모듈 개발 총괄 과업지시서 (v1.3)

목표

1. **DB 무관·단독 실행** 가능한 기능별 모듈 구현
2. **더미 데이터**로 I/O 검증 가능하도록 구성
3. ``** 페이지**에서 모든 모듈을 한눈에 테스트·시연

---

1. 디렉터리 / 파일 구조

> 프로젝트 특성상 `` 디렉터리가 3-중 구조로 존재합니다.
>
> 새 임시 모듈·더미 데이터·테스트 디렉터리는 **두 번째 레벨(`manage.py`가 위치한 곳)**에 추가합니다.

```
ab_foundation2/              # 1️⃣ 최상위 (IDE 프로젝트 루트)
├── ab_foundation2/          # 2️⃣ Django 프로젝트 루트 (manage.py 레벨)
│   ├── ab_foundation2/      # 3️⃣ settings & WSGI 패키지 (중첩 유지)
│   │   ├── settings.py
│   │   ├── urls.py
│   │   └── ...
│   ├── accounting/          # 1st-party 앱들
│   ├── i18n/
│   ├── users/
│   ├── temp_modules/        # 이번 과업 대상 모듈 저장
│   ├── dummy_data/          # 샘플 PDF·JSON·XML·YAML
│   ├── outputs/             # 모듈 실행 산출물
│   ├── tests/               # pytest
│   ├── test_home/           # FastAPI + HTMX UI
│   │   └── app.py
│   ├── manage.py
│   └── requirements.txt
└── ...                      # 기타 IDE 파일
```

---

2. 공통 가이드라인

- **입·출력** : 함수 호출형 + CLI 엔트리포인트, 결과는 `outputs/` 아래 JSON/HTML 저장
- **더미 데이터** : `dummy_data/` 경로를 하드코딩으로 사용
- **로깅** : `temp_modules.utils.get_logger(mod)` → 콘솔 + `outputs/{mod}.log`
- **테스트** : 모듈당 최소 1개 happy-path `pytest`
- **i18n** : `i18n_stub._("key", lang)` 사용 (`dummy_data/locales/*.json`)

2-1. 프레임워크 구성

📎 **FastAPI 서비스 구성, 실행 방법, 엔드포인트 구조 등은 별도 문서인 `` 를 참고할 것.**

- **Django 4.1** (필수) – Admin·ORM·Auth· `test_home` 페이지 호스팅
- **FastAPI (uvicorn)** – 외부 *auto-accounting* 마이크로서비스(API Gateway)

  - OCR·JE 생성·월마감 10-Step 등 비동기 처리

  - Django ⇔ FastAPI : REST/gRPC + mTLS & JWT

> 개발 진행 순서
>
> 1. *Django UI & **`` 먼저 구현하여 화면·양식·HTMX 요청 구조를 확정

> 2. 각 버튼이 호출하는 **FastAPI 기능 모듈**을 더미 데이터 기반으로 구현·테스트
> 3. 마지막 단계에서 **DB 연동(ORM 모델 교체)** 을 적용해 실데이터 저장·조회로 전환

2-2. `test_home` 화면 구성 세부 (사이드바 네비게이션)

