Projekt: *NWTiS_2021_2022 v1.1*

Sustav se treba sastojati od sljedećih elemenata:

1. aplikacija **{korisnicko_ime}-aplikacija_1** predstavlja servis za izračunavanje udaljenosti između aerodroma koji koriste ostale aplikacije. Svoj rad temelji na poslužitelju na mrežnoj utičnici (socket server) na određenom vratima (portu) (postavkom se određuje). Poslužitelj treba biti realiziran kao više dretveni sustav s ograničenim brojem dretvi (postavkom se određuje). Aplikacija nema bazu podataka niti datoteku s podacima za aerodrome. Aplikacija može primiti komandu QUIT u bilo kojem načinu rada i treba prekinuti rad poslužitelja. Aplikacija može primiti komandu STATUS u bilo kojem načinu rada i treba vratiti trenutni status poslužitelja. Aplikacija je na početku u statusu hibernacije i čeka svoju inicijalizaciju koja se provodi komandom INIT. Poslužitelj dok ne primi komandu INIT odbija sve ostale komande. Nakon INIT jedino može primiti komandu LOAD, a odbija sve ostale komande. Komandom LOAD prenose se podaci o aerodromima za rad aplikacije. Nakon završetka komande LOAD poslužitelj postaje aktivan i može primati ostale komande. Ako primi komandu CLEAR briše iz memorije podatke o aerodromima i vraća se u stanje hibernacije kao na početku prije inicijalizacije. Komande:

- **STATUS**
  - Npr. STATUS
  - Zadatak: vraća trenutni status poslužitelja
  - Korisniku se vraća odgovor OK n. Status n: 0 – hibernira, 1 – inicijaliziran, 2 - aktivan
  - Npr. OK 2

- **QUIT**
  - Npr. QUIT
  - Zadatak: prekid rada poslužitelja
  - Korisniku se vraća odgovor OK.
  - Npr. OK
- **INIT**
  - Npr. INIT
  - Zadatak: inicijalizacija poslužitelja
  - Korisniku se vraća odgovor OK
  - Npr. OK

- **LOAD [{...}]**
  - Npr. LOAD [{...}]
  - Zadatak: učitavanje podataka o aerodromima (koristi se kolekcija s klasom Aerodrom u formatu JSON) u kolekciju aerodroma na poslužitelja
  - Korisniku se vraća odgovor OK n
  - n označava broj aerodroma koji su učitani
  - Npr. OK 10

- **DISTANCE icao1 icao2**
    - Npr. DISTANCE LDZA LOWW
    - Provjera da li postoje aerodromi icao1 i icao2 u kolekciji aerodroma na poslužitelju. Ako bilo koji od njih ne postoji šalje kod za pogrešku. Ako postoje aerodromi vraća udaljenosti između dva aerodroma na temelju njihovih GPS pozicija. Algoritam treba uzeti u obzir zakrivljenost Zemlje. Vraća broj km.
    - Korisniku se vraća odgovor OK broj
    - Npr. OK 271

- **CLEAR**
    - Npr. CLEAR
    - Zadatak: pražnjenje kolekcije aerodroma na poslužitelju i prijelaz u stanje hibernacije
    - Korisniku se vraća odgovor OK
    - Npr. OK

Kodovi pogrešaka su:

- Kada poslužitelj hibernira a stigla je komanda (LOAD, DISTANCE ili CLEAR) vraća odgovor ERROR 01 tekst (tekst objašnjava razlog pogreške)
- Kada je poslužitelj inicijaliziran a stigla je komanda (INIT, DISTANCE ili CLEAR) vraća odgovor ERROR 02 tekst (tekst objašnjava razlog pogreške)
- Kada je poslužitelj aktivan a stigla je komanda (INIT ili LOAD) vraća odgovor ERROR 03 tekst (tekst objašnjava razlog pogreške)
- Kada ne postoji prvi aerodrom vraća odgovor ERROR 11 tekst (tekst objašnjava razlog pogreške)
- Kada ne postoji drugi aerodrom vraća odgovor ERROR 12 tekst (tekst objašnjava razlog pogreške)
- Kada ne postoje oba aerodroma vraća odgovor ERROR 13 tekst (tekst objašnjava razlog pogreške)
- U slučaju bilo koje ostale pogreške vraća odgovor ERROR 14 tekst (tekst objašnjava razlog pogreške)

