednostavna instalacija**  
   Aplikacija mora da bude funkcionalna na svim Windows mašinama koje imaju instaliran .NET Framework. Instalacija treba da bude jednostavna, bez potrebe za dodatnim konfiguracijama ili online servisima.
5. **Skalabilnost baze podataka**  
   Baza mora da podrži čuvanje bar 1000 zapisa bez pada performansi, kako bi aplikacija mogla da se koristi duže vreme bez potrebe za migracijom podataka.

# Korisničke priče

Korisničke priče su jednostavan, ali moćan način da se razumeju potrebe korisnika bez komplikovanih tehničkih specifikacija. Umesto da se pišu dugački dokumenti sa suvim opisima funkcija, korisnička priča stavlja stvarnog korisnika u centar pažnje i kroz kratak format objašnjava šta ta osoba želi da postigne, zašto joj je to važno i šta očekuje od sistema.

U ovom projektu, korisničke priče pomažu da se konkretizuju scenariji iz svakodnevnog poslovanja, kao što su kreiranje putnog naloga, unos zaposlenih, izračunavanje troškova ili generisanje PDF‑a. Takođe, omogućavaju timu da prioritizuje najvažnije funkcionalnosti, da testira aplikaciju na realnim primerima i da zna da li su korisnici zadovoljni načinom na koji softver rešava njihove probleme.

Bez korisničkih priča, razvoj bi se oslanjao isključivo na tehničke specifikacije, što često dovodi do toga da se napravi sistem koji tehnički radi, ali korisnicima zapravo nije koristan ni intuitivan. Zbog toga se korisničke priče pišu već u ranoj fazi projekta i stalno se osvežavaju kako bi softver pratio stvarne potrebe ljudi koji će ga koristiti.

Za ovu aplikaiju, imamo sledeće korisničke priče:

1. Kao administrator firme,

želim da mogu da unesem nove zaposlene sa osnovnim podacima (ime, prezime, pozicija, JMBG, adresa),

kako bih mogao da ih dodelim putnim nalozima bez potrebe da unosim te podatke svaki put iznova.

1. Kao sekretarica firme,

želim da brzo i jednostavno mogu da napravim novi putni nalog,

kako bih smanjila vreme koje trošim na papirologiju i obezbedila ispravnu dokumentaciju.

1. Kao korisnik aplikacije,

želim da mogu da unesem detalje o putovanju (datum polaska, povratka, relaciju, broj dana),

da bih precizno zabeležio putne naloge za zaposlene.

1. Kao vozač u firmi,

želim da moja vozila budu registrovana u sistemu sa svim podacima (tip, registracija, potrošnja, gorivo),

kako bi se automatski preračunavali troškovi i generisali ispravni dokumenti.

1. Kao nadređeni,

želim da mogu da pregledam već kreirane putne naloge,

da bih proverio gde su zaposleni išli, koliko su bili odsutni i koliki su bili troškovi.

1. Kao korisnik,

želim da aplikacija može automatski da izračuna troškove puta (gorivo, dnevnica, putarine),

da ne bih morao sam da pravim računicu.

1. Kao korisnik,

želim da iz aplikacije mogu da generišem pregledan PDF putnog naloga,

kako bih ga mogao lako poslati na mejl ili odštampati za arhivu.

1. Kao administrator baze,  
   želim da mogu da ažuriram ili obrišem podatke o zaposlenima i vozilima,  
   kako bih održavao tačnost baze i izbegao duplikate ili zastarele informacije.
2. Kao korisnik koji često koristi aplikaciju*,*  
   želim da mogu da pretražujem stare putne naloge po imenu zaposlenog, vozilu ili datumu,  
   da bih lako pronašao konkretan nalog bez ručnog listanja cele baze.
3. Kao finansijski referent,  
   želim da imam uvid u ukupne mesečne ili godišnje troškove putovanja po zaposlenom,  
   kako bih mogao da napravim izveštaje za internu evidenciju ili reviziju.

# Persone

Personae predstavljaju izmišljene, ali realistične korisnike sistema. Kreiraju se na osnovu stvarnih potreba i ponašanja krajnjih korisnika. Uloga persone nije da popuni dokumentaciju, već da pomogne timu da stalno razmišlja kako korisnici zapravo koriste aplikaciju, umesto da se donose odluke iz ugla developera ili profesora.

Kada u razvoju softvera postoje jasno definisane persone, mnogo je lakše proceniti koje funkcije su stvarno korisne, koji elementi interfejsa treba da budu jednostavni, i koje automatizacije zaista štede vreme. Tako se ceo tim usmerava ka istom cilju - pravljenju alata koji rešava realne probleme konkretnih ljudi.

## Persona 1: Jelena Marković **Godine:** 29 **Zanimanje:** Sekretarica u državnoj firmi **Cilj:** Želi da brzo i jednostavno unese podatke o putnim nalozima za svoje kolege bez da se gubi u papirologiji. **Opis:** Jelena je tehnički pismena, koristi računar svakodnevno i ume da radi u Excelu. Frustrira je kada mora da ispunjava ručno obrasce i računa troškove. Očekuje da softver bude jednostavan, pregledan i da izveze gotov PDF bez dodatnog obrađivanja.

## Persona 2: Marko Stojić **Godine:** 42 **Zanimanje:** Komercijalista koji često putuje po Srbiji **Cilj:** Želi da ima aplikaciju koja beleži sve njegove prethodne naloge, troškove i da može da dokaže gde je i kada bio. **Opis:** Marko ne voli da vodi beleške po papiru i često zaboravi da preda nalog na vreme. Voleo bi da može da popuni putni nalog na terenu i da mu se sve automatski izračuna i sačuva. Nije stručnjak za računare, ali koristi mobilne aplikacije svakodnevno.

## Persona 3: Marija Ristić **Godine:** 35 **Zanimanje:** Finansijska službenica zadužena za obračun troškova putovanja **Cilj:** Potrebna joj je aplikacija koja generiše precizan izveštaj za svakog zaposlenog, sa svim pripadajućim troškovima i podacima spremnim za arhivu ili štampu. **Opis:** Marija ima dosta iskustva u radu sa finansijskim softverima. Važno joj je da softver ne pravi greške i da bude u skladu sa zakonskim propisima. Mora imati uvid u sve prethodne naloge i opciju pretrage po datumu, zaposlenom ili vozilu.

Persona 4: Dragan Ilić  
**Godine:** 58  
**Zanimanje:** Direktor male firme  
**Cilj:** Želi brz uvid u sve službene puteve zaposlenih i minimalno vreme provedeno u papirologiji.  
**Opis:** Dragan nije tehnički potkovan i koristi računar samo za osnovne funkcije. Očekuje da aplikacija „radi sama“, bez komplikovanih koraka. Ne želi da se bavi unosom podataka, već da klikne i vidi izveštaj.

Persona 5: Ivana Milenković  
**Godine:** 26  
**Zanimanje:** Junior developer koji održava aplikaciju u firmi  
**Cilj:** Želi da softver bude stabilan, lako nadogradiv i da korisnici ne zovu zbog bagova.  
**Opis:** Ivana je u firmi zadužena za tehničku podršku softveru. Ne koristi ga svakodnevno kao ostali, ali mora da zna kako funkcioniše svaka opcija. Vodi računa o kompatibilnosti sa operativnim sistemima i bazom.

