Univerzitet “Džemal Bijedić” Mostar

Fakultet informacijskih tehnologija

UI Agent – Task agent sistem (Agent za upravljanje zadacima)

Predmet – **Umjetna inteligencija**

Akademska 2025/26. godina

Predmetni nastavnik: prof.dr. Nina Bijedić

Student: Esmir Hero (IB220119)

Sadržaj

Uvod

U okviru seminarskog rada implementiran je softverski inteligentni agent za upravljanje zadacima (Task Agent). Cilj rada nije bio razviti klasičnu CRUD aplikaciju, već agentički sistem koji autonomno obrađuje zadatke kroz iterativni ciklus Sense → Think → Act → Learn, uz mogućnost ograničene ljudske intervencije (human-in-the-loop). Agent radi u pozadini kroz kontinuirani *runner loop* i samostalno donosi odluke o obradi zadataka, dok je web aplikacija ograničena na ulogu transportnog i prezentacijskog sloja.

Diskusija ideje

Problem koji se rješava

U klasičnim task aplikacijama:

- zadaci se obrađuju ručno,
- nema autonomnog ponašanja,
- sistem ne reaguje samostalno na promjene stanja.

Problem koji ovaj agent rješava je:

kako modelirati zadatke kao entitete koje agent samostalno prati, procjenjuje i obrađuje kroz vrijeme, a ne samo prikazuje korisniku.

Da li je ovo agent ili analitička aplikacija?

Sistem nije analitička aplikacija jer:

- ne reaguje jednokratno na korisnički input,
- ne izvršava logiku u okviru HTTP requesta,
- ne vraća rezultat odmah nakon unosa.

Sistem jeste agent jer:

- ima vlastiti *runner* koji radi kontinuirano,
- posmatra stanje sistema (taskovi u bazi),
- donosi odluke u iteracijama,
- izvršava akcije i mijenja stanje svijeta,
- uči iz interakcija korisnika.

Šta se dešava kroz vrijeme?

Kroz vrijeme, agent periodično preuzima taskove, provjerava njihovo stanje i rokove, preskače završene / snoozed taskove, reaguje na korisničke namjere (complete, snooze, reject), prilagođava svoje ponašanje (learning). Ovo vremensko ponašanje je ključni dokaz da sistem nije request/response aplikacija.

Vrsta agenta

Implementirani agent se može klasifikovati kao:

- Goal-based agent
  cilj: pravilno i pravovremeno procesiranje taskova
- Context-aware agent
  ponašanje zavisi od stanja taska, vremena i user intent-a
- Learning agent (jednostavan)
  adaptira pragove i prioritete na osnovu korisničkih intervencija
- Human-in-the-loop agent
  korisnik može slati namjere, ali agent odlučuje

Ova kombinacija je izabrana jer realni sistemi rijetko funkcionišu potpuno autonomno.

Agent ciklus (Sense → Think → Act → Learn)

Sense (percepcija)

Agent opaža:

- taskove u bazi podataka,
- njihov status (Pending, Snoozed, Completed, Rejected),
- vremenske atribute (NextRunAt),
- korisničke namjere (Complete/Snooze/Reject requests).

Think (odlučivanje)

Agent odlučuje:

- da li postoji task za obradu u ovom tick-u,
- da li je task aktivan ili se treba preskočiti,
- kako primijeniti korisničku namjeru,
- da li je dozvoljena tranzicija stanja.

Odluke se donose kroz pravila i heuristike, ne kroz UI.

Act (akcija)

Agent izvršava:

- ažuriranje statusa taska,
- zakazivanje narednog izvršavanja (snooze),
- uklanjanje taska iz daljnje obrade (complete / reject),
- upis promjena u bazu podataka.

Learn (učenje)

Learning komponenta:

- prati koliko često korisnik:
    - završava taskove ručno,
    - odbija taskove,
    - snooze-uje taskove,
- na osnovu toga prilagođava:
    - prioritete,
    - vremenske pragove,
    - redoslijed obrade.

Učenje je namjerno jednostavno, ali jasno integrisano u agent ciklus.

Human-in-the-loop interakcija

Korisnik može:

- označiti task kao Completed,
- Snooze-ovati task,
- Reject-ovati task.

Važno:

- korisnik ne mijenja stanje direktno,
- šalje se namjera (intent),
- agent validira i primjenjuje promjenu.

Ovakav model zadržava autonomiju agenta i sprječava CRUD logiku u UI-ju.

Ideje za proširenje (diskusija)

Moguća proširenja:

- objašnjenje odluka agenta (zašto je task preskočen),
- adaptivni snooze intervali,
- više agenata (npr. PrioritizationAgent),
- simulacija rada agenta nad istorijskim podacima.

Nije sve implementirano, ali je jasno razmotreno.

Uloga LLM-a u razvoju

LLM je korišten:

- za diskusiju ideje i validaciju da li je sistem agent,
- za poređenje arhitektonskih opcija,
- za generisanje koda prema specifikaciji,
- za code review i refaktor,
- za provjeru agent ciklusa.
  Razvoj je rađen iterativno, sa više povratnih petlji.

Zaključak

Implementirani Task Agent predstavlja:

- autonomni softverski agent,
- sa jasnim agent ciklusom,
- čistom arhitekturom,
- i human-in-the-loop interakcijom.

Sistem ispunjava sve zahtjeve predmeta i jasno se razlikuje od klasične CRUD aplikacije.

SPECIFIKACIJA AGENTA

Cilj Agenta

Primarni cilj:
Maksimizirati propusnost završenih zadataka uz održavanje zdravlja sistema i minimiziranje zakasnjelih zadataka.

Podciljevi:

- Optimizacija hitnosti: Dati prednost zadacima visoke hitnosti u odnosu na zadatke niske hitnosti.
- Upravljanje kapacitetom: Balansirati aktivno opterećenje zadacima kako bi se spriječilo preopterećenje sistema.
- Poštivanje rokova: Minimizirati broj zakasnjelih zadataka kroz proaktivnu eskalaciju.
- Efikasnost resursa: Optimizirati sistemske parametre (kapacitet, praga) na osnovu povratnih informacija o performansama.

Formalizacija cilja:

Maksimizirati: $\eta = (\text{Završeni\_zadaci} / \text{Ukupno\_vrijeme}) \times \text{Faktor\_kvalitete}$

Gdje je:
$\text{Faktor\_kvalitete} = f(\text{stopa\_zakasnjenja}, \text{stopa\_eskalacije}, \text{iskorištenje})$

Pod ograničenjima:
$\text{Aktivni\_zadaci} \leq \text{MaksAktivniZadaci}$

Zakasnjeli_zadaci → min
Iskorištenje_sistema ∈ [0.5, 0.9] (izbjeći podopterećenje i preopterećenje)

Percepti (Senzorni ulazi)

Agent posmatra okolinu kroz strukturiranu percepciju:

TaskPercept (Primarna percepcija):

```
TaskPercept = {
    Tasks: Collection<TaskItem>,
    Settings: SystemSettings,
    Timestamp: DateTimeOffset
}

TaskItem = {
    Id: Guid,
    Title: String,
    Status: {Pending, Active, Snoozed, Escalated, Completed,
Rejected},
    Priority: {Low, Medium, High, Critical},
    DueDate: DateTimeOffset?,
    CreatedAt: DateTimeOffset,
    UpdatedAt: DateTimeOffset,
    SnoozedUntil: DateTimeOffset?,
    EscalationCount: Integer
}

SystemSettings = {
    MaxActiveTasks: Integer ∈ [1, 100],
    EscalationThresholdHours: Integer ∈ [1, 168],
    MinimumConfidenceThreshold: Real ∈ [0.0, 1.0],
    DefaultSnoozeDuration: TimeSpan,
    RecommendationValidityDuration: TimeSpan,
    AutoApplyRecommendations: Boolean,
    AutoEscalateOverdueTasks: Boolean,
    AutoAwakenSnoozedTasks: Boolean
}
```

Filtriranje percepcije:

- Terminalni zadaci (Status = Completed ∨ Status = Rejected) su isključeni.
- Posmatraju se samo zadaci u {Pending, Active, Snoozed, Escalated}.