2. web aplikacija **{korisnicko_ime}-aplikacija_2** učitava konfiguracijske podatke putem slušača aplikacije kod pokretanje aplikacije i upisuje ih u atribut konteksta pod nazivom „Postavke“. Naziv datoteke konfiguracijskih podataka zapisan je u web.xml kao inicijalni parametar konteksta „konfiguracija“ (jedna od datoteka NWTiS.db.config_1.xml ili NWTiS.db.config_2.xml). Nakon učitavanja konfiguracijskih podataka potrebno je pokrenuti dretvu koja radi u ciklusima identičnog trajanja. Dretva ima dva brojača ciklusa. Prvi je stvarni brojač ciklusa, a drugi je virtualni brojač ciklusa koji služi kod korekcija vremena spavanja. Oni imaju iste vrijednosti sve dok se ne desi jedna od kasnijih opisanih situacija. Vrijeme početka rada 1. ciklusa označava se kao t1. Vrijeme t1 izraženo je u milisekundama i sva kasnija vremena trebaju se pretvarati u milisekunde (sekunde, sati, dani). Zadano je vrijeme ciklusa (postavka „ciklus.vrijeme“, u sekundama), koje se označava kao Δtc. Vrijeme početka 2. ciklusa dobije se kao t2 = t1 + Δtc. Općenito može se zapisati da vrijeme početka i+1 tog ciklusa kao ti+1 = ti + Δtc ili kao ti+1 = t1 + (i * Δtc). To znači da svaki ciklus započinje u vrijeme koje je jednako vremenu početka prethodnog ciklusa uvećano za zadano vrijeme ciklusa. U vrijeme trajanje ciklusa ulazi vrijeme efektivnog rada koji se obavlja u ciklusu, označava se kao Δte, a preostalo vrijeme do zadanog vremena ciklusa dretva treba spavati, označava se kao Δts. To znači da se vrijeme spavanja izračunava kao Δts = Δtc - Δte. Ako je vrijeme efektivnog rada dretve dulje od zadanog vremena dretve (Δte > Δtc) tada se vrijeme spavanja izračunava kao razlika između prvog umnoška zadanog vremena dretve koji je veći od vremena efektivnog rada dretve i vremena efektivnog rada dretve, Δts = (n * Δtc) - Δte, gdje je n=2,3,… prvi koji ispunjava uvjet (n * Δtc) > Δte. Ovo je prva situacija u kojoj dolazi do razlike u stvarnom i virtualnom brojaču ciklusa. Virtualni brojač ciklusa će sada biti povećan za n, dok će stvarni brojač ciklusa biti povećan za 1. Korekcija vremena spavanja provodi se u određenim situacijama, (postavka „ciklus.korekcija“, broj, npr. 10), koja se označava kao ck. U svakom j-tom ciklusu (j - stvarni brojač ciklusa) u kojem je j % ck = 0, potrebno je napraviti korekciju vremena spavanja kako bi se postigla visoka točnost ciklusa (razina 1 milisekunda) u odnosu na početak rada (1. ciklus), broj ciklusa i zadano vrijeme ciklusa. Time će se korigirati odstupanje koje nastaje zbog internog rada sustava prilikom prijelaza dretve u spavanje i buđenja nakon spavanja. Odnosno postignut će se da tk+1 = t1 + (k * Δtc) (k - virtualni brojač ciklusa).

   Dretva ima zadatak da preuzima podatke o polascima i dolascima letova s određenog skupa aerodroma i upiše ih u tablice u bazi podataka. Trenutno vrijeme na računalu označavaju kao tr. Zbog točnosti podataka ne želi se preuzimati podaci koji su još u fazi usklađivanja tako da se daje vremenski odmak od stvarnog vremena, koji se označava kao Δto (postavka „preuzimanje.odmak“, u danima). U jednom ciklusu dretve preuzimaju se podaci za sve aerodrome iz skupa. Kako ne bi došlo do zagušenja poslužitelja s kojeg se preuzimaju podaci, nakon obrade svakog aerodroma potrebno je napraviti kratku pauzu tj. dretva treba spavati određeno vrijeme (postavka „preuzimanje.pauza“, u milisekundama). Podaci se preuzimaju za zadano razdoblje polaska i dolaska aerodroma (postavke „preuzimanje.od“ i „preuzimanje.do“, zapisane su u formatu dd.mm.gggg