# Scenariji za persone

**Jelena Marković:**

1. Unosi novi putni nalog za kolegu koji ide u Novi Sad. Bira zaposlenog sa liste, odabira vozilo i datume, unosi relaciju i svrhu puta. Sistem automatski računa troškove, Jelena proverava sve podatke i klikne “Generiši PDF” da dobije gotov dokument spreman za štampu.
2. Tokom provere primećuje da je pogrešno unela broj dana putovanja. Otvara prethodno sačuvani nalog, menja broj dana sa 2 na 3 i sistem odmah ažurira troškove i novi PDF. Jelena ponovo eksportuje ažurirani dokument.
3. Dodaje novog zaposlenog u sistem jer je kolega tek počeo da radi. Otvara formu “Dodaj zaposlenog”, unosi ime, prezime, JMBG i adresu i čuva. Novi zaposleni se automatski pojavljuje u listi za kreiranje putnog naloga.

**Marija Ristić:**

1. Pravi izveštaj za internu kontrolu. U aplikaciji unosi ime zaposlenog i period meseca, dobija spisak svih njegovih putnih naloga i jednim klikom eksportuje PDF-ove za inspekciju.
2. Uoči grešku u kilometraži kod jednog naloga dok pregledava troškove. Otvara nalog, upoređuje podatke i vraća ga sekretarici uz napomenu za ispravku.
3. Na kraju godine preuzima sve putne naloge iz baze, eksportuje ih u PDF i pravi ZIP arhivu spremnu za štampanje i dugoročno čuvanje.

**Dragan Ilić:**

1. Jednom mesečno ulazi u aplikaciju i koristi izveštaje da pogleda ukupne troškove po relacijama i zaposlenima. Ne unosi podatke već samo pregledava, jer aplikacija sve obračunava automatski.
2. Traži od sekretarice da mu pripremi listu naloga sortiranu po ceni, kako bi video koja putovanja su najskuplja i gde se troši najviše sredstava.
3. Otvara aplikaciju i iz nje ispisuje nekoliko PDF naloga koje treba da potpiše, bez unošenja ili menjanja bilo kakvih podataka.

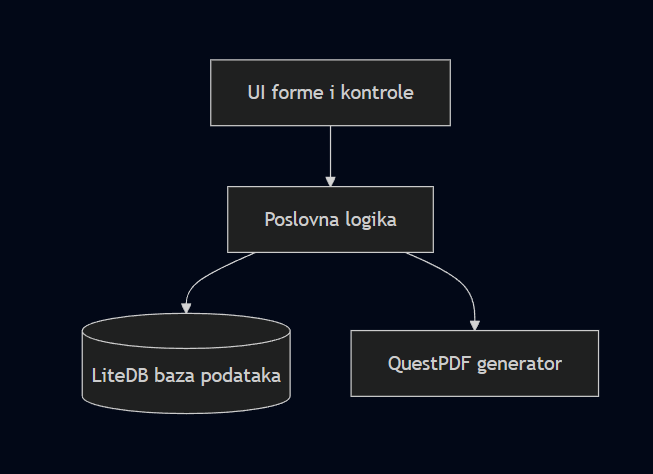
**Ivana Milenković:**

1. Testira novu verziju softvera nakon dodavanja novih funkcija. Unosi testne podatke, proverava da li se pravilno čuvaju u LiteDB i da li se pojavljuju u listi naloga bez duplikata.
2. Dobija prijavu od korisnika o grešci pri eksportovanju PDF-a. Otvara logove aplikacije, pronalazi da je uzrok nedozvoljen karakter u unetim podacima i rešava problem.
3. Po nalogu firme dodaje novu funkcionalnost – automatski obračun putarine. Implementira je u kod, pokreće aplikaciju, testira izračunavanje i proverava da li se vrednost prikazuje na pravom mestu u GUI i PDF-u.

# Stilovi softverske arhitekture

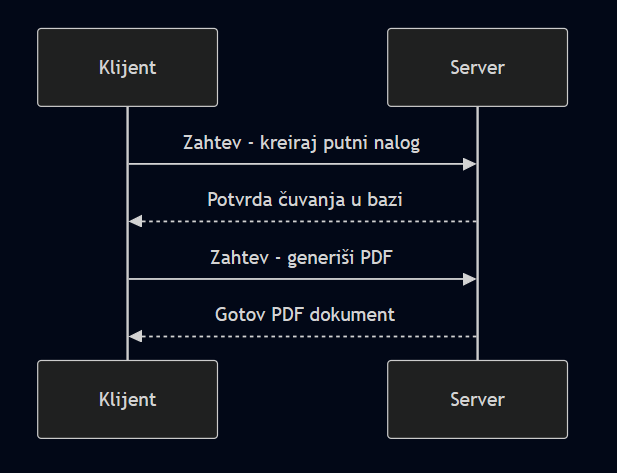
Prilikom razvoja bilo kog softverskog rešenja, arhitektura aplikacije igra ključnu ulogu u njenoj funkcionalnosti, održavanju i skalabilnosti. Iako se projekat u ovom slučaju odnosi na relativno jednostavnu desktop aplikaciju, važno je razumeti i razmotriti različite pristupe softverskoj arhitekturi, kako bi se napravili informisani izbori, ali i ostavila mogućnost za buduću nadogradnju sistema.

Monolitna arhitektura podrazumeva da je cela aplikacija implementirana kao jedna celina. Svi moduli – od korisničkog interfejsa, preko poslovne logike, do pristupa bazi podataka – nalaze se u istom projektu i rade u okviru istog procesa. Ovakav pristup je bio idealan za potrebe ovog projekta, jer se radi o jednostavnoj desktop aplikaciji koja nije distribuirana i ne zahteva rad na više instanci. Prednosti monolita su što omogućava brži razvoj, lakšu integraciju funkcionalnosti i jednostavniji deployment. Sve promene se dešavaju u okviru jednog koda, što dodatno pojednostavljuje testiranje i debugging. Međutim, kako aplikacija raste, monolit može postati teško održiv. Promena jednog dela sistema može nenamerno uticati na druge delove, a veći broj funkcionalnosti otežava čitljivost i modularnost.