Akcije (Aktuatori)

Agent može izvršiti sljedeće tranzicije stanja:

Akcije pokrenute od strane Agenta:

Akcija	Preuslov	Efekt	Domena Metoda
Activate	Status ∈ {Pending, Snoozed} ∧ Active_count < MaxActiveTasks	Status → Active, SnoozedUntil → null	task.Activate()
Snooze	Status ∈ {Pending, Active}	Status → Snoozed, SnoozedUntil → UtcNow + duration	task.Snooze(until)
Escalate	Status ≠ Escalated ∧ (overdue ≥ threshold ∨ critical_condition)	Status → Escalated, Priority → Critical, EscalationCount++	task.Escalate()
ReturnToPending	Status ∈ {Active, Snoozed, Escalated}	Status → Pending, SnoozedUntil → null	task.ReturnToPending()
Complete	Status ≠ Completed	Status → Completed, CompletedAt → UtcNow	task.Complete()

Akcije pokrenute od strane Korisnika (Intencije):

Akcija	Preuslov	Efekt	Domena Metoda
CompleteByUser	Status ∉ {Completed, Rejected}	Status → Completed (terminal)	task.CompleteByUser()
SnoozeByUser	Status ∉ {Completed, Rejected}	Status → Snoozed, SnoozedUntil → specified	task.SnoozeByUser(until)
RejectByUser	Status ∉ {Completed, Rejected}	Status → Rejected (terminal)	task.RejectByUser()

Politika odabira akcije:
Agent odabira najviše JEDNU akciju po ciklusu na osnovu zadatka najveće hitnosti i primjenjivih pravila politike.

Signali za Učenje

Evidentiranje iskustva (nakon svakog ciklusa):

```
TaskScoringExperience = {
    TasksProcessed: Integer,
    TasksScored: Integer,
    ActionsExecuted: Integer,
    SuccessfulActions: Integer,
    AverageUrgencyScore: Real ∈ [0.0, 1.0],
    MaxUrgencyScore: Real ∈ [0.0, 1.0],
    PerformanceSummary: String
}
```

Metrike performansi (korištene za adaptaciju):

```
Metrics = {
    AverageUrgency: Mean(UrgencyScore),
    MaxUrgency: Max(UrgencyScore),
    SuccessRate: SuccessfulActions / ActionsExecuted,
    TaskLoad: Mean(TasksProcessed),
    ActionRate: Mean(ActionsExecuted)
}
```

Okidači za adaptaciju:

- AKO AverageUrgency > 0.7 ∧ Iskorištenje ≥ 90% ONDA PovećajKapacitet
- AKO AverageUrgency < 0.3 ∧ Iskorištenje ≤ 50% ONDA SmanjiKapacitet
- AKO BrojZakasnjelih > 0.5 × MaksZadataka ONDA SmanjiPragEskalacije
- AKO ActionRate < 0.2 ONDA PovećajPragPouzdanosti

Ograničenja

Čvrsta ograničenja:

- Atomičnost ciklusa: Svaki StepAsync() izvršava tačno jedan Sense-Think-Act-Learn ciklus.
- Bez internih kašnjenja: Runner NE SME sadržavati Task.Delay() ili blokirajuće operacije.
- Idempotentnost: Ponovljeni pozivi sa istim stanjem moraju proizvesti determinističke rezultate.
- Valjanost stanja: Sve tranzicije stanja moraju zadovoljiti invarijante domene.

- Konačnost terminalnog stanja: Zadaci u {Completed, Rejected} su nepromjenjivi i isključeni iz procesiranja agenta.

Meka ograničenja:

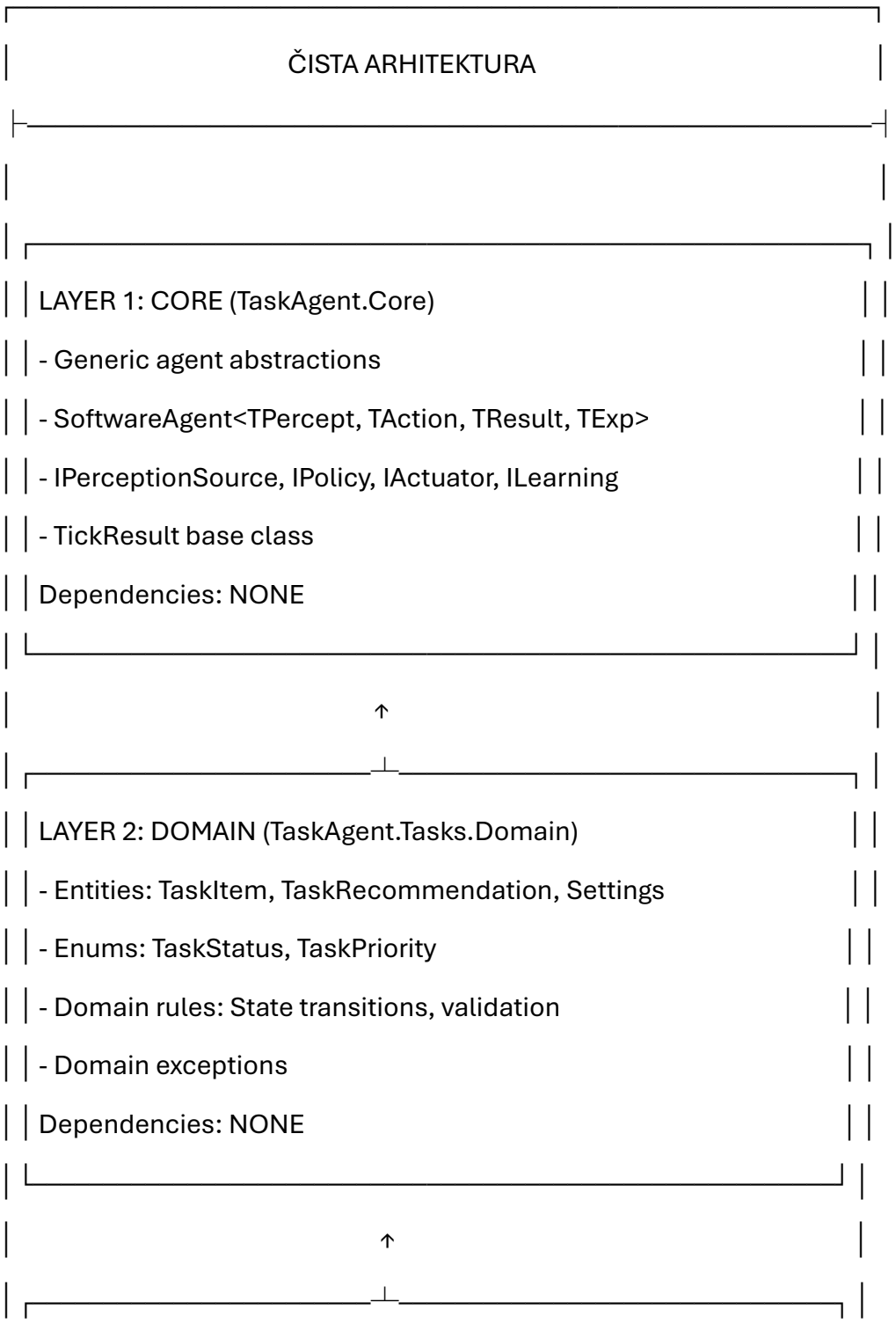
- Granice kapaciteta: MaksAktivniZadaci ∈ [1, 100]
- Granice praga: EscalationThresholdHours ∈ [1, 168]
- Granice pouzdanosti: MinimumConfidenceThreshold ∈ [0.0, 1.0]
- Frekvencija ciklusa: Scoring agent: ~5 sekundi, Adaptation agent: ~5 minuta (prisiljeno od strane hosta)