아래 레이아웃은 좌측 사이드바 메뉴를 통해 11개 기능 화면을 HTMX로 동적으로 로드합니다.

```
<div class="layout flex h-screen">
 <!-- 사이드바 -->
 <nav class="w-1/5 bg-gray-200 p-4">
  <ul>
   <li hx-get="/test/upload/"    hx-target="#main-content" class="menu-item">1. Upload</li>
   <li hx-get="/test/parser/"    hx-target="#main-content" class="menu-item">2. Parser</li>
   <li hx-get="/test/masters/"   hx-target="#main-content" class="menu-item">3. Masters Sync</li>
   <li hx-get="/test/je/"        hx-target="#main-content" class="menu-item">4. JE Build</li>
   <li hx-get="/test/gl/"        hx-target="#main-content" class="menu-item">5. GL Posting</li>
   <li hx-get="/test/fs/"        hx-target="#main-content" class="menu-item">6. FS Aggregate</li>
   <li hx-get="/test/recon/"     hx-target="#main-content" class="menu-item">7. Bank Recon</li>
   <li hx-get="/test/closing/"   hx-target="#main-content" class="menu-item">8. Closing</li>
   <li hx-get="/test/dw/"        hx-target="#main-content" class="menu-item">9. DW Load</li>
   <li hx-get="/test/i18n/"      hx-target="#main-content" class="menu-item">10. i18n Lookup</li>
   <li hx-get="/test/dashboard/" hx-target="#main-content" class="menu-item">11. Dashboard</li>
  </ul>
 </nav>
 <!-- 메인 콘텐츠 영역 -->
 <main id="main-content" class="flex-1 p-4 overflow-auto">
  <!-- 메뉴 클릭 시 해당 모듈 UI가 HTMX로 로드 -->
 </main>
</div>
```

- **공통 컴포넌트**:

```
<div class="module-panel">
 <button id="btn-<mod>" hx-post="<api-endpoint>" hx-target="#result-<mod>" hx-indicator="#spinner-<mod
>">실행</button>
 <span id="spinner-<mod>" class="spinner hidden"></span>
 <div id="result-<mod>" class="mt-2 bg-gray-100 p-2"></div>
 <div id="err-<mod>" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
```

- **동작 흐름**:
  1. 사이드바 메뉴 클릭 → HTMX(`hx-get`)로 해당 템플릿 로드
  2. 실행 버튼 클릭 → API 호출 및 결과/오류 표시

2-3. `test_home` 템플릿 파일 예시

아래는 각 화면별로 `test_home/templates/` 디렉터리에 생성할 HTML 파일 샘플입니다.

upload.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
 <input type="file" id="file-input-upload" name="file" />
 <button id="btn-upload" hx-post="/test/upload/" hx-include="#file-input-upload" hx-target="#result-upload" hx-i
ndicator="#spinner-upload">Upload</button>
```

```
  <span id="spinner-upload" class="spinner hidden"></span>
  <div id="result-upload" class="mt-2 bg-gray-100 p-2"></div>
  <div id="err-upload" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
{% endblock %}
```

parser.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <input type="file" id="file-input-parse" name="file" />
  <button id="btn-parse" hx-post="/test/parser/" hx-include="#file-input-parse" hx-target="#result-parse" hx-indi
cator="#spinner-parse">Parse</button>
  <span id="spinner-parse" class="spinner hidden"></span>
  <pre id="result-parse" class="mt-2 bg-gray-100 p-2"></pre>
  <pre id="err-parse" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></pre>
</div>
{% endblock %}
```

masters.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-masters" hx-post="/test/masters/" hx-target="#result-masters" hx-indicator="#spinner-master
s">Sync Masters</button>
  <span id="spinner-masters" class="spinner hidden"></span>
  <table id="result-masters" class="mt-2 w-full bg-gray-100"></table>
  <div id="err-masters" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
{% endblock %}
```

je.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-je" hx-post="/test/je/" hx-target="#result-je" hx-indicator="#spinner-je">Build JE</button>
  <span id="spinner-je" class="spinner hidden"></span>
  <table id="result-je" class="mt-2 w-full bg-gray-100"></table>
  <div id="err-je" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
{% endblock %}
```

gl.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-gl" hx-post="/test/gl/" hx-target="#result-gl" hx-indicator="#spinner-gl">Post GL</button>
  <span id="spinner-gl" class="spinner hidden"></span>
  <table id="result-gl" class="mt-2 w-full bg-gray-100"></table>
  <div id="err-gl" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
```