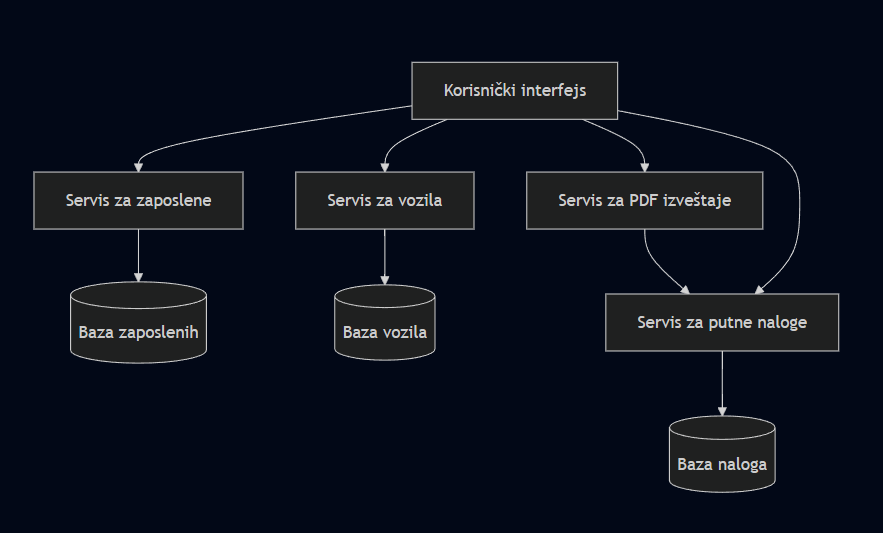
Slika 1 – Dijagram monolita

Sa druge strane, klijent-server arhitektura uvodi jasnu separaciju između klijentskog dela (frontend) i serverskog dela (backend). Klijent šalje zahteve serveru koji obrađuje logiku i vraća odgovore. Ovaj model je posebno koristan u aplikacijama koje se koriste preko mreže, kada više korisnika pristupa centralizovanom serveru. Za sistem putnih naloga, klijent-server model bi bio od koristi ako bi se aplikacija razvila u web platformu kojoj pristupaju različiti zaposleni iz više kancelarija. Prednosti ovog modela su u centralizaciji podataka, lakšem skaliranju i mogućnosti da se frontend i backend razvijaju nezavisno. Međutim, za trenutne potrebe koje ne uključuju mrežnu komunikaciju, ova arhitektura bi bila nepotrebno komplikovana.



Slika 2 – Dijagram klijent-server arhitekture

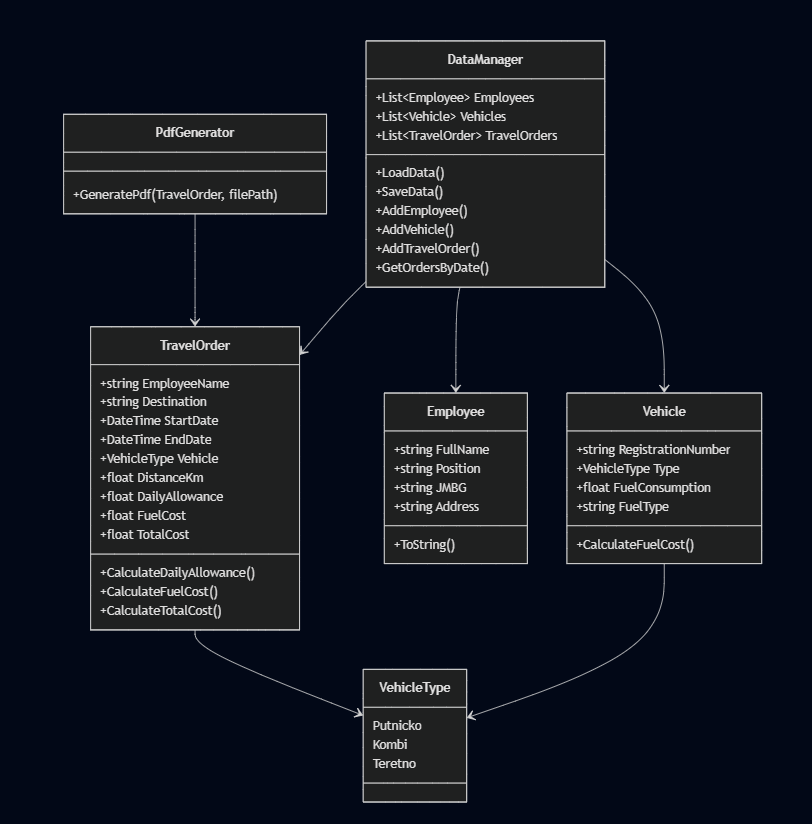
Mikroservisna arhitektura, iako nije primenjena u ovom projektu, razmatrana je kao teoretski okvir za moguće buduće nadogradnje. Ovaj stil podrazumeva razbijanje aplikacije na niz malih, nezavisnih servisa, gde svaki servis obavlja tačno definisanu funkciju i komunicira sa drugim servisima preko API-ja. Mikroservisi omogućavaju veću fleksibilnost, bolje skaliranje i nezavisnost u razvoju različitih delova sistema. Međutim, kompleksnost takvog sistema, potreba za dodatnom infrastrukturom i komplikovana komunikacija između servisa čine ga neprikladnim za male i srednje projekte poput ovog.



Slika 3 – Dijagram mikroservisne arhitekture

# Klase

Klase predstavljaju logičke delove aplikacije – svaka ima svoju ulogu, podatke koje čuva i metode koje izvršava. Na ovaj način sistem je organizovan modularno, pa se svaka funkcionalnost (poput unosa podataka, obračuna troškova ili generisanja PDF dokumenata) nalazi u odgovarajućoj klasi. Ovo olakšava održavanje i razvoj aplikacije, jer je jasno gde se koja logika nalazi i kako su klase međusobno povezane.



Slika 4 – Klasni dijagram aplikacije

**TravelOrder – Klasa za putni nalog**

Klasa TravelOrder predstavlja osnovni entitet sistema – konkretan putni nalog koji korisnik kreira. Svaki nalog sadrži podatke o zaposlenom koji ide na put, destinaciji, datumima polaska i povratka, vrsti i karakteristikama vozila, kao i svim finansijskim elementima vezanim za put. Unutar klase se nalaze metode koje automatski računaju troškove, poput dnevnice i potrošnje goriva, čime se korisnicima olakšava rad i eliminiše potreba za ručnim izračunavanjem. TravelOrder je centralna tačka svih ostalih klasa – povezana je i sa zaposlenima i sa vozilima, a koristi se i prilikom generisanja PDF izveštaja.

**Employee – Klasa za zaposlenog**

Employee klasa služi za predstavljanje zaposlenih osoba koje se dodeljuju putnim nalozima. Svaki zaposleni ima ime, prezime, poziciju, adresu i jedinstveni identifikator u vidu JMBG-a. Ideja je da se jednom uneti zaposleni čuvaju u sistemu i kasnije jednostavno biraju iz liste prilikom kreiranja novog putnog naloga. Time se izbegava višestruko unošenje istih podataka i ubrzava sam proces. Klasa je jednostavna, ali ključna za personalizaciju i automatizaciju procesa.

**Vehicle – Klasa za vozilo**

Klasa Vehicle predstavlja podatke o službenim vozilima firme koja se koriste prilikom realizacije službenih putovanja. Svako vozilo ima registraciju, tip (putničko, kombi, teretno...), potrošnju goriva na 100 kilometara i vrstu goriva. Ovi podaci su značajni zato što na osnovu njih sistem računa troškove putovanja, a posebno trošak goriva. Metoda CalculateFuelCost() omogućava da se automatski preračuna trošak za zadatu relaciju, što olakšava rad korisnicima i smanjuje mogućnost greške.

**DataManager – Upravljanje podacima**

DataManager je pomoćna, ali izuzetno važna klasa koja centralizuje rad sa podacima. Ona čuva liste svih zaposlenih, svih vozila i svih postojećih putnih naloga. Ova klasa omogućava učitavanje i čuvanje podataka na lokalni disk, kao i osnovne CRUD operacije nad svakom vrstom entiteta. Na ovaj način se obezbeđuje postojanost podataka i omogućava korisniku da nastavi rad tamo gde je prethodno stao. Takođe, omogućava filtriranje naloga po datumu i drugim parametrima, što dodatno poboljšava korisničko iskustvo.