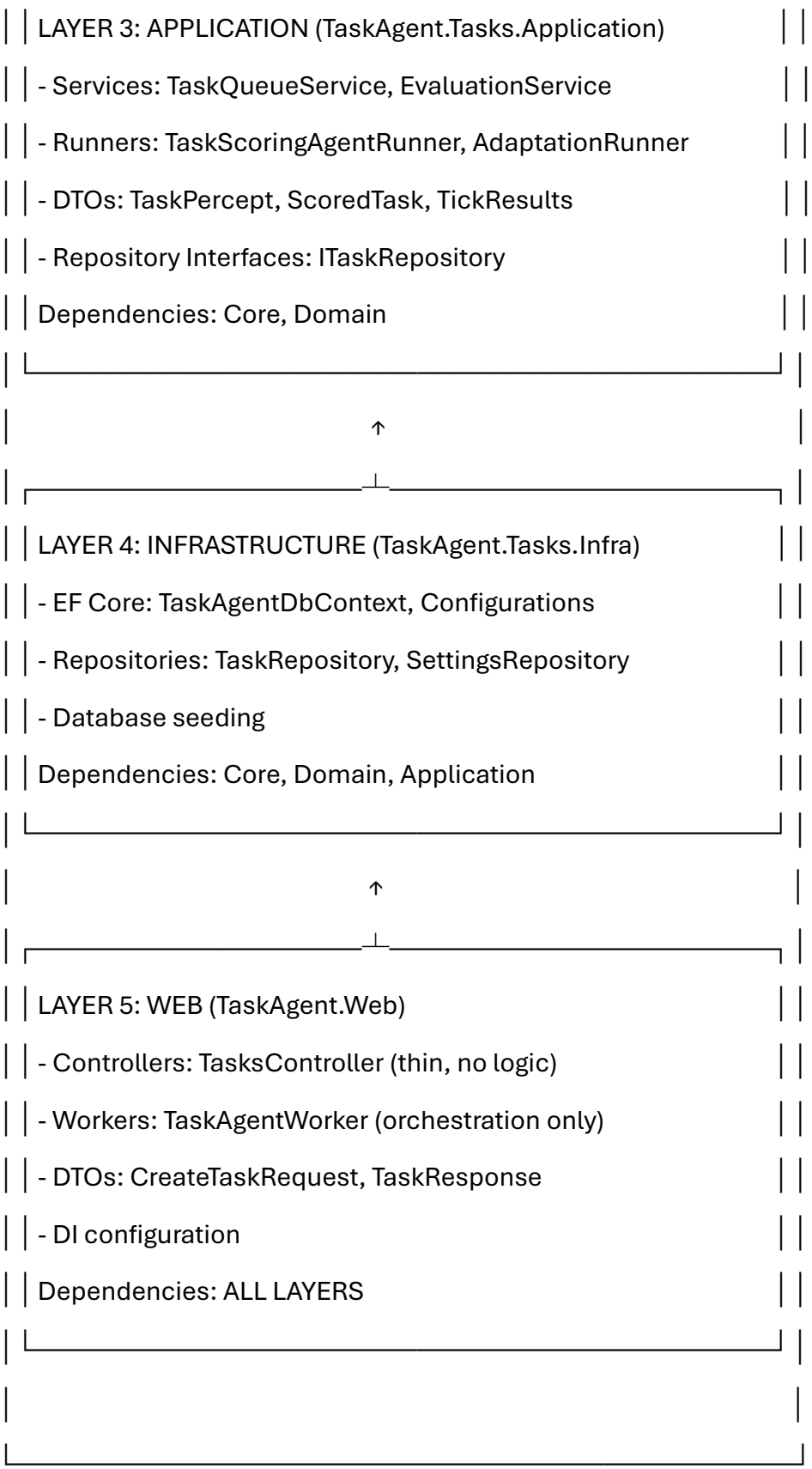
Arhitektonska ograničenja:

- Izolacija slojeva: Domena → Aplikacija → Infrastruktura (jednosmjerno)
- Autoritet odlučivanja: Agent upravlja životnim ciklusom; Korisnik donosi terminalne odluke.
- Tanak Web sloj: Bez poslovne logike u Controllerima.
- Čistoća Runnera: Bez infrastrukturnih briga (HTTP, SignalR, kašnjenja) unutar runnera.

SISTEMSKA ARHITEKTURA

Čista Arhitektura Slojeva





Pravilo zavisnosti: Zavisnosti pokazuju samo UNUTRA. Unutrašnji slojevi nikada ne zavise od spoljašnjih slojeva.

