

10 ערכת כלים בפייתון

10.1 מבוא: מהקונפיגורציה לקוד

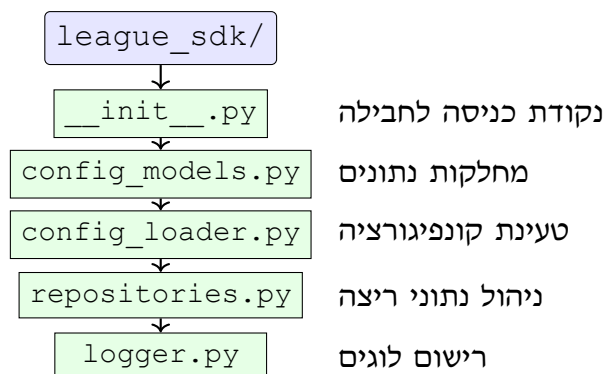
בנספח ב הצגנו את ארכיטקטורת הנתונים מבוססת קובצי JSON – שלוש שכבות של קונפיגורציה, נתוני ריצה, ולוגים. אך כיצד סוכן בודד ניגש לנתונים אלו בפועל?

נספח זה מציג את league_sdk – ספריית Python שמגשרת בין קובצי ה-JSON לבין האובייקטים שבהם משתמשים הסוכנים. הספרייה מיישמת שני דפוסי עיצוב מרכזיים:

1. **Dataclasses** – מודלים טיפוסיים (typed models) שמשקפים את מבנה ה-JSON.

2. **Repository Pattern** – שכבת הפשטה לגישה לנתונים.

10.2 מבנה הספרייה



10.3 מודלים טיפוסיים – config_models.py

10.3.1 הגישה: Dataclasses

Python 3.7+ מספקת את הדקורטור @dataclass שמאפשר הגדרת מחלקות נתונים בצורה תמציתית. כל שדה ב-JSON הופך לשדה במחלקה עם טיפוס מוגדר:

דוגמה: הגדרת Dataclass

```
from dataclasses import dataclass
from typing import List

@dataclass
class NetworkConfig:
    base_host: str
    default_league_manager_port: int
    default_referee_port_range: List[int]
    default_player_port_range: List[int]
```

10.3.2 מודלי קונפיגורציית מערכת

הקובץ מגדיר את כל המודלים התואמים ל-`config/system.json`:

מודלי קונפיגורציה גלובלית

```
@dataclass
class SecurityConfig:
    enable_auth_tokens: bool
    token_length: int
    token_ttl_hours: int

@dataclass
class TimeoutsConfig:
    register_referee_timeout_sec: int
    register_player_timeout_sec: int
    game_join_ack_timeout_sec: int
    move_timeout_sec: int
    generic_response_timeout_sec: int

@dataclass
class SystemConfig:
    schema_version: str
    system_id: str
    protocol_version: str
    default_league_id: str
    network: NetworkConfig
    security: SecurityConfig
    timeouts: TimeoutsConfig
    # ...additional fields
```

כל סוג סוכן מקבל מחלקת קונפיגורציה משלו:

מודלי קונפיגורציית סוכנים

```
@dataclass
class RefereeConfig:
    referee_id: str
    display_name: str
    endpoint: str
    version: str
    game_types: List[str]
    max_concurrent_matches: int
    active: bool = True

@dataclass
class PlayerConfig:
    player_id: str
    display_name: str
    version: str
    preferred_leagues: List[str]
    game_types: List[str]
    default_endpoint: str
    active: bool = True
```

הגדרות ספציפיות לליגה כוללות תזמון, ניקוד, ומשתתפים:

מודלי קונפיגורציית ליגה

```
@dataclass
class ScoringConfig:
    win_points: int
    draw_points: int
    loss_points: int
    technical_loss_points: int
    tiebreakers: List[str]

@dataclass
class LeagueConfig:
    schema_version: str
    league_id: str
    display_name: str
    game_type: str
    status: str
    scoring: ScoringConfig
    # ...additional fields
```

10.4 טוען קונפיגורציה – ConfigLoader

10.4.1 העיקרון: טעינה עצלה עם מטמון

המחלקה ConfigLoader מיישמת את דפוס ה-Lazy Loading – קובצי הקונפיגורציה נטענים רק כשיש בהם צורך, ונשמרים במטמון לגישה חוזרת:

מבנה ConfigLoader

```
class ConfigLoader:
    def __init__(self, root: Path = CONFIG_ROOT):
        self.root = root
        self._system = None          # lazy cache
        self._agents = None          # lazy cache
        self._leagues = {}           # league_id -> LeagueConfig

    def load_system(self) -> SystemConfig:
        """Load global system configuration."""
        if self._system:
            return self._system
        path = self.root / "system.json"
        data = json.loads(path.read_text(encoding="utf-8"))
        self._system = SystemConfig(...)
        return self._system
```

10.4.2 שיטות הטעינה

ConfigLoader מספקת ממשק אחיד לטעינת כל סוגי הקונפיגורציה:

טבלה 19: שיטות הטעינה של ConfigLoader

שיטה	מחזירה	תיאור
load_system()	SystemConfig	קונפיגורציה גלובלית
load_agents()	AgentsConfig	רשימת כל הסוכנים
load_league(id)	LeagueConfig	קונפיגורציית ליגה ספציפית
load_games_registry()	GamesRegistry	רישום סוגי המשחקים

10.4.3 שיטות עזר

בנוסף לטעינה ישירה, המחלקה מספקת שיטות נוחות לחיפוש:

שיטות עזר

```
def get_referee_by_id(self, referee_id: str) -> RefereeConfig:
    """Get a referee configuration by ID."""
    agents = self.load_agents()
    for ref in agents.referees:
        if ref.referee_id == referee_id:
            return ref
    raise ValueError(f"Referee not found: {referee_id}")

def get_player_by_id(self, player_id: str) -> PlayerConfig:
    """Get a player configuration by ID."""
    agents = self.load_agents()
    for player in agents.players:
        if player.player_id == player_id:
            return player
    raise ValueError(f"Player not found: {player_id}")
```

10.5 מאגרי נתונים – Repositories

10.5.1 דפוס המאגר (Repository Pattern)

בעוד ש-ConfigLoader מטפל בקונפיגורציה סטטית, שכבת המאגרים מטפלת בנתונים דינמיים. כל מאגר אחראי על קריאה, עדכון, ושמירה של סוג נתונים ספציפי.

מאגר טבלת דירוג

```

class StandingsRepository:
    def __init__(self, league_id: str, data_root: Path = DATA_ROOT):
        self.league_id = league_id
        self.path = data_root / "leagues" / league_id / "standings.json"
        self.path.parent.mkdir(parents=True, exist_ok=True)

    def load(self) -> Dict:
        """Load standings from JSON file."""
        if not self.path.exists():
            return {"schema_version": "1.0.0", "standings": []}
        return json.loads(self.path.read_text(encoding="utf-8"))

    def save(self, standings: Dict) -> None:
        """Save standings to JSON file."""
        standings["last_updated"] = datetime.utcnow().isoformat() + "Z"
        self.path.write_text(json.dumps(standings, indent=2))

    def update_player(self, player_id: str, result: str, points: int):
        """Update a player's standings after a match."""
        standings = self.load()
        # ... update logic
        self.save(standings)

```

10.5.3 מאגרים נוספים

הספרייה כוללת מאגרים נוספים לניהול נתוני ריצה:

טבלה 20: מאגרי נתונים זמינים

מאגר	קובץ	תפקיד
StandingsRepository	standings.json	טבלת דירוג ליגה
RoundsRepository	rounds.json	היסטוריית מחזורים
MatchRepository	<match_id>.json	נתוני משחק בודד
PlayerHistoryRepository	history.json	היסטוריית שחקן

10.6 רישום לוגים – JsonLogger

10.6.1 פורמט JSON Lines

הספרייה משתמשת בפורמט JSONL (JSON Lines) – כל שורה בקובץ הלוג היא אובייקט JSON עצמאי. פורמט זה מאפשר:

- הוספת רשומות חדשות בצורה יעילה (append-only).
- קריאה וניתוח באמצעות כלים סטנדרטיים.
- סטרימינג של לוגים בזמן אמת.