**PdfGenerator – Generisanje PDF dokumenata**

Ova klasa je odgovorna za pretvaranje podataka iz TravelOrder objekata u formatiran PDF fajl, koji je spreman za štampu, potpisivanje ili slanje putem mejla. PdfGenerator koristi podatke iz putnog naloga i na osnovu unapred definisanog šablona formira dokument koji je usklađen sa zakonskim standardima. Time se korisnicima štedi vreme, jer ne moraju ručno da prepravljaju ili formatiraju dokument, već dobijaju gotov fajl jednim klikom. Klasa je implementacijski izolovana i koristi se samo kada se eksplicitno zatraži generisanje.

**VehicleType – Enumeracija tipova vozila**

VehicleType nije klasična klasa, već enumeracija koja definiše moguće vrednosti za tip vozila. Njena svrha je da standardizuje unos i obradu podataka o vrstama vozila, čime se izbegavaju greške koje bi nastale pri ručnom unosu teksta. Na primer, korisnik bira „Putničko“ iz padajuće liste umesto da ukucava tekst, što sprečava nevalidne unose i olakšava kasniju obradu.

# Baza podataka

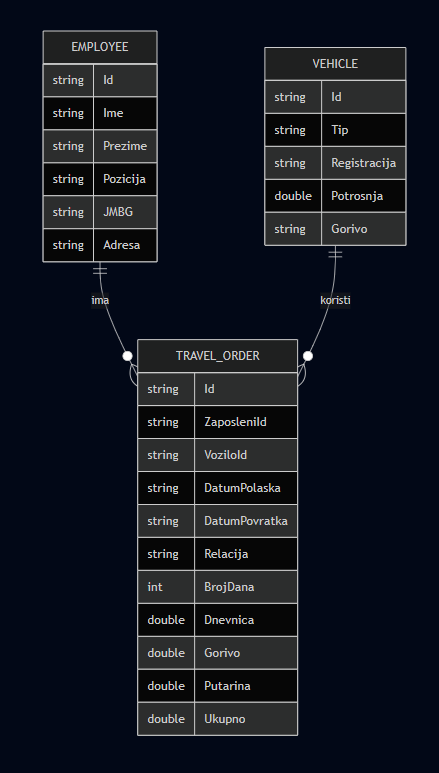
Za čuvanje podataka u ovoj aplikaciji koristi se **LiteDB**, lagana NoSQL baza podataka zasnovana na fajlu, idealna za desktop aplikacije. LiteDB omogućava da se svi podaci smeste u jedan .db fajl bez potrebe za instalacijom posebnog servera, što je idealno rešenje za aplikacije koje se koriste lokalno, bez mrežne povezanosti.

Podaci koji se čuvaju u bazi uključuju informacije o zaposlenima, vozilima, putnim nalozima i troškovima putovanja. Svaka od ovih informacija se smešta u posebnu kolekciju u okviru baze, što omogućava jednostavno i organizovano upravljanje podacima. Podaci se čuvaju kao **dokumenta u BSON formatu**, koji je sličan JSON-u, i lako se mapiraju na C# klase.

U implementaciji, klasa LiteDatabaseManager predstavlja most između ostatka aplikacije i same baze podataka. Ona sadrži metode za dodavanje, čitanje, ažuriranje i brisanje podataka, čime se odvaja logika pristupa bazi od ostatka koda, što je dobra praksa u softverskom inženjerstvu (princip **separacije odgovornosti**)**.**

Naravno, ovakav sistem ima i svoja ograničenja – LiteDB nije predviđen za rad u više korisnika istovremeno preko mreže, niti za vrlo velike količine podataka, ali za potrebe ovog projekta (lokalna desktop aplikacija za evidenciju putnih naloga) predstavlja optimalan balans između jednostavnosti i funkcionalnosti.

U daljim fazama razvoja, ukoliko bi se aplikacija širila ka mrežnom radu, bilo bi moguće migrirati podatke iz LiteDB baze u neki od većih sistema kao što su PostgreSQL, MongoDB ili SQL Server.



Slika 5 – Dijagram baze podataka

# Ponovno upotrebljive komponente

Tokom razvoja desktop aplikacije za upravljanje putnim nalozima, identifikovano je nekoliko ključnih komponenata koje su implementirane tako da budu ponovno upotrebljive, kako bi se omogućila lakša nadogradnja sistema, bolja održivost i brže širenje funkcionalnosti u budućnosti.

**1. Komponenta za unos zaposlenih (AddEmployeeControl / AddEmployeeForm)**

Ova forma omogućava unos svih potrebnih podataka o zaposlenom: ime, prezime, JMBG, adresa i pozicija. Umesto da se svaki put ponovo piše forma za unos tih podataka (npr. prilikom dodele naloga, pregleda ili uređivanja), napravljena je kao zasebna forma koja se može koristiti bilo gde u aplikaciji. Time se smanjuje duplirani kod, a i svi budući zahtevi za izmenama (npr. dodavanje novog polja kao što je broj telefona) mogu se uraditi na jednom mestu.

**2. Komponenta za unos vozila**

Slično kao i za zaposlene, komponenta za unos vozila sa parametrima poput tipa vozila, registarske oznake, potrošnje i vrste goriva napravljena je tako da može da se koristi i prilikom registracije vozila, i prilikom izbora u kreiranju naloga. Ovakva ponovna upotrebljivost omogućava konzistentnost unosa i sprečava greške zbog različitih verzija iste logike u više delova aplikacije.

**3. Komponenta za kalkulaciju troškova**

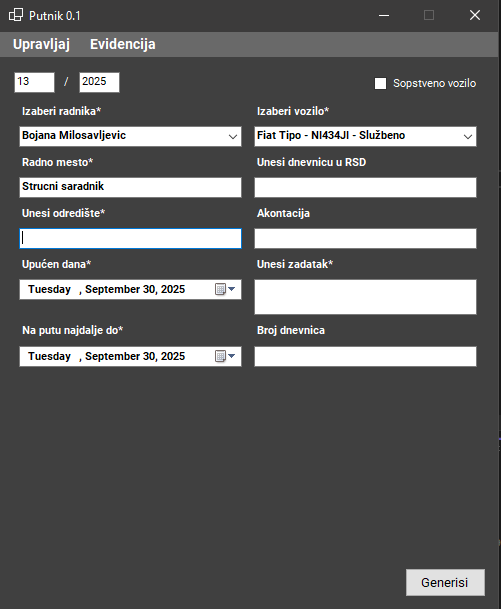
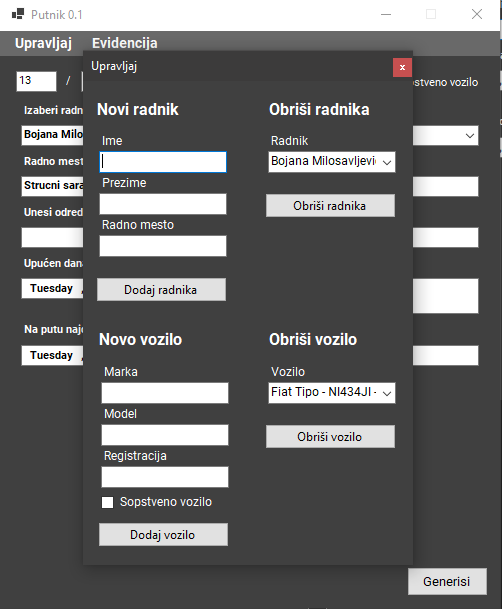
Logika za izračunavanje ukupnih troškova (gorivo, dnevnica, putarine, itd.) izdvojena je u posebnu funkcionalnu klasu ili servis koji se može pozvati sa različitih mesta - bilo iz forme za pregled, forme za kreiranje, ili buduće statistike. Time je omogućeno da se računanje troškova centralizuje, izbegavajući situaciju gde dve različite forme daju različite rezultate zbog neusklađenog koda.