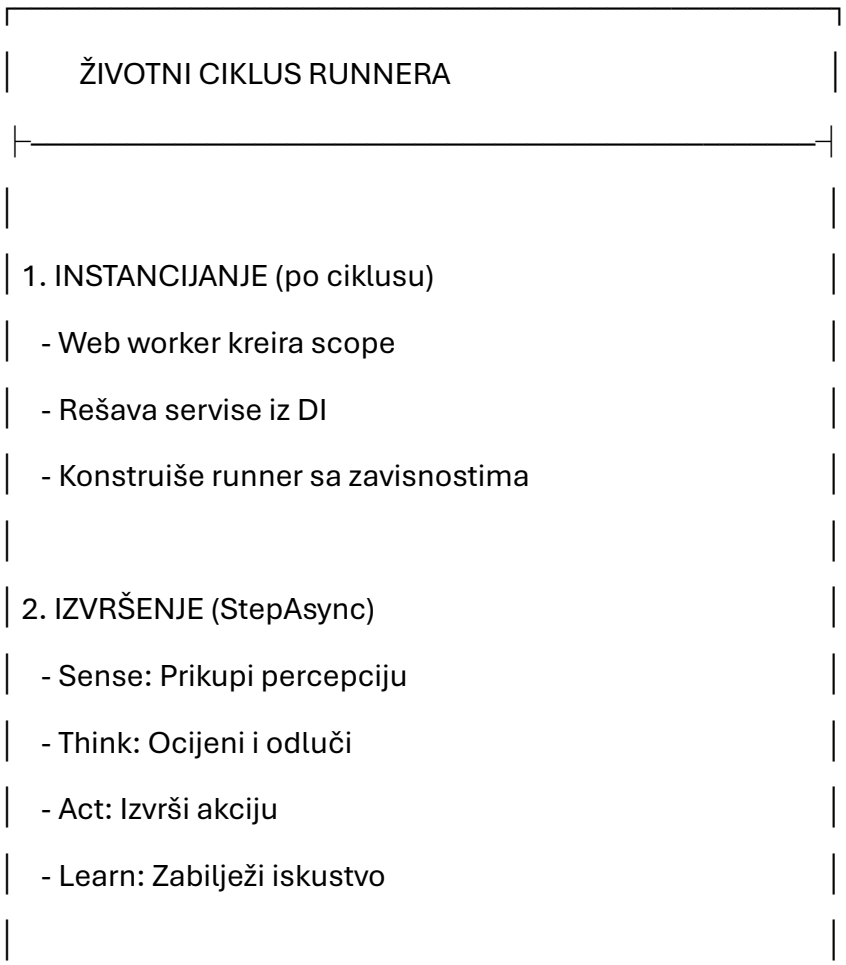
Uloga Agent Runnera

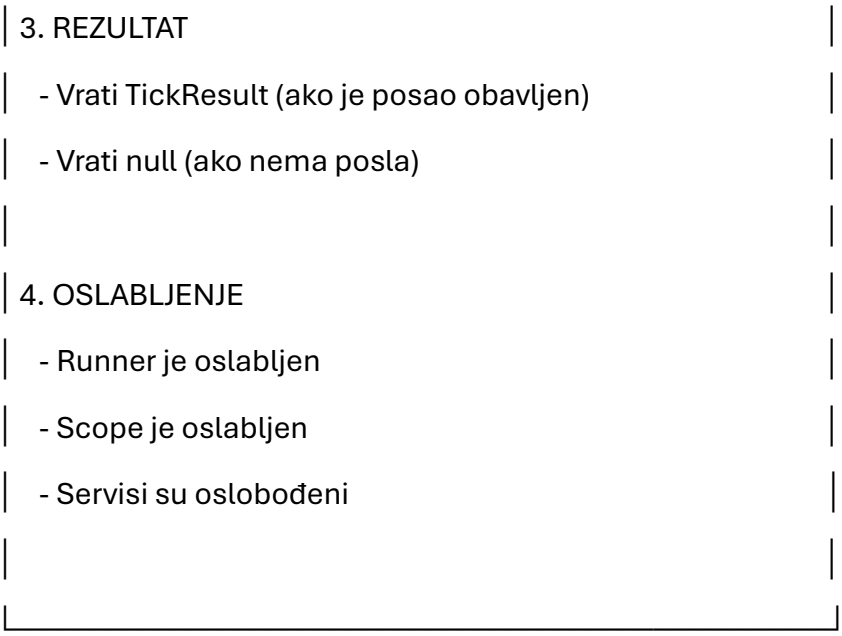
Definicija:
Runner je komponenta na Aplikacijskom sloju koja orkestrira Sense-Think-Act-Learn ciklus komponovanjem domen servisa i apstrakcija agenta.

Ključne karakteristike:

- Orkestracija bez stanja: Svaki StepAsync() je nezavisan.
- Kompozicija komponenti: Ubrizgava IPerceptionSource, IPolicy, IActuator, ILearningComponent.
- Izvršenje ciklusa: Vraća TickResult? (null ako nije obavljen posao).
- Bez infrastrukture: Bez HTTP, baze podataka, kašnjenja ili I/O operacija.
- Čista poslovna logika: Delegira sve odluke servisima i domeni.

Životni ciklus Runnera:





Web kao Tanak Host

Odgovornosti Web Sloja:

- HTTP Transport
    - Prihvata zahtjeve.
    - Usmjerava do kontrolera.
    - Vraća odgovore.
- Pozadinska orkestracija
    - Pokreće/zaustavlja pozadinske workere.
    - Upravlja vremenom ciklusa (kašnjenje između ciklusa).
    - Kreira DI scope po ciklusu.
- DTO Mapiranje
    - Mapira domen entitete na DTO-e.
    - Parsira request DTO-e u domenske tipove.
- Unakrsne brige
    - Logovanje.
    - CORS.
    - Autentifikacija/Autorizacija.

Šta Web Sloj NE RADI:

- Poslovna pravila.
- Logika tranzicije stanja.
- Ocenjivanje hitnosti.
- Odluke politike.
- Logika adaptacije.

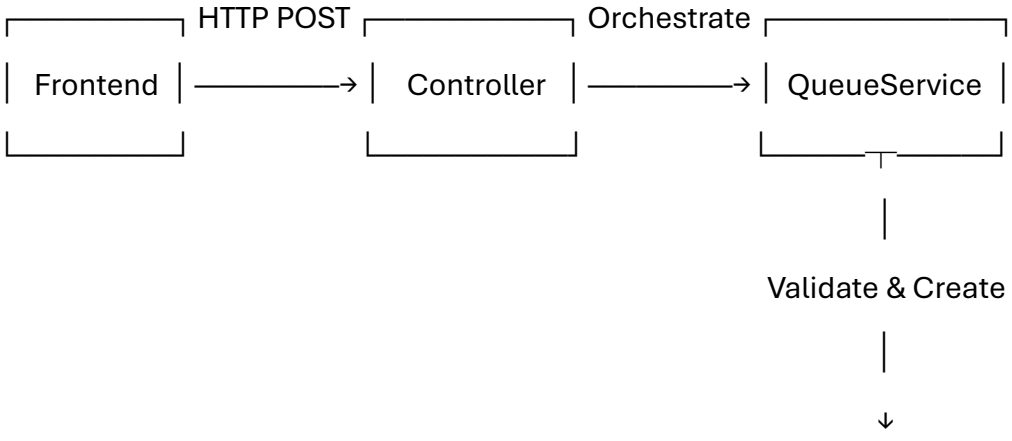
Primjer: TasksController (Tanak)

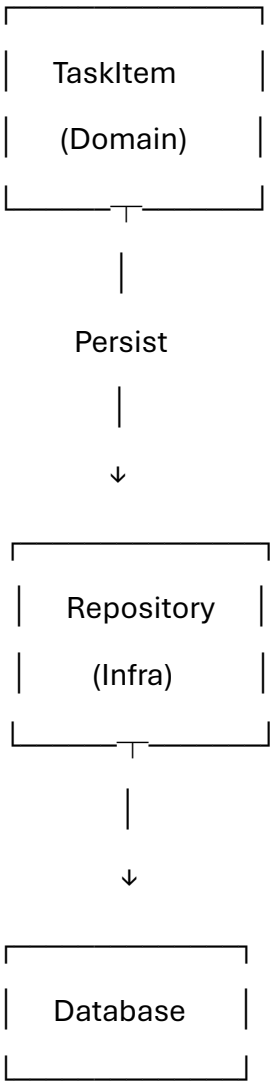
```
[HttpPost("{id}/complete")]
public async Task<IActionResult> CompleteTask(Guid id)
{
    // 1. Prihvati zahtjev (transport)
    // 2. Delegiraj servisu (aplikacijski sloj odlučuje)
    var success = await
_userActionService.RequestCompleteTaskAsync(id);
    // 3. Mapiraj rezultat u HTTP odgovor (transport)
    return success ? Ok(new { message = "..." }) : NotFound();
}
```

Nema logike! Samo orkestracija i transport.