hh:mm:ss), koje se označavaju kao tp1 i tpn. U jednom ciklusu dretve preuzimaju se podaci za određeno vrijeme (postavka „preuzimanje.vrijeme“, u satima), koje se označava kao Δtv. Dretva u 1. ciklusu preuzima podatke za interval vremena koji je definiran početkom tp1 i krajem tp1 + Δtv. Dretva u 2. ciklusu preuzima podatke za interval vremena koji je definiran početkom tp2 = tp1 + Δtv i krajem tp2 + Δtv. Dretva prestaje s radom u i-tom ciklusu kada je tpi > tpn. Ako je tpi > tr – Δto tada dretva treba spavati 1 dan. Ovo služi da se ne preuzimaju podaci koji se usklađuju u servisu OpenSky Network. U slučaju spavanja dretve u trajanju od 1 dan dolazi se do druge situacije u kojoj dolazi do razlike u stvarnom i virtualnom brojaču ciklusa. Virtualni brojač ciklusa će biti povećan za m, m = 1 dan/Δtc, dok će stvarni brojač ciklusa biti povećan za 1.

Skup aerodroma koji se prate definiran je u tablici „AERODROMI_PRACENI“. Polasci letova upisuju se u tablicu „AERODROMI_POLASCI“ ako imaju definiranog aerodroma dolaska (!= null). Dolasci letova upisuju se u tablicu „AERODROMI_DOLASCI“ ako imaju definiranog aerodroma polaska (!= null). Ukoliko u pojedinom ciklusu preuzimanje podataka za određeni aerodrom nije uspješno tada se upisuje u tablicu „AERODROMI_PROBLEMI“.

Aplikacija se smatra nevažećim (0 bodova) ukoliko ne postoji:

- **minimalno 20 aerodroma za preuzimanje (EBBR, EDDF, EDDM, EGGP, EGLL, EIDW, EPWA, GCLP, HEGN, LDZA, LEBL, LEPA, LFPG, EDDS, LIPZ, LOWW, LTBJ, LZLH, LJLJ, OMDB). Može više aerodroma no prethodno prikazani moru biti.**
    - **minimalno 100.000 preuzetih polazaka s aerodroma**
    - **minimalno 100.000 preuzetih dolazaka na aerodrome**
    - **minimalno 15 dana u cijelosti za koje su preuzeti podaci polazaka i dolazaka**
    - **datoteka s pripremljen SQL komandama kojima se dokazuju:**
        - **prethodni uvjeti**
        - **broj preuzetih podataka po danima za sve aerodrome**
        - **broj preuzetih podataka po danima za sve aerodrome pojedinačno**
        - **broj preuzetih podataka po danima za odabrani aerodrome**