**4. PDF generator**

Klasa zadužena za generisanje putnih naloga u PDF formatu je takođe modularno dizajnirana. Umesto da bude vezana za konkretnu formu, ona prihvata podatke kao parametre i generiše dokument na osnovu šablona. Ova komponenta se može koristiti i u budućim verzijama aplikacije (npr. za eksport svih naloga iz meseca), čime se izbegava potreba za pisanjem više sličnih generacija PDF dokumenata.

**5. Baza podataka i model sloj**

Modeli koji predstavljaju zaposlene, vozila i putne naloge su odvojeni i generički - što znači da nisu čvrsto vezani za korisnički interfejs. Ovo omogućava ponovno korišćenje baze (LiteDB) i u drugim projektima, kao što je web aplikacija ili mobilna verzija, bez potrebe za menjanjem osnovne strukture podataka.

Slika 6, 7 – Snimci ekrana aplikacije

# Računarstvo u oblaku

# Iako je aplikacija za upravljanje putnim nalozima zamišljena kao lokalna desktop aplikacija, mnoge njene komponente mogu se veoma efikasno preneti na cloud, čime bi se unapredila dostupnost, sigurnost i skalabilnost sistema.

# **1. Baza podataka korisnika, vozila i naloga**

# Cloud model: Platform as a Service (PaaS), npr. Azure Cosmos DB, Firebase Realtime Database, MongoDB Atlas

# Zašto: Korišćenjem cloud baze, omogućava se centralizovan pristup podacima sa više uređaja i lokacija. Na primer, sekretarica u sedištu firme može uneti nalog, dok ga nadređeni u drugom gradu može pregledati u realnom vremenu. Cloud baza takođe omogućava automatsko bekapovanje podataka, visoku dostupnost (uptime) i lako širenje kapaciteta ukoliko broj naloga naraste.

# Prednost: Nema potrebe za lokalnom instalacijom baze, neophodna je samo internet konekcija. Idealno za firme sa više lokacija ili korisnika koji rade na terenu.

# **2. Generisanje i skladištenje PDF dokumenata**

# Cloud model: Infrastructure as a Service (IaaS) ili Object Storage , npr. Amazon S3, Azure Blob Storage, Google Cloud Storage

# Zašto: Umesto da se generisani PDF fajlovi skladište lokalno (gde su podložni gubitku ili oštećenju), oni se mogu automatski čuvati u cloud storage-u. Time korisnici imaju pristup nalogu kad god im zatreba, bez obzira da li su na računaru ili mobilnom uređaju. Takođe, moguće je obezbediti verzije dokumenata i pristup samo ovlašćenim osobama.

# Prednost: Visoka dostupnost, lako deljenje i izbegavanje gubitka podataka zbog lokalnih kvarova.

# **3. Autentifikacija korisnika i logovanje aktivnosti**

# Cloud model: Software as a Service (SaaS) , npr. Firebase Authentication, Auth0

# Zašto: Sistem za prijavu korisnika, dodelu prava pristupa (admin, sekretar, nadređeni...) i beleženje logova korišćenja aplikacije može se izvesti pomoću gotovih cloud rešenja za autentifikaciju. Ova rešenja nude podršku za bezbednu prijavu (npr. putem mejla, šifre, pa čak i 2FA), uz zaštitu podataka u skladu sa najnovijim bezbednosnim standardima.

# Prednost: Ne mora se razvijati sopstveni login sistem, čime se štedi vreme i izbegavaju sigurnosni propusti. Ujedno se lako prati ko je kada i šta radio u aplikaciji.

# Mikroservisi U okviru predloženog sistema za evidenciju putnih naloga, razmatrana je mogućnost prelaska sa monolitne arhitekture na mikroservisnu, s ciljem poboljšanja skalabilnosti i održivosti aplikacije. U tom kontekstu identifikovane su tri ključne komponente koje bi mogle biti implementirane kao nezavisni mikroservisi: servis za rad sa putnim nalozima, servis za obračun troškova i servis za generisanje PDF dokumenata.

# Servis za putne naloge bio bi zadužen za kreiranje, izmenu i validaciju svih unosa koji se odnose na putovanja zaposlenih. Ovaj servis bi primao ulazne podatke o relaciji, zaposlenom, vozilu i datumu, proveravao njihovu ispravnost i čuvao ih u bazi. Na taj način bi se centralizovala logika validacije i obezbedilo konzistentno ponašanje aplikacije, bez obzira na to da li joj se pristupa putem desktop interfejsa ili mobilne aplikacije.

# Pored toga, posebno bi bio izdvojen servis za obračun troškova, koji bi na osnovu dobijenih parametara (kilometraža, tip goriva, prosečna potrošnja, broj dana puta) izračunavao ukupne troškove putovanja. Ovaj servis bi mogao biti dodatno unapređen integracijom sa eksternim izvorima podataka, poput API-ja za dnevne cene goriva, čime bi se obezbedila ažurnost i preciznost kalkulacija. Pošto je logika obračuna često podložna promenama (npr. zbog novih zakonskih propisa ili internog pravilnika firme), njeno izdvajanje u poseban servis omogućava lakšu nadogradnju bez potrebe da se menjaju ostali delovi sistema.

# Treći servis koji se prirodno izdvaja je servis za generisanje PDF dokumenata. Njegova uloga bila bi da na osnovu prethodno unetih i obrađenih podataka kreira standardizovan PDF putnog naloga spreman za štampu ili elektronsku arhivu. Budući da PDF generisanje često zahteva dodatne biblioteke i može biti procesorski zahtevno, korisno je da ova funkcionalnost bude izolovana kako bi se mogla nezavisno skalirati u zavisnosti od opterećenja. Takođe, ovakav servis bi mogao biti korišćen i u drugim modulima sistema (npr. izveštaji o troškovima po mesecu ili zaposlenom), čime se dodatno opravdava njegova izdvojenost.

# 

# Ranjivosti

U trenutnoj implementaciji aplikacije za upravljanje putnim nalozima, sistem ne koristi klasičnu relacionu bazu sa SQL upitima, već koristi **LiteDB**, što je dokument-orijentisana NoSQL baza optimizovana za lokalnu upotrebu. Pored toga, aplikacija **ne poseduje login mehanizam niti korisničku autentifikaciju**, pa samim tim ne postoji ni tačka ulaza koja bi bila pogodna za Brute Force napade na korisničke naloge. Zbog lokalnog režima rada, aplikacija je u velikoj meri izolovana od tipičnih web pretnji poput **SQL injection-a**, jer nema pozadinski server koji prima i obrađuje korisničke upite iz spoljnog sveta.

Međutim, u cilju demonstracije sigurnosne svesti i planiranja za budući razvoj (npr. potencijalnu web verziju aplikacije), analiziran je hipotetički slučaj SQL injection napada.