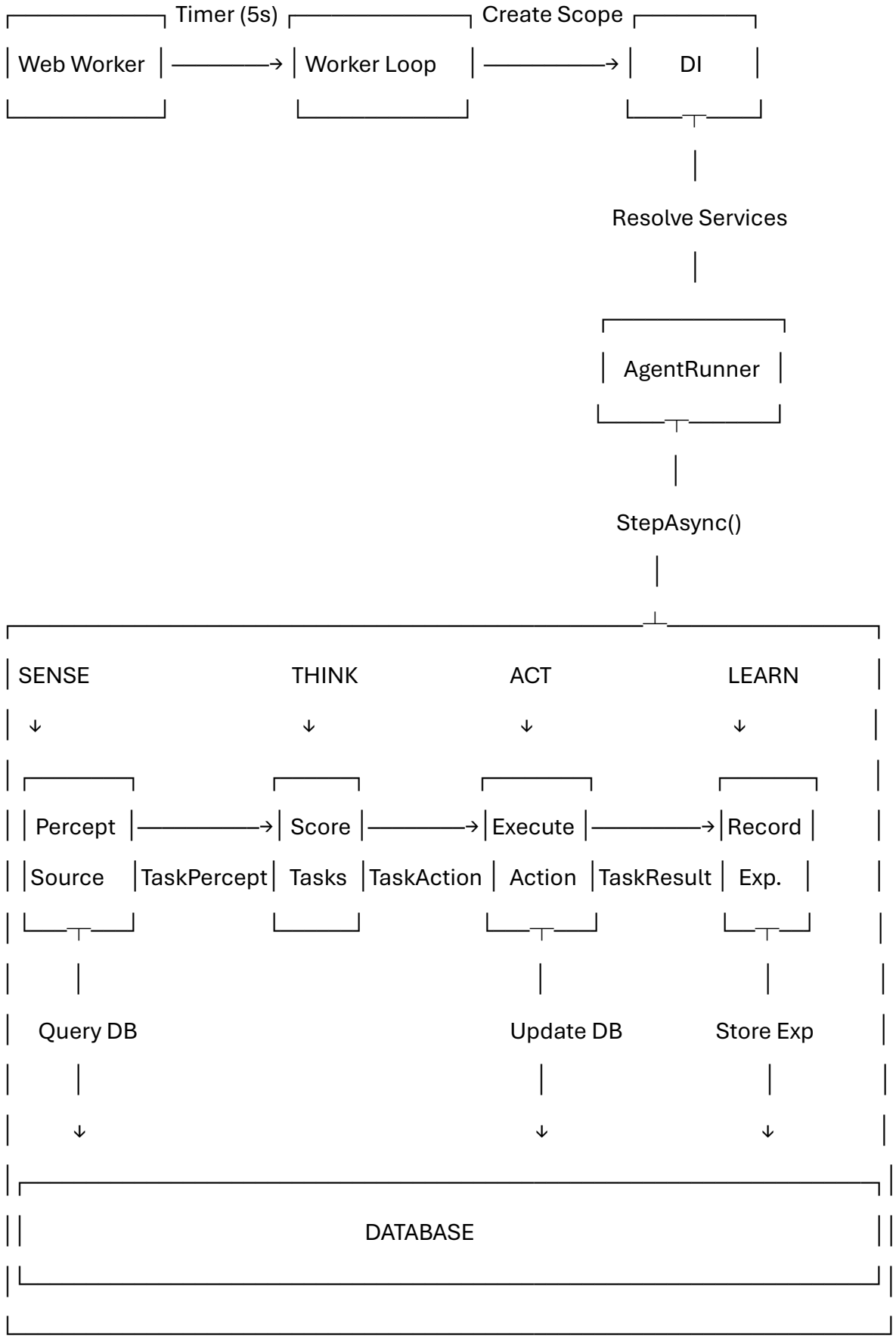
2.4 Protok Podataka Između Slojeva

Protok 1: Korisnik Kreira Zadatak (Zahtjev/Odgovor)

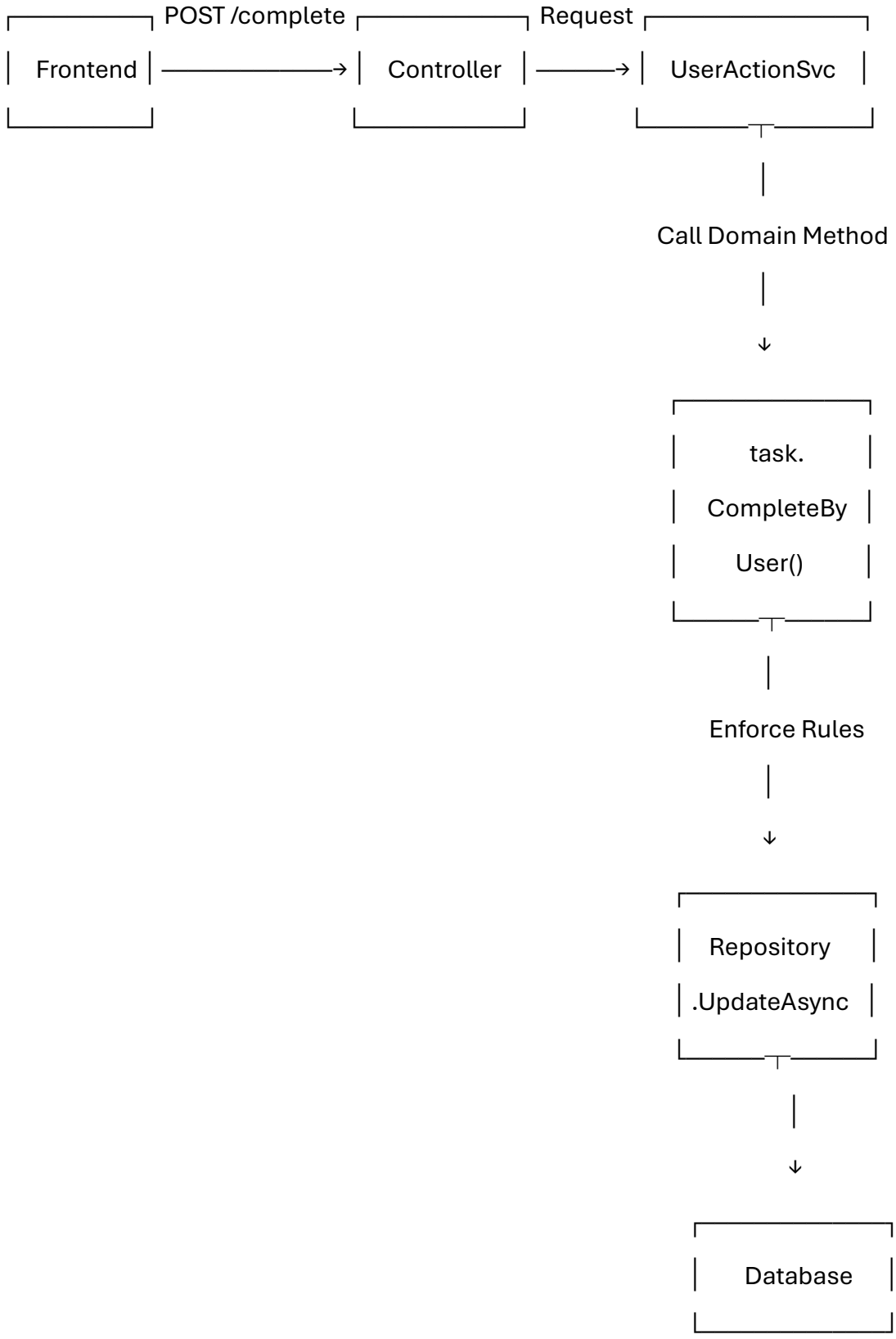



TaskItem
(Domain)
Persist
Repository
(Infra)
Database

Protok 2: Agent Procesira Zadataka (Autonomno)



Protok 3: Korisnička Akcija – Intencija (Komanda)



Ključna zapažanja:

1. Agenti i Korisnici nikada ne komuniciraju direktno.
2. Sve interakcije su posredovane bazom podataka.
3. Web sloj obezbjeđuje vremenovanje, ne logiku.
4. Domen primjenjuje SVA poslovna pravila.