3. web aplikacija **{korisnicko_ime}-aplikacija_3** predstavlja središnji autentikacijski, autorizacijski i nadzorni servis za ostale aplikacije na bazi RESTful (JAX-RS) web servisa na krajnjoj točki „api“. Aplikacija koristi tablice korisnici, grupe i uloge (sa strukturom i relacijama kao što su definirane na vježbama). Web servis ima resurse: putanja „provjere“, putanja „korisnici“, putanja „aerodromi“, putanja „serveri“. Potrebno se držati zadanih osobina i realizirati sljedeće operacije:
    1. Sve operacije RESTful web servisa na resursu „provjere“ moraju kod slanja zahtjeva pripremiti podatke za autentikaciju u atributima zaglavlja pod nazivima „korisnik“ i „lozinka“.
    2. Sve operacije RESTful web servisa na ostalim resursima moraju kod slanja zahtjeva pripremiti podatke za autorizaciju u atributima zaglavlja pod nazivima „korisnik“ i „zeton“. Šalje se prethodno dobiveni žeton iz GET zahtjeva na resursu „provjere“. Sve operacije vraćaju status 200 ako je aktivan žeton ili 4xx ako nije aktivan uz opis razloga. Kod aktivnog žetona operacije vraćaju podatke svoje domene. Kod provjere potrebno je usporediti korisničko ime iz zahtjeva i korisničko ime koje je kreiralo žeton.
    3. Sve operacije RESTful web servisa koje trebaju dodatne ulazne podatke (osim putem zaglavlja i parametara u adresi) šalju ih u application/json formatu sa strukturom koja je zadana kod pojedine operacije ili je ostavljena na vlastiti izbor.
    4. Sve operacije RESTful web servisa vraćaju odgovor u application/json formatu pomoću klase Response tako da vraćaju status izvršene operacije i objekt tražene klase kao entitet.
    5. Sve operacije na resursu serveri koriste adresu (a) i port (p) poslužitelja, koji se preuzimaju iz datoteke postavki.
    6. Resurs „provjere“:
        1. GET metoda - osnovna adresa – vraća status 200 ako je uspješna autentikacija ili status 401 ako nije uspješna uz opis razloga. Kod uspješne autentikacije vraća žeton (t) i vrijeme do kada traje (v). Format vraćenih podataka je {zeton: t, vrijeme: v}. Žeton predstavlja vrijednost koja se dobije povećavanje za 1 kod svakog kreiranja novog žetona. Svaki kreirani žeton sprema se u tablicu u bazi podataka uz upis podataka o korisniku (iz parametra zaglavlja) , vremenu do kada vrijedi (trenutno vrijeme na poslužitelju + trajanje žetona) i statusu (0 – nije važeći, 1 – važeći). Inicijalno je važeći. Trajanje žetona određeno je postavkom zeton.trajanje.
        2. GET metoda - putanja "{token}" - vraća status 200 ako je žeton aktivan (status važeći i vrijeme do kada vrijedi nije isteklo) ili status 401 ako nije od korisnika uz opis razloga ili status 408 je isteklo vrijeme žetona uz opis razloga. Kod provjere potrebno je usporediti korisničko ime iz zahtjeva i korisničko ime koje je kreiralo žeton.
        3. DELETE metoda - putanja "{token}" - vraća status 200 ako je aktivan žeton (postavlja mu status da nije važeći) ili status 401 ako nije od korisnika uz opis razloga ili status 408 je isteklo vrijeme žetona uz opis razloga. Kod provjere potrebno je usporediti korisničko ime iz zahtjeva i korisničko ime koje je kreiralo žeton.

4. DELETE metoda - putanja "korisnik/{korisnik}" - vraća status 200 ako je korisnik imao barem jedan aktivan žeton (svim aktivnim postavlja se status da nije važeći) ili status 404 ako nema ni jedan aktivan žeton uz opis razloga ili status 401 ako korisnik nema ovlaštenje za brisanje. Tuđe žetone može brisati korisnik koji je član grupe (u tablici grupe) iz postavke sustav.administratori.