Zamislimo da aplikacija u budućnosti prelazi na relacionu bazu podataka (npr. SQLite ili PostgreSQL) i omogućava login korisnika putem forme. Ukoliko bi uneti podaci iz login forme direktno bili ubačeni u SQL upit bez validacije i sanacije, napadač bi mogao uneti zlonamerni string poput:

' OR '1'='1

što bi moglo dovesti do toga da sistem pogrešno dozvoli pristup.

SQL injection napadi su među najopasnijim, jer mogu omogućiti neautorizovani pristup sistemu, brisanje ili krađu podataka, i kompromitovanje celokupne baze. U slučaju aplikacije koja vodi evidenciju putnih naloga i zaposlenih, kompromitacija bi mogla otkriti osetljive lične podatke zaposlenih, putne troškove, pa čak i interne finansijske informacije firme.

Mere zaštite:

* Korišćenje **parametrizovanih upita** (umesto dinamičkog konkatinovanja stringova).
* Uvođenje **input validacije** i sanitizacije korisničkih podataka.
* Implementacija **rate limiting-a i lockout mehanizama** kako bi se onemogućio Brute Force pristup formama za prijavu (ako se kasnije dodaju).
* Logovanje i audit trail aktivnosti korisnika radi lakšeg praćenja sumnjivih radnji.

U trenutnoj verziji, aplikacija je projektovana za rad u zatvorenom okruženju, bez interakcije sa mrežnim servisima, što značajno smanjuje rizik od spoljnog napada. Ipak, u slučaju prelaska na cloud infrastrukturu ili dodavanja korisničke autentifikacije, ove bezbednosne smernice bi bile osnov za dalju implementaciju sigurnosti sistema.

# Atributi kvaliteta sistema

U kontekstu sistema za upravljanje putnim nalozima, atributi kvaliteta igraju ključnu ulogu u tome koliko je softver zaista koristan, stabilan i spreman za realnu upotrebu. U nastavku su analizirana **pet ključnih atributa kvaliteta** i njihov uticaj na funkcionalnost sistema.

**1. Sigurnost (Security)**  
Sigurnost je u svakom informacionom sistemu osnova poverenja. Iako trenutna verzija aplikacije radi lokalno i ne koristi korisničke naloge, planiranje bezbednosti je važno - posebno ako se u budućnosti bude prelazilo na web verziju ili rad u mrežnom okruženju. Zaštita podataka o zaposlenima, njihovim rutama i ličnim informacijama mora biti osigurana enkripcijom fajlova, kontrolom pristupa i zaštitom od manipulacije podacima. Kroz proaktivno razmišljanje o sigurnosti, sistem ostaje otporan na potencijalne napade i zloupotrebe.

**2. Performanse (Performance)**  
Performanse direktno utiču na korisničko iskustvo. Sistem mora brzo da reaguje - unos novih podataka, pretraga postojećih naloga i generisanje PDF dokumenata treba da se obavlja bez čekanja. Korišćenjem LiteDB baze, podaci se čuvaju lokalno, što omogućava brzi pristup bez mrežnog kašnjenja. Dobra optimizacija u kodu (npr. keširanje upita, optimizovano generisanje PDF-a) obezbeđuje da čak i korisnici sa slabijim računarima mogu da koriste aplikaciju bez problema.

**3. Pouzdanost (Reliability)**  
Pouzdanost znači da sistem mora raditi predvidivo i bez grešaka. Aplikacija mora sačuvati podatke čak i nakon pada sistema, mora detektovati nedostajuće vrednosti pre pokušaja generisanja PDF-a i mora korisniku jasno prijaviti grešku kada nešto pođe po zlu. Stabilnost aplikacije u radu sa većim brojem unosa i u raznim operativnim uslovima direktno podiže poverenje korisnika.

**4. Održivost (Maintainability)**  
Kada sistem ima jasnu arhitekturu, modularne komponente i lako razumljiv kod, lakše je dodavati nove funkcionalnosti ili menjati postojeće. Na primer, ako se u budućnosti promeni zakonski iznos dnevnice, održavanje aplikacije bi trebalo da se svodi na izmenu jedne konfiguracione konstante, a ne prepravku više delova koda. Ovo štedi vreme i smanjuje rizik od uvođenja novih grešaka prilikom izmene.

**5. Skalabilnost (Scalability)**  
Iako se aplikacija trenutno koristi lokalno i za manji broj korisnika, skalabilnost postaje važna ako se planira prelazak na verziju za više korisnika ili integracija sa cloud servisima. Dizajn baze i struktura koda moraju da predvide tu mogućnost. Npr. mogućnost da više korisnika koristi isti sistem preko mreže zahteva razdvajanje podataka po korisnicima, zaključavanje istovremenih pristupa i brigu o bezbednosti.

# Pouzdano programiranje

Pouzdano programiranje predstavlja pristup razvoju softvera koji ima za cilj da sistem ostane stabilan, tačan i otporan na greške, čak i u situacijama kada korisnik unosi pogrešne podatke, dolazi do izuzetaka ili se aplikacija koristi van očekivanog toka. U okviru ove aplikacije za upravljanje putnim nalozima, implementirane su tri ključne tehnike pouzdanog programiranja koje značajno doprinose ukupnoj robusnosti sistema.

**1. Validacija korisničkog inputa**

Kroz ceo sistem se sprovodi detaljna validacija unetih podataka. Umesto da se aplikacija oslanja na pretpostavku da će korisnik uvek uneti ispravne vrednosti, implementirane su kontrole koje proveravaju validnost unosa pre nego što se izvrši bilo kakva dalja obrada. Na primer, ukoliko korisnik pokuša da unese negativan broj kilometara, ili da generiše putni nalog bez unetog datuma povratka, sistem će na to pravovremeno reagovati prikazivanjem jasne poruke i sprečiti potencijalne greške u izračunavanju troškova. Takođe se proverava i logička doslednost - kao što je da li je datum povratka posle datuma polaska - što dodatno unapređuje integritet podataka.

**2. Korišćenje pomoćnih klasa i šablona za validaciju**

U cilju lakšeg održavanja i ponovne upotrebe, validacija unosa je izdvojena u posebnu klasu ValidationHelper. Ovaj pristup omogućava da se pravila validacije centralizuju i lako proširuju, umesto da se ista logika ponavlja kroz više delova aplikacije. Takav princip prati osnovne smernice dizajn šablona kao što je SRP (Single Responsibility Principle), jer je validacija izolovana i ne ometa logiku glavnih formi. Na taj način, dodavanje novih pravila ili izmena postojećih ne zahteva izmene na više mesta, već se obavlja na jednom centralnom mestu.

**3. Exception handling sa korisničkim porukama**

Na mestima gde se očekuje mogućnost greške u toku izvršavanja - poput generisanja PDF fajlova, pristupa fajl sistemu ili čitanja iz baze podataka - implementirani su try-catch blokovi. Umesto da program neočekivano prekine rad, sistem hvata izuzetke i korisniku prikazuje razumljive poruke, kao što je "Došlo je do greške pri generisanju dokumenta. Molimo proverite da li je destinacija dostupna." Tako ne samo da poboljšavamo korisničko iskustvo, već i štitimo podatke od oštećenja i omogućava da korisnik ispravi grešku bez restartovanja aplikacije.