```
</div>
{% endblock %}
```

fs.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-fs" hx-post="/test/fs/" hx-target="#result-fs" hx-indicator="#spinner-fs">Aggregate FS</button
>
  <span id="spinner-fs" class="spinner hidden"></span>
  <table id="result-fs" class="mt-2 w-full bg-gray-100"></table>
  <div id="err-fs" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
{% endblock %}
```

recon.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-recon" hx-post="/test/recon/" hx-target="#result-recon" hx-indicator="#spinner-recon">Run Re
con</button>
  <span id="spinner-recon" class="spinner hidden"></span>
  <div id="result-recon" class="mt-2 bg-gray-100 p-2"></div>
  <div id="err-recon" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></div>
</div>
{% endblock %}
```

closing.html

```
{% extends "base.html" %}
{% block content %}
<div class="module-panel">
  <button id="btn-closing" hx-post="/test/closing/" hx-target="#result-closing" hx-indicator="#spinner-closing">R
un Closing</button>
  <span id="spinner-closing" class="spinner hidden"></span>
  <pre id="result-closing" class="mt-2 bg-gray-100 p-2"></pre>
  <pre id="err-closing" class="mt-2 bg-red-100 text-red-800 p-2 hidden"></pre>
</div>
{% endblock %}
```

---

3. 기능별 모듈 명세 및 테스트 화면 연결

아래 명세는 **백엔드 모듈(서비스)** 와 ``** UI 컴포넌트**를 1-대-1로 매핑한다. 각 화면에는 *실행 버튼*, *결과 패널*, *오류 패널*이 공통 배치된다.

No	Layer	모듈 (서비스)	FastAPI 엔드포인트	Django View (HTMX)	버튼 ID	성공 출력
1	Django	`uploader.py`	`POST /api/upload/`	`/test/upload/`	`btn-upload`	`upload_log.json` 내용 테이블
2	FastAPI	`parser.py`	`POST /api/parse/`	`/test/parser/`	`btn-parse`	인보이스 JSON pretty-print
3	FastAPI	`masters_sync.py`	`POST /api/masters/sync/`	`/test/masters/`	`btn-masters`	동기화 결과 리스트

4	FastAPI	`je_builder.py`	`POST /api/je/build/`	`/test/je/`	`btn-je`	차·대 테이블 & saldo OK 뱃지
5	FastAPI	`gl_posting.py`	`POST /api/gl/post/`	`/test/gl/`	`btn-gl`	`trial_balance.jso` 표
6	FastAPI	`fs_aggregate.py`	`POST /api/fs/aggregate/`	`/test/fs/`	`btn-fs`	PL·BS·CF 테이
7	FastAPI	`bank_recon.py`	`POST /api/recon/run/`	`/test/recon/`	`btn-recon`	매칭률 % & 리스트
8	FastAPI	`closing_runner.py`	`POST /api/closing/run/`	`/test/closing/`	`btn-closing`	단계별 로그 스트림
9	Django	`dw_loader.py`	`POST /api/dw/load/`	`/test/dw/`	`btn-dw`	DW 적재 성공 뱃지
10	Django	`i18n_stub.py`	`GET /api/i18n/lookup/`	`/test/i18n/`	`btn-i18n`	선택 언어 번역 문자열
11	Django	`dashboard_stub.py`	`GET /test/dashboard/embed/`	`/test/dashboard/`	`btn-dashboard`	`dashboard.html` iframe 로드

각 화면별 상세 지시사항

1. **Upload** (`btn-upload` 클릭)