7. Resurs „korisnici“:
    1. GET metoda - osnovna adresa - vraća kolekciju korisnika. Za korisnika podaci trebaju odgovarati klasi Korisnik.
    2. POST metoda - osnovna adresa - dodaje korisnika u bazu podataka. Ulazni podaci za korisnika trebaju odgovarati klasi Korisnik.
    3. GET metoda - putanja "{korisnik}" - vraća podatke izabranog korisnika. Za korisnika podaci trebaju odgovarati klasi Korisnik.
    4. GET metoda - putanja "{korisnik}/grupe" - vraća kolekciju grupa izabranog korisnika. Za grupe podaci trebaju odgovarati klasi Grupa.
8. Resurs „aerodromi“:
    1. GET metoda - osnovna adresa - vraća kolekciju aerodroma. Za aerodrom podaci trebaju odgovarati klasi Aerodrom.
    2. POST metoda - osnovna adresa – dodaje aerodrom za preuzimanje. Za aerodrom podaci trebaju odgovarati klasi Aerodrom.
    3. GET metoda - putanja "{icao}" - vraća aerodrom koji ima traženi icao. Za aerodrom podaci trebaju odgovarati klasi Aerodrom.
    4. GET metoda - putanja "{icao}/polasci?vrsta=x&od=vrijemeOd&do=vrijemeDo“ – vraća kolekciju letova u zadanom intervalu koji su poletjeli s aerodroma koji ima traženi icao. Ako je vrsta 0, vrijemeOd/Do je u formatu dd.mm.gggg. Ako je vrsta 1, vrijemeOd/Do je broj sekundi od 01.01.1970. Za letove podaci trebaju odgovarati klasi AvionLeti. Podaci se preuzimaju iz baze podataka koje je aplikacija {korisnicko_ime}-aplikacija_2 preuzela s OpenSkyNetwork.
    5. GET metoda - putanja "{icao}/dolasci?vrsta=x&od=vrijemeOd&do=vrijemeDo“– vraća kolekciju letova u zadanom intervalu koji su sletjeli na aerodrom koji ima traženi icao. Ako je vrsta 0, vrijemeOd/Do je u formatu dd.mm.gggg. Ako je vrsta 1, vrijemeOd/Do je broj sekundi od 01.01.1970. Za letove podaci trebaju odgovarati klasi AvionLeti. Podaci se preuzimaju iz baze podataka koje je aplikacija {korisnicko_ime}-aplikacija_2 preuzela s OpenSkyNetwork.
    6. GET metoda - putanja "{icao1}/{icao2}" - šalje komandu DISTANCE icao1 icao2 na poslužitelj {korisnicko_ime}-aplikacija_1. Postavkama se određuje adresa i port poslužitelja. Vraća status 200 ako je komanda uspješna i vraćena vrijednost od poslužitelja n. Format vraćenih podataka je {udaljenost: n}. Vraća status 404 ako nije uspješna uz tekst pogreške koji je primljen od poslužitelja.
9. Resurs „serveri“:
    1. GET metoda - osnovna adresa – šalje komandu STATUS na poslužitelj {korisnicko_ime}-aplikacija_1. Format vraćenih podataka je {adresa: a, port: p}.
    2. GET metoda - putanja "{komanda}" - šalje komandu (QUIT, INIT ili CLEAR) na poslužitelj {korisnicko_ime}-aplikacija_1. Vraća status 200 ako je komanda uspješna. Vraća status 400 ako nije uspješna uz tekst pogreške koji je primljen od poslužitelja.

3. POST metoda - putanja "/LOAD" - šalje komandu LOAD na poslužitelj {korisnicko_ime}-aplikacija_1. Šalje se kolekcija List<Aerodromi> u JSON formatu. Vraća status 200 ako je komanda uspješna i vraćena vrijednost od poslužitelja jednaka broju aerodroma koji se šalju. Vraća status 409 ako nije uspješna uz tekst pogreške koji je primljen od poslužitelja.

Za testiranje RESTful servisa potrebno je pripremit SoapUI datoteku ili sh datoteku s curl komandama koja sadrži pozive svih implementiranih operacija. **Aplikacija se smatra nevažećim (0 bodova) ukoliko ne postoji funkcionalna datoteka za testiranje**.

4. web aplikacija **{korisnicko_ime}-aplikacija_4** bavi se administrativnim poslovima kroz korisničko sučelje realizirano s Jakarta MVC. Svaki obrazac u bilo kojem pogledu treba POST metodom slati podatke. Ova aplikacija nema vlastitu bazu podataka. Svoj rada temelji na slanju komandi prema {korisnicko_ime}-aplikacija_1 i slanju zahtjeva na RESTful web servis iz {korisnicko_ime}-aplikacija_3. Potrebno se držati zadanih osobina i realizirati sljedeće dijelove:

   - pogled 4.1:
     - registraciju korisnika. Šalje se POST zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „korisnici“. Za slanje autentikacijskih podataka korisnika preuzimaju podaci iz postavki sustav.korisnik i sustav.lozinka.
   - pogled 4.2:
     - prijavljivanje korisnika. Šalje se GET zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „provjere“. Za ovaj dio potrebno je provesti autentikaciju korisnika. Ako je operacija uspješna dobiveni žeton treba spremiti kako bi se koristio kod sljedećih zahtjeva RESTful web servisu iz {korisnicko_ime}-aplikacija_3.
   - pogled 4.3:
     - pregled korisnika. Korisnici se dobiju slanjem GET zahtjeva RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „korisnici“. Ako je korisnik član grupe iz postavke sustav.administratori tada se uz podatke o pojedinom korisniku dodaje poveznica za brisanje žetona tog korisnika. Brisanje se obavlja slanjem DELETE zahtjeva RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „provjere“ za određenog korisnika. Postoji poveznica za brisanje vlastitog trenutnog žetona. Brisanje se obavlja slanjem DELETE zahtjeva RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „provjere“ za određeni žeton.
   - pogled 4.4:
     - upravljanje poslužiteljem iz {korisnicko_ime}-aplikacija_1. Postavkama se određuje adresa i port poslužitelja. Na početku pogleda ispisuje se status poslužitelja koji se dobije komandom STATUS. U padajućem izborniku nalazi se opis komandi (Inicijalizacija poslužitelja, Prekid rada poslužitelja, Učitavanje podataka, Brisanje podataka). Gumbom se aktivira odabrana komanda. Pogled sadrži ispis statusa izvršene komande i tekst ako je vraćena pogreška.