# Scenariji testiranja

Testiranje softverskog sistema je ključno kako bi se osigurala tačnost, pouzdanost i korisnička upotrebljivost aplikacije. U okviru razvoja desktop aplikacije za upravljanje putnim nalozima, definisani su test scenariji koji obuhvataju sve ključne funkcionalnosti, od unosa i validacije podataka, preko obračuna troškova, do generisanja izveštaja. Svaki scenario sadrži jasno definisane ulaze i očekivane rezultate, čime se proverava ponašanje sistema u realnim i graničnim slučajevima.

Testovi su kreirani tako da obuhvate i pozitivne i negativne slučajeve korišćenja, a fokus je bio na najčešćim situacijama s kojima bi se krajnji korisnici mogli suočiti. Kroz ovu fazu se identifikuju potencijalne greške, problemi sa validacijom, kao i mesta gde korisniku treba pružiti jasnu povratnu informaciju.

|  |  |  |  |
| --- | --- | --- | --- |
| Broj | Naziv testa | Ulaz | Očekivani rezultat |
| 1 | Kreiranje validnog naloga | Zaposleni: Jelena, Datum: 01-05.04.2025, Vozilo: Fiat Panda, 300 km | Nalog uspešno kreiran, troškovi izračunati, PDF dostupan |
| 2 | Negativna kilometraža | Kilometraža: -120 km | Greška: „Kilometraža ne može biti negativna“, onemogućeno čuvanje |
| 3 | Izostavljen datum povratka | Samo datum polaska unet | Upozorenje: „Unesite i datum povratka“ |
| 4 | Zaposleni bez JMBG | Ime i prezime uneti, JMBG prazan | Greška: „JMBG je obavezan podatak“, unos se ne čuva |
| 5 | Generisanje PDF-a | Validan nalog unet | PDF se generiše, prikazuje podaci tačno |
| 6 | Pretraga po imenu | Zaposleni: Marija Ristić | Prikazani svi nalozi dodeljeni Mariji |
| 7 | Trošak dizel vozila | Vozilo: Kombi (dizel), 450 km | Gorivo se računa automatski po ceni i potrošnji |
| 8 | Brisanje zaposlenog u nalogu | Brisanje Ivana koji ima naloge | Greška: „Zaposleni se koristi u nalozima, ne može se obrisati“ |
| 9 | Vozilo bez registracije | Registracija prazna | Greška: „Registracija je obavezna“ |
| 10 | Prvo pokretanje | Prazna baza, nova instalacija | Aplikacija se otvara, prikazuje poruke: „Nema unetih naloga“ itd. |

# xUnit testiranje

Pošto je aplikacija razvijena u .NET okruženju, za automatsko testiranje funkcionalnosti korišćen je **xUnit** framework, koji predstavlja standardnu biblioteku za unit testing u C# aplikacijama. Implementirani testovi pokrivaju ključne domenske klase i poslovnu logiku, a testovi su koncipirani tako da se automatski proverava ispravnost ponašanja aplikacije pri različitim ulazima i stanjima.

**Analiza uspešnosti testova**

Svi testovi su **uspešno prošli** pri pokretanju komande dotnet test. Testovi su izolovani, ne zavise od prethodnog stanja aplikacije, i koriste nasumično generisane ID-eve kako bi se izbegle kolizije. Takođe, implementirana je logika za **čišćenje (cleanup)** testnih podataka nakon izvršavanja, čime se obezbeđuje da testovi ne utiču na produkcionu bazu (uz napomenu da je preporučljivo testove pokretati nad kopijom baze tokom razvoja).

# 

Slika 8 – xUnit testovi u razvojnom okruženju Visual Studio 2022

# Trello alat

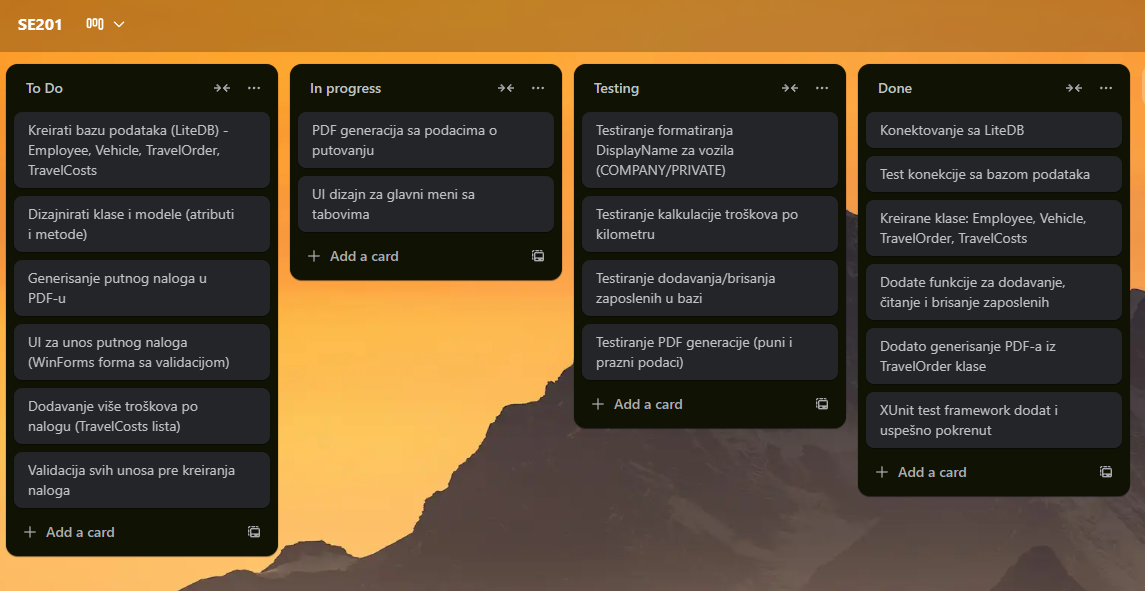
U okviru razvoja ove aplikacije, Trello je iskorišćen kao alat za upravljanje projektom i praćenje napretka svih razvojnih aktivnosti. Trello omogućava jasan i vizuelno organizovan prikaz svih zadataka kroz četiri osnovne faze: To Do, In Progress, Testing i Done. Ovaj pristup omogućava timu (ili pojedincu, u slučaju samostalnog projekta) da ima uvid u kompletan tok rada, prioritete, kao i da na jednostavan način ažurira status svakog zadatka.

U listi "To Do" nalaze se svi zadaci koji još nisu započeti, ali su planirani u okviru projekta. Tu su obuhvaćeni zadaci kao što su kreiranje baze podataka (koristeći LiteDB), dizajn klasa i modela (Employee, Vehicle, TravelOrder itd.), kreiranje PDF putnog naloga, izrada korisničkog interfejsa za unos podataka, kao i dodavanje mogućnosti za višestruke troškove i validaciju unosa. Ova faza predstavlja planiranje funkcionalnosti i sve ono što mora biti urađeno kako bi sistem bio potpun.

U fazi "In Progress" nalaze se zadaci koji su trenutno u radu. U ovom slučaju, u razvoju su generisanje putnog naloga u PDF formatu sa dinamičkim podacima i dizajn glavnog interfejsa sa tabovima, što su ključne funkcionalnosti koje zahtevaju pažljivo testiranje i integraciju sa ostalim delovima aplikacije.