PSEUDOKOD

Agent Runner StepAsync()

Generički Agent Pattern:

```
ALGORITHM: SoftwareAgent.StepAsync()
INPUT: None (stateless)
OUTPUT: TickResult<TPercept, TAction, TResult, TExperience> |
null

BEGIN

    percept ← SenseAsync() // FAZA 1: SENSE

    action ← ThinkAsync(percept) // FAZA 2: THINK

    result ← ActAsync(action) // FAZA 3: ACT

    experience ← LearnAsync(percept, action, result) // FAZA 4:
LEARN

    // Provjeri scenarij bez posla

    IF percept = null AND action = null AND result = null AND
experience = null THEN

        RETURN null // Nije obavljen posao

    END IF

    // Kreiraj rezultat ciklusa

    tick_result ← CreateTickResult(percept, action, result,
experience)

    RETURN tick_result

END
```

Task Scoring Agent Runner

Konkretna Implementacija:

```
ALGORITHM: TaskScoringAgentRunner.StepAsync()
OUTPUT: TaskScoringTickResult | null
BEGIN
    // ==================== SENSE FAZA ====================
    settings ← SettingsRepository.EnsureExistsAsync()
    tasks ← TaskRepository.GetAllAsync()

    // Filtriraj terminalne zadatke
    live_tasks ← tasks.Where(t → t.Status ≠ Completed AND
t.Status ≠ Rejected)

    IF live_tasks.IsEmpty() THEN
        RETURN null // Nema zadataka za procesiranje
    END IF
    percept ← TaskPercept {
        Tasks: live_tasks,
        Settings: settings,
        Timestamp: UtcNow
    }

    // ==================== THINK FAZA ====================
    scored_tasks ← []
    FOR EACH task IN percept.Tasks DO
        // Izračunaj komponente hitnosti
        priority_weight ← MAP priority TO [0.25, 1.0]
        time_weight ← CalculateTimeWeight(task.DueDate)
        status_weight ← MAP status TO [0.0, 1.0]
```

```
        // Funkcija korisnosti
        urgency_score ← (priority_weight × 0.40)
                      + (time_weight × 0.35)
                      + (status_weight × 0.25)

        scored_tasks.Add(ScoredTask {
            Task: task,
            UrgencyScore: urgency_score,
            ShouldEscalate: IsOverdue(task) AND OverdueBy(task)
≥ EscalationThreshold,
            ShouldAwaken: task.Status = Snoozed AND UtcNow ≥
task.SnoozedUntil
        })
    END FOR

    // Sortiraj po hitnosti (opadajuće)
    scored_tasks ← Sort(scored_tasks, BY urgency_score DESC)

    // Odaberi zadatak najveće hitnosti
    urgent_task ← scored_tasks.First()

    // Primjeni pravila politike
    action ← null
    IF urgent_task.ShouldEscalate THEN
        action ← TaskAction { Type: "Escalate", Task:
urgent_task.Task }
    ELSE IF urgent_task.ShouldAwaken THEN
        action ← TaskAction { Type: "Awaken", Task:
urgent_task.Task }
```

```
    ELSE IF urgent_task.UrgencyScore ≥ 0.7 AND ActiveCount <
MaxActiveTasks THEN
        action ← TaskAction { Type: "Activate", Task:
urgent_task.Task }
    ELSE IF urgent_task.UrgencyScore < 0.3 AND ActiveCount ≥ 0.8
× MaxActiveTasks THEN
        action ← TaskAction { Type: "Snooze", Task:
urgent_task.Task }
    END IF

    // ==================== ACT FAZA ====================
    result ← null
    IF action ≠ null THEN
        success ← EXECUTE action.Type ON action.Task
        result ← TaskActionResult { Action: action, Success:
success }
    END IF

    // ==================== LEARN FAZA ====================
    experience ← TaskScoringExperience {
        TasksProcessed: percept.Tasks.Count,
        TasksScored: scored_tasks.Count,
        ActionsExecuted: action ≠ null ? 1 : 0,
        SuccessfulActions: result.Success ? 1 : 0,
        AverageUrgencyScore: Mean(scored_tasks.UrgencyScore),
        MaxUrgencyScore: Max(scored_tasks.UrgencyScore)
    }

    // Sačuvaj iskustvo za adaptaciju
    ExperienceHistory.Add(experience)
```

```
    // ==================== RETURN REZULTAT ====================
    RETURN TaskScoringTickResult {
        Percept: percept,
        Action: action,
        Result: result,
        Experience: experience,
        ScoredTasks: scored_tasks
    }
END
```

Adaptation Agent Runner

Meta-Učenje Implementacija:

```
ALGORITHM: TaskAdaptationAgentRunner.StepAsync()
OUTPUT: AdaptationTickResult | null
BEGIN
    // ==================== SENSE FAZA ====================
    recent_experiences ← ExperienceHistory.TakeLast(100)
    IF recent_experiences.Count < 5 THEN
        RETURN null // Nedovoljno podataka
    END IF

    current_settings ← SettingsRepository.GetAsync()
    statistics ← GatherStatistics() // Active, Pending, Overdue
counts

    percept ← AdaptationPercept {
        RecentExperiences: recent_experiences,
        CurrentSettings: current_settings,
        Statistics: statistics
```

```
    }


    // ==================== THINK FAZA ====================
    metrics ← {
        AverageUrgency:
Mean(recent_experiences.AverageUrgencyScore),
        SuccessRate: Mean(recent_experiences.SuccessRate),
        TaskLoad: Mean(recent_experiences.TasksProcessed)
    }


    trend ← DeterminePerformanceTrend(recent_experiences)
    action ← null
    adapted_settings ← current_settings.Clone()


    // Pravila adaptacije
    IF metrics.AverageUrgency > 0.7 AND statistics.ActiveCount ≥
0.9 × MaxActiveTasks THEN
        // Sistem preopterećen sa visokom hitnošću → Povećaj
kapacitet
        adapted_settings.MaxActiveTasks ← Min(MaxActiveTasks +
5, 100)
        action ← AdaptationAction {
            Type: "IncreaseCapacity",
            UpdatedSettings: adapted_settings,
            Reasoning: "High urgency at capacity",
            Confidence: 0.80
        }
    ELSE IF metrics.AverageUrgency < 0.3 AND
statistics.ActiveCount ≤ 0.5 × MaxActiveTasks THEN
        // Sistem nedovoljno iskorišten → Smanji kapacitet
```

```
        adapted_settings.MaxActiveTasks ← Max(MaxActiveTasks -
3, 1)
        action ← AdaptationAction {
            Type: "DecreaseCapacity",
            UpdatedSettings: adapted_settings,
            Reasoning: "Low urgency underutilized",
            Confidence: 0.70
        }
    ELSE IF statistics.OverdueCount > 0.5 × MaxActiveTasks THEN
        // Previše zakasnjelih → Eskaliraj ranije
        adapted_settings.EscalationThresholdHours ←
Max(EscalationThreshold - 6, 1)
        action ← AdaptationAction {
            Type: "ReduceEscalationThreshold",
            UpdatedSettings: adapted_settings,
            Reasoning: "High overdue count",
            Confidence: 0.85
        }
    END IF

    // ==================== ACT FAZA ====================
    result ← null
    IF action ≠ null THEN
        TRY
            adapted_settings.ValidateConsistency()
            SettingsRepository.SaveAsync(adapted_settings)
            result ← AdaptationResult {
                Action: action,
                Success: true,
```

```
                PreviousSettings: current_settings,
                NewSettings: adapted_settings
            }
        CATCH exception
            result ← AdaptationResult {
                Action: action,
                Success: false,
                ErrorMessage: exception.Message
            }
        END TRY
    END IF

    // ==================== LEARN FAZA ====================
    experience ← AdaptationExperience {
        ExperiencesAnalyzed: recent_experiences.Count,
        AdaptationTriggered: action ≠ null,
        PerformanceTrend: trend,
        TriggerMetrics: metrics
    }

    // ==================== RETURN REZULTAT ====================
    RETURN AdaptationTickResult {
        Percept: percept,
        Action: action,
        Result: result,
        Experience: experience,
        SettingsChanged: result.Success
    }
END
```