5. ~~web aplikacija **{korisnicko_ime}-aplikacija_5** pruža JAX-WS (SOAP) servise na krajnjim točkama „aerodromi“ i „meteo“, i pruža WebSocket krajnju točku „info“. U radu šalje zahtjeve RESTful web servisu iz {korisnicko_ime}-aplikacija_3 i servisu OpenWeatherMap. Ova aplikacija nema vlastitu bazu podataka Potrebno se držati zadanih osobina i realizirati sljedeće metode:~~
    1. JAX-WS krajnja točka „aerodromi“:
        1. List<AvionLeti> dajPolaskeDan(String korisnik, String zeton, String icao, String danOd, String danDo) – šalje GET zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3
        2. List<AvionLeti> dajPolaskeVrijeme(String korisnik, String zeton, String icao, String vrijemeOd, String vrijemeDo) – GET zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3
        3. boolean dodajAerodromPreuzimanje(String korisnik, String zeton, String icao) – šalje GET zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3. Kod svakog uspješnog dodavanja aerodroma za preuzimanje šalje poruku putem WebSocket svim aktivnim sesijama korisnika.
    2. JAX-WS krajnja točka „meteo“:
        1. MateoPodaci dajMeteo(String icao) - dohvaća meteorološke podatke za zadani aerodrom iz primljenog argumenta. Kako bi se dobila GPS lokacija aerodrom šalje se GET zahtjev RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „aerodromi“. S lokacijom aerodroma poziva se metoda getRealTimeWeather(lokacija) na objektu OWMKlijent. Nakon uspješnog dohvaćanja podataka šalje JMS poruku (ObjectMessage) s podacima na red čekanja „jms/NWTiS_{korisnicko_ime}“.
    3. WebSocket krajnja točka „info“:
        1. dajMeteo(info) - obavještavanje svojih aktivnih korisnika o trenutnom vremenu na poslužitelju aplikacije i broju aerodroma za koje se prikupljaju podaci.

Za testiranje JAX-WS servisa potrebno je pripremit SoapUI datoteku koja sadrži pozive svih implementiranih operacija. **Aplikacija se smatra nevažećim (0 bodova) ukoliko ne postoji funkcionalna datoteka za testiranje**.

6. projekt **{korisnicko_ime}-aplikacija_6** sastoji od Maven modula/projekata: {korisnicko_ime}-modul_6_jpa i {korisnicko_ime}-modul_6_wa.

   JPA biblioteka **{korisnicko_ime}- aplikacija _6_jpa** sadrži JPA entitete za rad s podacima u bazi podataka. Baza podataka sadrži samo tablice korisnici, grupe, uloge, PUTOVANJA, PUTOVANJA_LETOVI. Koristi JPA perzistentnu jedinicu NWTiS_{korisnicko_ime}-PU koja se povezuje na izvor podataka na poslužitelju pod nazivom „jdbc/NWTiS_{korisnicko_ime}-bp_2" uz pridruženi bazen veza.