   - 파일 선택 모달 오픈 → `dummy_data/invoices/1. Sales Invoice Selling Two Computers.pdf` 선택
   - 업로드 요청 전송 (`POST /api/upload/`)
   - 성공 시 `result-upload`에 테이블 렌더링, 거래문서 번호 표시
     - 문서번호 **YYYY_MM_CI_######** 포맷 (년4+월2자리+CI+숫자6)
     - 번호는 연속적 부여 (삭제해도 공백 없이)
   - 실패 시 `err-upload`에 스택트레이스 출력
2. **Parser** (`btn-parse` 클릭)
   - 지정된 PDF 경로로 OCR 요청 (`POST /api/parse/`)
   - 성공 응답 JSON을 `result-parse`에 pretty-print
   - 실패 시 `err-parse`에 에러 스택트레이스 표시
3. **Masters Sync** (`btn-masters` 클릭)
   - `invoice_dict` 기반 get_or_create 실행 (`POST /api/masters/sync/`)
   - 성공 시 `result-masters`에 생성·기존 구분 리스트 테이블
     - 마스터 종류: 거래처(Entity), 품목(Article), 고정자산(Fixed asset), 프로젝트(Project)
     - 신규 생성된 각 마스터에 6자리 숫자 부여
     - 생성 후 사용자가 확인(Confirm) 시 승인 처리
   - 실패 시 `err-masters`에 에러 표시
4. **JE Build** (`btn-je` 클릭)
   - 차·대 분개 로직 호출 (`POST /api/je/build/`)
   - 인보이스 정보 기반 입력·추론 필드 채움
   - 동일 거래처/유사 거래 분석 통한 이상감지 수행
   - 모든 수치는 소수점 2자리까지 표기
   - 작성된 회계전표는 임시전표(임시번호)→사용자 확인 후 Confirm 시 최종 전표 번호 발행
     - 전표번호 포맷: YYYY + 숫자 8자리, 연속 발번
     - 전표 목록 페이지에서 검색·조회 가능 (Posting Date·계정번호 기준)
   - 성공 시 `result-je`에 테이블+`badge-success`, 실패 시 `err-je`
5. **GL Posting** (`btn-gl` 클릭)

- 잔액 실시간 갱신 (`POST /api/gl/post/`)
- `result-gl` 에 Trial Balance 표 출력
- `err-gl` 에 오류 상세 표시

6. **FS Aggregate** (`btn-fs` 클릭)
   - PL·BS·CF 집계 (`POST /api/fs/aggregate/`)
   - `result-fs` 에 각 재무제표 테이블 출력
   - `err-fs` 에 에러 표시
7. **Bank Recon** (`btn-recon` 클릭)
   - CAMT v2 매칭 실행 (`POST /api/recon/run/`)
   - 진행률 표시바 업데이트 → `result-recon` 에 매칭률 + 리포트
   - 매칭 실패 거래는 Open AR/AP 리스트로 제공, 사용자가 수동 처리
   - `err-recon` 에 에러 표시
8. **Closing Runner** (`btn-closing` 클릭)
   - 10-Step 순차 실행 (`POST /api/closing/run/`)
   - 각 단계별 과업 설명 + 요약 결과 표시
   - 최종 마감 결과 리포트 생성
   - `result-closing` 에 실시간 로그 append, `err-closing` 에 에러 표시
9. **DW Loader** (`btn-dw` 클릭)
   - JE·FS Fact 적재 (`POST /api/dw/load/`)
   - `result-dw` 에 성공 뱃지, `err-dw` 에 에러 표시
10. **i18n Lookup** (`btn-i18n` 클릭)
    - 언어 선택 드롭다운 → 조회(`GET /api/i18n/lookup/?lang={code}`)
    - `result-i18n` 에 선택 언어 값 표시, `err-i18n` 에 에러 표시
11. **Dashboard Embed** (`btn-dashboard` 클릭)
    - 대시보드 결과물에 대한 사용자 선택·필터 옵션 UI 제공
    - `<iframe>` 로 로드 (`GET /test/dashboard/embed/`)
    - `err-dashboard` 에 로딩 에러 메시지 표시

각 화면별 상세 지시사항 및 구현 파일/함수 명세

각 화면별로 다음 **파일**과 **함수** 형태로 구현하세요. 각 파일의 정확한 경로를 기재하고, 함수 시그니처에 입력(Parameter)과 반환값(Return)을 명시합니다.

1. **Upload** (`POST /api/upload/` / `/test/upload/`)
   - **Files**:
     - `accounting/models.py`
     - `accounting/views.py`
     - `accounting/urls.py`
     - `test_home/templates/upload.html`
   - **Function** (`accounting/views.py`):