Web Worker Orkestracija

Vremenovanje na nivou Host-a (NE u Runneru):

```
ALGORITHM: TaskAgentWorker.ExecuteAsync()
INPUT: CancellationToken stoppingToken
OUTPUT: void

BEGIN
    LogInformation("Agent worker started")
    // Pokreni dvije petlje agenata istovremeno
    PARALLEL DO
        RunScoringAgentLoop(stoppingToken)
        RunAdaptationAgentLoop(stoppingToken)
    END PARALLEL
END

ALGORITHM: RunScoringAgentLoop(stoppingToken)
BEGIN
    WHILE NOT stoppingToken.IsCancellationRequested DO
        TRY
            // === ODGOVORNOST HOST-A: Kreiraj Scope ===
            scope ← ServiceProvider.CreateScope()
            // === ODGOVORNOST HOST-A: Rešavi Servise ===
            evaluationService ←
scope.Resolve<TaskEvaluationService>()
            recommendationService ←
scope.Resolve<RecommendationService>()
            queueService ← scope.Resolve<TaskQueueService>()
            learningService ← scope.Resolve<LearningService>()
```

```
            // === ODGOVORNOST HOST-A: Instanciraj Runner ===
            runner ← NEW TaskScoringAgentRunner(
                evaluationService,
                recommendationService,
                queueService,
                learningService
            )

            // === ODGOVORNOST RUNNER-A: Izvrši Ciklus ===
            result ← AWAIT runner.StepAsync(stoppingToken)

            // === ODGOVORNOST HOST-A: Loguj i Vremenuj ===
            IF result ≠ null THEN
                LogInformation("Tick completed: {0} tasks
evaluated",
                               result.TasksEvaluated)
                AWAIT Task.Delay(5 seconds, stoppingToken) // ←
HOST kontroliše vremenovanje
            ELSE
                LogDebug("No work available")
                AWAIT Task.Delay(10 seconds, stoppingToken) // ←
Duže čekanje
            END IF

            // === ODGOVORNOST HOST-A: Čišćenje ===
            scope.Dispose()

        CATCH OperationCanceledException
            BREAK // Graciozno gašenje
        CATCH exception
```

```
            LogError("Error in agent loop", exception)
            AWAIT Task.Delay(10 seconds, stoppingToken) // ← Čekanje pri grešci
        END TRY
    END WHILE
END
```

Ključno razdvajanje:

- Runner: Čista logika (Sense-Think-Act-Learn).
- Host: Vremenovanje, scoping, logovanje, rukovanje greškama.

STRUKTURA PROJEKTA

Organizacija Solution-a

```
TaskAgent.Backend/
├── TaskAgent.Backend.sln
│
├── TaskAgent.Core/              [LAYER 1: Core Abstractions]
│   ├── IPerceptionSource.cs
│   ├── IPolicy.cs
│   ├── IActuator.cs
│   ├── ILearningComponent.cs
│   ├── SoftwareAgent.cs
│   └── TickResult.cs
│
├── TaskAgent.Tasks/             [LAYERS 2-4: Domain/App/Infra]
│   ├── TaskAgent.Tasks.csproj
│   │
│   ├── Domain/                  [LAYER 2: Domain]
```

```
│   │   ├── Entities/
│   │   │   ├── TaskItem.cs
│   │   │   ├── TaskRecommendation.cs
│   │   │   └── SystemSettings.cs
│   │   ├── Enums/
│   │   │   ├── TaskStatus.cs
│   │   │   └── TaskPriority.cs
│   │   └── Exceptions/
│   │       └── DomainExceptions.cs
│   │
│   ├── Application/            [LAYER 3: Application]
│   │   ├── Services/
│   │   │   ├── TaskQueueService.cs
│   │   │   ├── TaskEvaluationService.cs
│   │   │   ├── RecommendationService.cs
│   │   │   ├── LearningService.cs
│   │   │   └── UserActionService.cs
│   │   ├── Runners/
│   │   │   ├── TaskScoringAgentRunner.cs
│   │   │   └── TaskAdaptationAgentRunner.cs
│   │   ├── DTOs/
│   │   │   ├── TaskScoringTickResult.cs
│   │   │   └── AdaptationTickResult.cs
│   │   └── Interfaces/
│   │       ├── ITaskRepository.cs
│   │       └── ISettingsRepository.cs
│   │
│   └── Infrastructure/         [LAYER 4: Infrastructure]
│       ├── Persistence/
```

```
│       │   ├── TaskAgentDbContext.cs
│       │   └── Configurations/
│       │       ├── TaskItemConfiguration.cs
│       │       ├── TaskRecommendationConfiguration.cs
│       │       └── SystemSettingsConfiguration.cs
│       ├── Repositories/
│       │   ├── TaskRepository.cs
│       │   └── SettingsRepository.cs
│       ├── Seeding/
│       │   └── DatabaseSeeder.cs
│       ├── Abstractions/
│       │   └── ISystemClock.cs
│       └── InfrastructureServiceExtensions.cs
│
└── TaskAgent.Web/              [LAYER 5: Web Host]
    ├── TaskAgent.Web.csproj
    ├── Program.cs
    ├── appsettings.json
    ├── Controllers/
    │   └── TasksController.cs
    ├── Workers/
    │   └── TaskAgentWorker.cs
    ├── DTOs/
    │   └── TaskDtos.cs
    └── Mapping/
        └── TaskMapper.cs
```

Odgovornosti Slojeva

LAYER 1: TaskAgent.Core

Svrha: Generičke, ponovno upotrebljive apstrakcije agenta.
Sadrži: Generic SoftwareAgent<> base class, Interface contracts for agent components, TickResult base DTO.
Zavisnosti: NONE
Pravila: No domain knowledge, No infrastructure dependencies, Reusable across any agent system.