   Web aplikacija **{korisnicko_ime}-modul_6_wa** koristi JPA biblioteku {korisnicko_ime}-aplikacija _6_jpa. Za rad s JPA entitetima koristi se Criteria API. ~~Ako se koristi JPQL tada se dobije 60% bodova. Ne može se koristiti Criteria API i JPQL istovremeno~~. Aplikacija sadrži potrebne Singleton Session Bean (SiSB), Stateful Sesssion Bean (SfSB), Stateless Sesssion Bean (SlSB), Message-Driven Bean i JPA za pristup do podataka u bazi podataka. Aplikacija preuzima JMS poruke iz reda poruka „jms/NWTiS_{korisnicko_ime}" koje su poslane kod dodavanja aerodroma za preuzimanje. Primljene JMS poruke spremaju se u SiSB. Ako aplikacija prestaje s radom (brisanje SiSB), potrebno je poruke serijalizirati na vanjski spremnik (naziv datoteke u postavkama s apsolutnom adresom). Kada se aplikacija podiže (kreiranje SiSB) potrebno je učitati serijalizirane poruke (ako postoji datoteka) u SiSB. Postoji mogućnost pražnjenja svih primljenih JMS poruka. Korisničko sučelje može se realizirati putem JSF (facelets)/PrimeFaces. Ako se korisničko sučelje realizira putem Jakarta MVC tada se dobije 50% u odnosu da je ista funkcionalnost realizirana putem JSF/PrimeFaces. Kod Jakarta MVC svaki obrazac u bilo kojem pogledu treba POST metodom slati podatke. Svi pogledi moraju biti realizirani putem istog načina (JSF/PrimeFaces ili Jakarta MVC). Za sve poglede osim 6.1 potrebno je imati prethodno prijavljenog korisnika jer se inače ne dozvoljava ulaz u pogled te se preusmjerava na pogled za pogrešku. Potrebni su sljedeći pogledi:

   - pogled 6.1:
     - prijavljivanje korisnika. Prvo se provjerava u lokalnoj bazi podataka. Ako nije uspješna, prikazuje informaciju o nastaloj situaciji. Ako je uspješna, šalje zahtjev GET metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „provjere" kako bi dobio žeton za rad s drugim metodama. Na pogledu postoji poveznica za odjavljivanje koja odjavljuje korisnika od aplikacije i šalje zahtjev DELETE metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „provjere" za brisanje važećeg žetona korisnika. Ako ne postoji žeton i/ili korisnik nije prijavljen preusmjerava se na pogled za pogrešku.
   - pogled 6.2:
     - pregled svih korisnika (straničenje, Ajax). Šalje zahtjev GET metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „korisnici". Uspoređuje dobivene korisnike s onima u lokalnoj bazi podataka na osnovi korisničkog imena. Kod korisnika koji se razlikuju stavlja vidljivu oznaku u pregledu. Postoji gumb „Sinkroniziraj" kojim se pokreće upis korisnika koji nedostaju u lokalnu bazu podataka i osvježava se pregled korisnika.

- pogled 6.3:
    - pregled svih primljenih JMS poruka (straničenje, Ajax) iz reda čekanja NWTiS_{korisnicko_ime}. Postoji gumb „Obriši“ kojim se pokreće brisanje svih primljenih JMS poruka i osvježava se pregled JMS poruka.
- pogled 6.4:
    - pregled aerodroma za preuzimanje i pregled svih aerodroma (Ajax). Kod pregleda svih aerodroma inicijalno se ne prikazuje ni jedan aerodrom. Postoji element za unos naziva koji služi kao filtar temeljem kojeg se prikazuju aerodromi koji zadovoljavaju uvjet. Filtriranje se pokreće gumbom. Odabrani aerodrom može se dodati za preuzimanje aktiviranjem gumba. Pri tome se šalje zahtjev POST metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „aerodromi“. Osvježava se pregled aerodroma za preuzimanje.
- pogled 6.5:
    - pregled vlastitih putovanja (Ajax) uz prikaz vremena prvog leta, naziv aerodroma polaska prvog leta, naziv aerodroma dolaska zadnjeg leta. Odabirom jednog putovanja prikazuju se kronološki podaci o letovima koji su sadržani u putovanju. Za svaki let prikazuje se ICAO i naziv aerodroma s kojeg se poletjelo, vrijeme kada se poletjelo u formatu dd.mm.gggg hh:mm:ss, ICAO i naziv aerodroma na koji se sletjelo, vrijeme kada se poletjelo u formatu dd.mm.gggg hh:mm:ss, trajanje leta u formatu hh:mm:ss, udaljenost između aerodroma. Za informaciju o aerodromu šalje se zahtjev GET metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „aerodromi“. Za informaciju o udaljenosti između dva aerodroma šalje se zahtjev GET metodom RESTful web servisu iz {korisnicko_ime}-aplikacija_3 na resursu „aerodromi“. Ispod prikaza letova putovanja prikazuje se ukupno vrijeme putovanja u formatu hh:mm:ss (vrijeme kada se sletjelo u zadnjem letu – vrijeme kada se poletjelo u prvom letu) i ukupna udaljenost svih letova.
- pogled 6.6:
    - plan novog putovanja (Ajax). Na vrhu ekrana prikazuje se broj aerodroma koji se preuzimaju, a osvježava se temeljem WebSocket poruka iz {korisnicko_ime}-aplikacija_5 na krajnjoj točki „info“. Postoje elementi za unos početnog i završnog vremena u formatu dd.mm.gggg hh:mm:ss. Prikazuju se aerodromi za preuzimanje. Odabirom jednog aerodroma i izvršavanjem gumba kronološki se prikazuju polasci aviona s tog aerodroma kojima je odredište jedan od aerodroma koji se preuzimaju i čiji je polazak unutar zadanog intervala. Gumbom se odabrani let dodaje kao let u putovanju i prikazuje se u kronološkom pregledu letova putovanja. Nakon toga se osvježava prikaz polazaka aviona s time da se kao aerodrom polaska uzima aerodrom na koji se sletjelo u prethodnom letu, a početak intervala pomiče se na vrijeme kada se sletjelo u prethodnom letu. Postupak se nastavlja dok se ne izvrši gumb završetak putovanja. Tada se podaci upisuju u bazu podataka. Postoje gumbi za brisanje zadnjeg leta i za brisanje svih letova u putovanju. Osvježava se pregled letova za putovanje.