10.6.2 מחלקת הלוגר

מחלקת JsonLogger

```
class JsonLogger:
    def __init__(self, component: str, league_id: str | None = None):
        self.component = component
        # Determine log directory
        if league_id:
            subdir = LOG_ROOT / "league" / league_id
        else:
            subdir = LOG_ROOT / "system"
        subdir.mkdir(parents=True, exist_ok=True)
        self.log_file = subdir / f"{component}.log.jsonl"

    def log(self, event_type: str, level: str = "INFO", **details):
        entry = {
            "timestamp": datetime.utcnow().isoformat() + "Z",
            "component": self.component,
            "event_type": event_type,
            "level": level,
            **details,
        }
        with self.log_file.open("a", encoding="utf-8") as f:
            f.write(json.dumps(entry, ensure_ascii=False) + "\n")
```


הלוגר מספק שיטות לרמות לוג שונות ולאירועים נפוצים:

שיטות נוחות ללוגר

```
def debug(self, event_type: str, **details):
    self.log(event_type, level="DEBUG", **details)

def info(self, event_type: str, **details):
    self.log(event_type, level="INFO", **details)

def warning(self, event_type: str, **details):
    self.log(event_type, level="WARNING", **details)

def error(self, event_type: str, **details):
    self.log(event_type, level="ERROR", **details)

def log_message_sent(self, message_type: str, recipient: str, **details):
    :
    self.debug("MESSAGE_SENT", message_type=message_type,
               recipient=recipient, **details)
```

שימוש ב-ConfigLoader במנהל ליגה

```
from league_sdk import ConfigLoader, JsonLogger

class LeagueManager:
    def __init__(self, league_id: str):
        loader = ConfigLoader()
        self.system_cfg = loader.load_system()
        self.agents_cfg = loader.load_agents()
        self.league_cfg = loader.load_league(league_id)

        self.logger = JsonLogger("league_manager", league_id)

        # Build lookup maps
        self.referees_by_id = {
            r.referee_id: r.endpoint
            for r in self.agents_cfg.referees if r.active
        }

    def get_timeout_for_move(self) -> int:
        return self.system_cfg.timeouts.move_timeout_sec
```

שימוש ב-ConfigLoader בשופט

```

from league_sdk import ConfigLoader, JsonLogger

class RefereeAgent:
    def __init__(self, referee_id: str, league_id: str):
        loader = ConfigLoader()
        self.system_cfg = loader.load_system()
        self.league_cfg = loader.load_league(league_id)
        self.self_cfg = loader.get_referee_by_id(referee_id)

        self.logger = JsonLogger(f"referee:{referee_id}", league_id)

    def register_to_league(self):
        payload = {
            "jsonrpc": "2.0",
            "method": "register_referee",
            "params": {
                "protocol": self.system_cfg.protocol_version,
                "message_type": "REFEREE_REGISTER_REQUEST",
                "referee_meta": {
                    "display_name": self.self_cfg.display_name,
                    "version": self.self_cfg.version,
                    "game_types": self.self_cfg.game_types,
                }
            }
        }
        # ... send request

```

TIMEOUT שגיאת

```

logger = JsonLogger("referee:REF01", "league_2025_even_odd")

logger.error(
    "GAME_ERROR",
    match_id="R1M1",
    error_code="TIMEOUT_MOVE",
    player_id="P02",
    timeout_sec=30,
)
# Output to logs/league/league_2025_even_odd/referee_REF01.log.jsonl:
# {"timestamp": "2025-01-15T10:15:00Z", "component": "referee:REF01",
#  "event_type": "GAME_ERROR", "level": "ERROR", "match_id": "R1M1", ...}

```

10.8 סיכום

ספריית league_sdk מספקת שכבת הפשטה נקייה בין קובצי ה-JSON לקוד הסוכנים:

- config_models.py – מגדיר טיפוסים בטוחים (type-safe) לכל מבנה נתונים.
 - config_loader.py – מספק גישה נוחה לקונפיגורציה עם מטמון.
 - repositories.py – מנהל נתוני ריצה בדפוס Repository.
 - logger.py – מאפשר רישום לוגים מובנה בפורמט JSONL.
- שימוש בספרייה זו מבטיח:

1. **עקביות** – כל הסוכנים משתמשים באותם מודלים ונתונים.
2. **תחזוקתיות** – שינויים במבנה הנתונים מתרכזים במקום אחד.
3. **בטיחות טיפוסים** – שגיאות נתפסות בזמן כתיבת הקוד.
4. **יכולת ניפוי** – לוגים מובנים מאפשרים מעקב קל אחר בעיות.

הספרייה זמינה בתיקייה:

L07/SHARED/league_sdk/