```
def upload_invoice(file: UploadedFile) -> dict:
    """
    :param file: 업로드된 인보이스 파일
    :return: {
        'document_no': str,  # YYYY_MM_CI_######
```

```
    'file_url': str,
    'created_at': str
}
"""
```

2. **Parser** (`POST /api/parse/` / `/test/parser/`)
   - **Files**:
     - `parser_service/main.py`
     - `parser_service/schemas.py`
     - `test_home/templates/parser.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def parse_invoice(file: UploadFile) -> InvoiceDict:
    """
    :param file: 업로드된 인보이스 파일
    :return: {invoice_id: str, issuance_date: str, items: list, total_amount: float, ...}
    """
```

3. **Masters Sync** (`POST /api/masters/sync/` / `/test/masters/`)
   - **Files**:
     - `parser_service/main.py`
     - `accounting/models.py`
     - `test_home/templates/masters.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def sync_masters(invoice: InvoiceDict) -> List[MasterResult]:
    """
    :param invoice: parse_invoice 반환 구조체
    :return: [
        {'type': 'Entity'|'Article'|'FixedAsset'|'Project', 'code': str, 'status': 'created'|'existing'},
        ...
    ]
    """
```

4. **JE Build** (`POST /api/je/build/` / `/test/je/`)
   - **Files**:
     - `parser_service/main.py`
     - `test_home/templates/je.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def build_journal_entries(invoice: InvoiceDict) -> JournalResult:
    """
    :param invoice: parse_invoice 반환 구조체
    :return: {
        'header': {'doc_no': str, 'trx_date': str},
        'lines': [{'account': str, 'debit': float, 'credit': float}, ...],
        'warnings': [str]
```

```
    }
    """
```

5. **GL Posting** (`POST /api/gl/post/` / `/test/gl/`)
   - **Files**:
     - `parser_service/main.py`
     - `test_home/templates/gl.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def post_gl(lines: List[JELine]) -> List[GLBalance]:
    """
    :param lines: build_journal_entries 반환 'lines'
    :return: [{'account': str, 'balance': float}, ...]
    """
```

6. **FS Aggregate** (`POST /api/fs/aggregate/` / `/test/fs/`)
   - **Files**:
     - `parser_service/main.py`
     - `test_home/templates/fs.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def aggregate_fs(trial_balance: List[GLBalance]) -> FinancialStatements:
    """
    :param trial_balance: post_gl 반환 리스트
    :return: {'PL': [...], 'BS': [...], 'CF': [...]}
    """
```

7. **Bank Recon** (`POST /api/recon/run/` / `/test/recon/`)
   - **Files**:
     - `parser_service/main.py`
     - `test_home/templates/recon.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def reconcile_bank(camt_xml: str, lines: List[JELine]) -> ReconReport:
    """
    :param camt_xml: CAMT.053 v2 XML
    :param lines: build_journal_entries 반환 'lines'
    :return: {'matched': [...], 'unmatched': [...], 'match_rate': float}
    """
```

8. **Closing Runner** (`POST /api/closing/run/` / `/test/closing/`)
   - **Files**:
     - `parser_service/main.py`
     - `test_home/templates/closing.html`
     - `test_home/urls.py`
   - **Function** (`parser_service/main.py`):

```
def run_month_end(data: RunData) → List[StepResult]:
    """
    :param data: aggregate_fs + reconcile_bank 결과
    :return: [{'step': int, 'name': str, 'status': str, 'message': str}, ...]
    """
```

9. **DW Loader** (`POST /api/dw/load/` / `/test/dw/`)
    - **Files**:
        - `parser_service/main.py`
        - `accounting/models.py`
        - `test_home/templates/dw.html`
        - `test_home/urls.py`
    - **Function** (`parser_service/main.py`):