LAYER 2: Domain (TaskAgent.Tasks.Domain)

Svrha: Poslovni entiteti i domen pravila.
Sadrži: Entiteti: TaskItem, TaskRecommendation, SystemSettings (Bogati domen modeli sa ponašanjem, metode tranzicije stanja, nametanje invarijanti). Enumi: TaskStatus, TaskPriority (Domen-specifični tipovi). Izuzeci: Domain-specific errors (npr. InvalidStateTransitionException, TaskAlreadyCompletedException).
Zavisnosti: NONE
Pravila: NO infrastructure concerns (no EF attributes), NO DTOs (pure domain objects), NO application logic (only domain rules), State transitions enforce invariants.
Primjer:

```
// Domen nameće poslovna pravila
public void Escalate()
{
    EnsureValidTransition(TaskStatus.Escalated); // ← Domen pravilo
    _status = TaskStatus.Escalated;
    _priority = TaskPriority.Critical; // ← Domen pravilo
    EscalationCount++;
}
```

LAYER 3: Application (TaskAgent.Tasks.Application)

Svrha: Slučajevi korištenja, heuristika i logika agenta.
Sadrži: Servisi: Orchestrate domain operations (TaskQueueService, TaskEvaluationService, RecommendationService, LearningService, UserActionService). Runneri: Agent loop orchestration (TaskScoringAgentRunner,

TaskAdaptationAgentRunner). DTO-ovi: Application-level models (TaskPercept, ScoredTask, TaskAction, etc.). Interfejsi: Repository contracts (ITaskRepository, ISettingsRepository).
Zavisnosti: Core, Domain
Pravila: Contains ALL business logic (heuristics, policies), NO infrastructure code (no EF, no HTTP), Defines repository interfaces (Dependency Inversion), Runners are stateless orchestrators.
Primjer:

```
// Aplikacija sadrži heuristiku
private double CalculateTimeWeight(TaskItem task)
{
    if (!task.DueDate.HasValue)
        return 0.3;
    var hoursUntilDue = (task.DueDate.Value - 
UtcNow).TotalHours;
    if (hoursUntilDue <= 1) return 0.95;
    if (hoursUntilDue <= 4) return 0.85;
    if (hoursUntilDue <= 12) return 0.70;
    // ... heuristic logic
}
```

LAYER 4: Infrastructure (TaskAgent.Tasks.Infrastructure)

Svrha: Tehničke implementacije (baza podataka, eksterni servisi).
Sadrži: Persistence: EF Core DbContext i konfiguracije. Repozitoriji: Implementiraju aplikacijske interfejse. Seeding: Inicijalizacija baze podataka. Apstrakcije: Tehnički interfejsi (ISystemClock). DI Extensions: Registracija servisa.
Zavisnosti: Core, Domain, Application
Pravila: NO business logic, Implements repository interfaces from Application, EF configurations in separate files (not in entities), Pure data access.

Primjer:

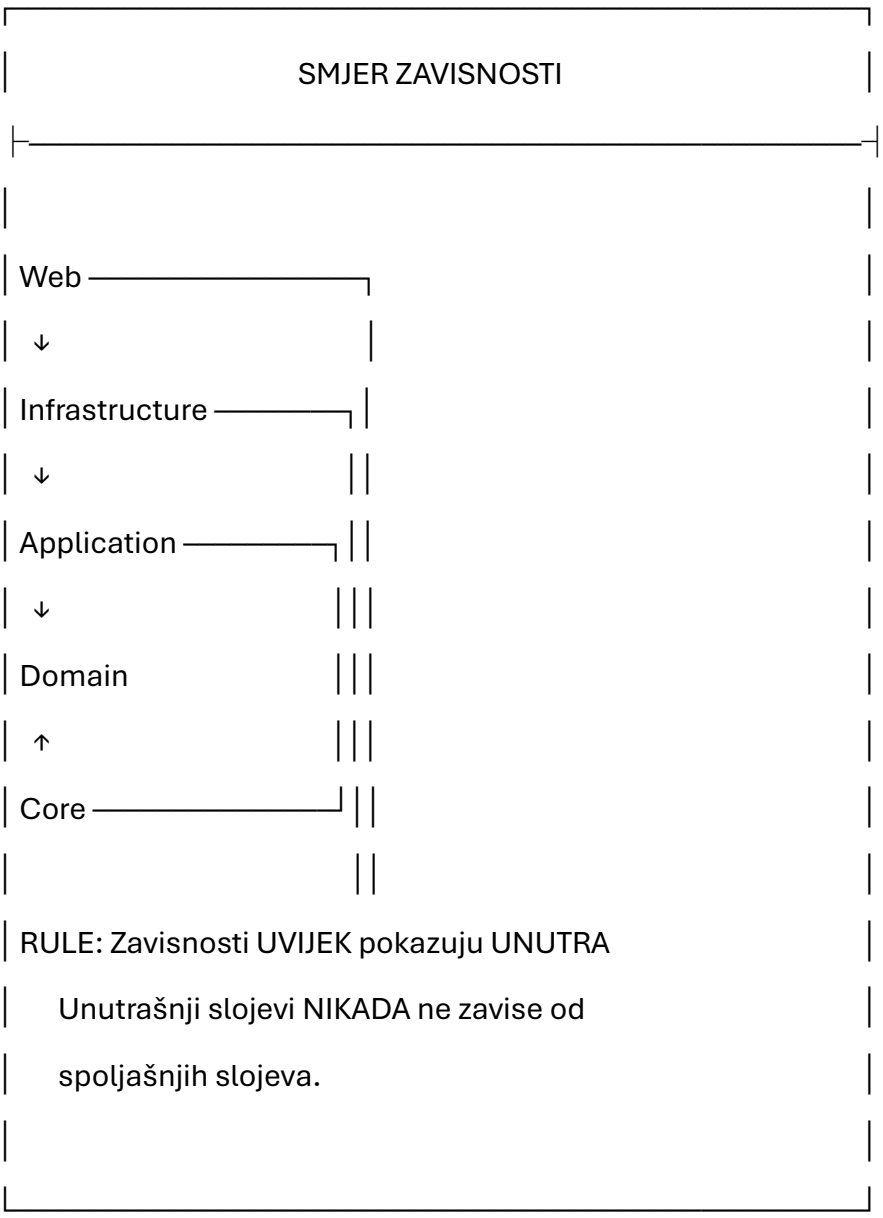
```
// Infrastruktura: Čist pristup podacima
public async Task<TaskItem?> GetByIdAsync(Guid id)
{
    return await _context.Tasks
        .AsNoTracking()
        .FirstOrDefaultAsync(t => t.Id == id);
}
```

LAYER 5: Web (TaskAgent.Web)

Svrha: HTTP transport i pozadinska orkestracija.
Sadrži: Kontroleri: Thin HTTP endpoints (TasksController: CRUD + korisničke akcije). Workeri: Background services (TaskAgentWorker: Orchestrates agent ticks). DTO-ovi: Web-specific models (CreateTaskRequest, TaskResponse). Mapiranje: Domain ↔ DTO conversion (TaskMapper).
Zavisnosti: ALL LAYERS (composition root)
Pravila: NO business logic in controllers, NO decision-making in workers, Workers only handle timing and scoping, Controllers delegate to services immediately.
Primjer:

```
// Web: Tanak kontroler
[HttpPost("{id}/complete")]
public async Task<IActionResult> CompleteTask(Guid id)
{
    var success = await
_userActionService.RequestCompleteTaskAsync(id);
    return success ? Ok() : NotFound();
}
```

Protok Zavisnosti



5. SAŽETAK

Ovaj sistem implementira inteligentnog agenta za produkciju slijedeći:

1. Čistu Arhitekturu – Strogo razdvajanje slojeva.
2. Dizajn zasnovan na Agentima – Sense-Think-Act-Learn petlja.
3. Domain-Driven Design – Bogati domen modeli.
4. SOLID principe – Jedinstvena odgovornost, Inverzija zavisnosti.
5. Razdvajanje briga – Granice odlučivanja između Agenta i Korisnika.
6. Izvršenje zasnovano na ciklusima – Atomične, operacije bez stanja.
7. Meta-Učenje – Samooptimizacija kroz adaptaciju.

Akademska klasifikacija:
Hibridni Agent zasnovan na Korisnosti sa Učenjem i Multi-Agent Arhitekturom (Russell & Norvig, AIMA).