Faza "Testing" sadrži zadatke koji su implementirani i sada su u procesu provere ispravnosti. Test scenariji uključuju provere formatiranja prikaza podataka (npr. DisplayName vozila), validaciju kalkulacija troškova, kao i testove za dodavanje i brisanje zaposlenih iz baze. Posebno je naglašeno i testiranje PDF generacije sa različitim ulazima (punim i praznim), što osigurava da sistem može da obradi različite realne scenarije korišćenja.

Konačno, u fazi "Done" su svi zadaci koji su uspešno završeni i implementirani. To uključuje konekciju sa bazom podataka, testiranje te konekcije, kreiranje svih potrebnih modela i funkcionalnosti za rad sa bazom, generisanje PDF-a, kao i integraciju i uspešno pokretanje XUnit test okruženja.



Slika 9 – Trello board

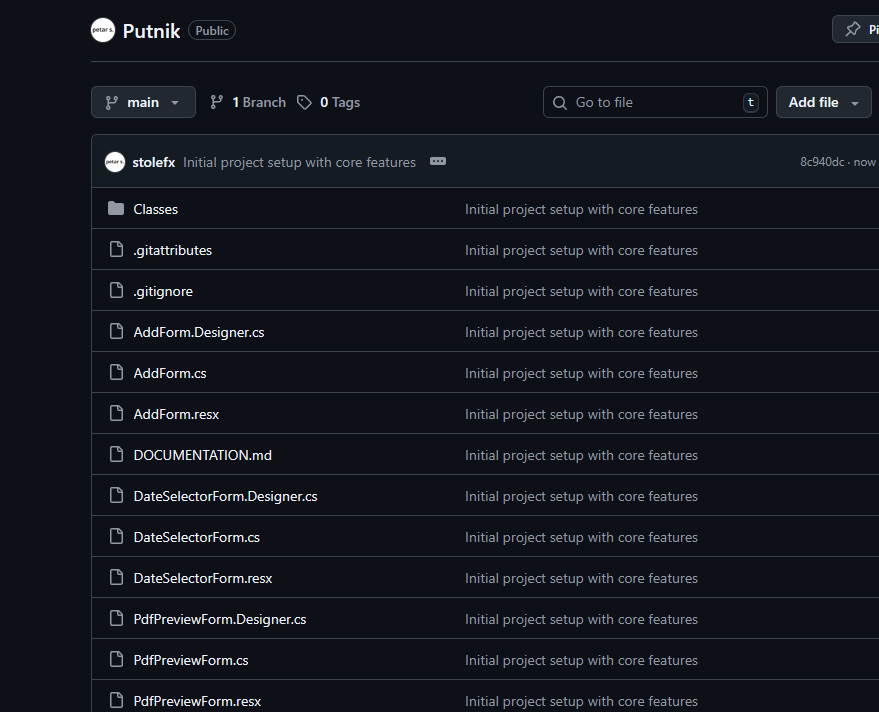
# Git

Git je najpopularniji alat za verzionisanje koda i upravljanje softverskim projektima danas. On omogućava programerima da prate sve izmene u kodu, rade paralelno na različitim delovima projekta, čuvaju bezbedno svaku verziju i lako se vrate na bilo koju prethodnu tačku razvoja ako dođe do problema. Git funkcioniše tako što čuva kompletnu istoriju svih izmena kroz posebne “commit”-ove, omogućavajući jasnu evidenciju ko je, kada i šta menjao.

Git se koristi i za rad u timu i za samostalne projekte – svako može da napravi svoju granu (branch), isproba nove ideje ili doda novu funkcionalnost, a zatim te izmene spoji sa glavnim projektom tek kada su spremne i ispravne. Na taj način se izbegava haos i greške koje nastaju kada više ljudi radi direktno na istim fajlovima. Kroz jednostavne komande, Git omogućava lako čuvanje, deljenje i spajanje svih izmena, što čini razvoj softvera bržim, sigurnijim i preglednijim. U kombinaciji sa online servisima kao što je GitHub, moguće je još lakše sarađivati i čuvati svoj rad na daljinu.

Link do repozitorijuma:

<https://github.com/stolefx/Putnik>



Slika 10 – Repozitorijum na platformi Github

# Zaključak

U ovom radu razvijen je **softverski sistem za kreiranje i upravljanje putnim nalozima**, sa jasno definisanom strukturom klasa, bazom podataka, korisničkim interfejsom i funkcionalnostima koje podržavaju sve ključne aspekte službenih putovanja. Sistem je izrađen kao **desktop aplikacija u .NET okruženju**, korišćenjem **WinForms tehnologije**, uz podršku **LiteDB** baze podataka i **PDF generacije** radi štampanja naloga.

U toku razvoja projekta primenjeni su brojni principi **pouzdanog programiranja**, uključujući validaciju korisničkog unosa, jasno razdvajanje logike po klasama, i rukovanje izuzecima sa korisnički čitljivim porukama. Posebna pažnja posvećena je **testiranju**, gde su napisani **xUnit testovi** koji pokrivaju različite scenarije - od ispravnosti prikaza informacija do uspešnog generisanja PDF-a i rada sa bazom podataka. Kroz to je osigurano da sistem bude robustan, stabilan i spreman za produkcionu upotrebu.

Pored funkcionalnog aspekta, analizirane su i **arhitektonske karakteristike** sistema. Identifikovani su delovi koji bi se lako mogli izdvojiti u mikroservise u slučaju migracije na distribuirano okruženje, a komponente sistema su razmatrane i u kontekstu mogućeg premeštanja u cloud infrastrukturu. Takođe, razmotrene su i potencijalne **bezbednosne pretnje**, kao što su brute force napadi ili neovlašćeno pristupanje bazi, uz predloge za preventivne mere.

Planiranje projekta je vođeno pomoću **Trello alata**, gde su jasno definisane faze razvoja (To Do, In Progress, Testing, Done), što je omogućilo transparentno praćenje napretka i zadataka. Svaki korak je bio dokumentovan, što je olakšalo i testiranje i kasniju analizu sistema.

Na kraju, ceo projekat predstavlja **uspešno realizovan softverski sistem srednje veličine**, koji demonstrira razumevanje principa objektno-orijentisanog programiranja, rada sa lokalnim bazama, generisanja dokumenata, kao i praktične primene testiranja i verzionisanja koda. Projekat je skalabilan i lako proširiv - nove funkcionalnosti, kao što su različiti tipovi vozila, dinamički izračun dnevnica, ili integracija sa online bazama, mogu se lako dodati u budućnosti.

# Literatura Microsoft Learn. (2023). **WinForms Overview**. https://learn.microsoft.com/en-us/dotnet/desktop/winforms/

LiteDB Documentation. (2024).  
**LiteDB v5.0 – A lightweight .NET NoSQL Document Store in a single data file**.  
https://www.litedb.org/

Microsoft Learn. (2023).  
**Working with PDFs in C# using System.Drawing and PDF libraries**.  
https://learn.microsoft.com/en-us/dotnet/api/system.drawing

xUnit Documentation. (2024).  
**xUnit.net Documentation**.  
https://xunit.net/

 Martin, R. C. (2009).  
**Clean Code: A Handbook of Agile Software Craftsmanship**.