Struktura projekta treba biti sljedeća:

```
{LDAP_korisnik}-projekt
      pom.xml
      {LDAP_korisnik}-obrazac_1.pdf  - popunjeni obrazac za projekt
      {LDAP_korisnik}-obrazac_2.pdf  - popunjeni obrazac za samoprocjenu projekta
      {LDAP_korisnik}-aplikacija_1
            pom.xml
            src
                  main
                        ...
      {LDAP_korisnik}-aplikacija_2
            pom.xml
            src
                  main
                        ...
      {LDAP_korisnik}-aplikacija_3
            pom.xml
            src
                  main
                        ...
      {LDAP_korisnik}-aplikacija_4
            pom.xml
            src
                  main
                        ...
      {LDAP_korisnik}-aplikacija_5
            pom.xml
            src
                  main
                        ...

      {LDAP_korisnik}-aplikacija_6
            pom.xml
            src
                  main
                        ...
            {LDAP_korisnik}-aplikacija_6_jpa
                  pom.xml
                  src
                        main
                              ...
            {LDAP_korisnik}-aplikacija_6_wa
                  pom.xml
                  src
                        main
                              ...
```

Korijenski projekt treba sadržati ostale projekte kao module tako da se može pokrenuti priprema svih projekata na najvišoj razini.

Instalacijska i programska arhitektura sustava

Aplikacija {korisnicko_ime} _aplikacija_	**Razvojni alat** (IDE) kod obrane projekta	**Java**	**Poslužitelj**	**EE osobine**	**Korisničko sučelje**	**Baza podataka**	**Rad s bazom podataka**	**Namjena**
1	Eclipse s Maven	17						Poslužitelj na mrežnoj utičnici (socket server)
2	Eclipse s Maven	17	Payara Web	Jakarta EE9.1 Web		MySQL nwtis_bp_1	JDBC, SQL	Preuzima podatke o polascima i dolascima aviona s izabranih aerodroma
3	Eclipse s Maven	17	Payara Web	Jakarta EE9.1 Web		MySQL nwtis_bp_1	JDBC, SQL	RESTful/JAX-RS web servis
4	Eclipse s Maven	17	Payara Web	Jakarta EE9.1 Web	Jakarta MVC			Pogledi za administraciju
5	Eclipse s Maven	17	Glassfish EE Server	Jakarta EE9.1				JAX-WS web servis WebSocket krajnja točka
6	Eclipse s Maven	17	Glassfish EE Server	Jakarta EE9.1	JSF / PrimeFaces Ajax	HSQLDB nwtis_bp_2	JPA Criteria API	Pogledi za rad s korisnicima, JMS porukama, aerodromima za preuzimanje, putovanja i izradu putovanja