```
def load_dw(journal_header: dict, journal_lines: List[JELine], fs: FinancialStatements) → bool:
    """
    :param journal_header: build_journal_entries 'header'
    :param journal_lines: build_journal_entries 'lines'
    :param fs: aggregate_fs 반환 dict
    :return: True|False
    """
```

10. **i18n Lookup** (`GET /api/i18n/lookup/` / `/test/i18n/`)
    - **Files**:
        - `accounting/i18n_stub.py`
        - `test_home/templates/i18n.html`
        - `test_home/urls.py`
    - **Function** (`accounting/i18n_stub.py`):

```
def lookup_i18n(key: str, lang: str) → str:
    """
    :return: 번역된 문자열
    """
```

11. **Dashboard Embed** (`GET /test/dashboard/embed/` / `/test/dashboard/`)
    - **Files**:
        - `test_home/templates/dashboard.html`
        - `test_home/urls.py`
    - **Function** (`test_home/views.py`):

```
def render_dashboard(request, filters: dict=None) → HttpResponse:
    """
    :param filters: 사용자 선택 필터
    :return: 대시보드 HTML
    """
```

2-4. 데이터 양식 정의 (Pydantic 스키마)

FastAPI `response_model` / `request_model` 및 Django Serializer 변환의 기준이 되는 데이터 구조입니다.

```
# parser_service/schemas.py
from typing import List
from pydantic import BaseModel, Field
from datetime import date

# ——————————— Invoice Dict ———————————
class Item(BaseModel):
    description_native: str
    description_en: str
    quantity: int = Field(..., ge=1)
    unit_price: float = Field(..., ge=0)

class Supplier(BaseModel):
    name_native: str
    name_en: str

class InvoiceDict(BaseModel):
    invoice_id: str
    issuance_date: date
    supplier: Supplier
    items: List[Item]
    total_amount: float = Field(..., ge=0)

# ——————————— Master Sync ———————————
class MasterResult(BaseModel):
    type: str  # Entity | Article | FixedAsset | Project
    code: str  # 6자리
    name_native: str | None = None
    name_en: str | None = None
    description_native: str | None = None
    description_en: str | None = None
    status: str  # created | existing

# ——————————— Journal Entry ———————————
class JELine(BaseModel):
    account: str
    description: str
    debit: float = 0
    credit: float = 0

class JournalHeader(BaseModel):
    doc_no: str  # YYYY########
    trx_date: date

class JournalResult(BaseModel):
    header: JournalHeader
    lines: List[JELine]
    warnings: List[str] = []

# ——————————— GL / FS ———————————
class GLBalance(BaseModel):
    account: str
    balance: float

class FinancialStatements(BaseModel):
    PL: List[dict]
    BS: List[dict]
```

```
    CF: List[dict]

# ———————————— Bank Reconciliation ————————————
class ReconReport(BaseModel):
    matched: List[dict]
    unmatched: List[dict]
    match_rate: float

# ———————————— Closing Step Result ————————————
class StepResult(BaseModel):
    step: int
    name: str
    status: str  # success | fail
    message: str
```

> 더미 데이터 예시는 dummy_data/ 경로에 invoice_sample.json, fs_sample.json 등으로 저장하고, FastAPI 엔드포인트에 response_model=InvoiceDict 형식으로 지정하여 Swagger 문서에도 자동 반영합니다.

2-5. Entity · Article 마스터 자동등록 모듈 설계

구분	내용
모듈 위치	`parser_service/master_sync.py` (FastAPI 레이어)
엔드포인트	`POST /api/masters/sync/` – Body: `InvoiceDict` JSON
입력	**InvoiceDict** (Parser 출력)
처리 단계	1. 거래처(Entity) 등록 → 코드 `ENT######` 연속 발번

1. 품목(Article) 등록 → 코드 `ART######` 연속 발번
2. 고정자산·재고·서비스 자동 분류 (키워드 매핑 YAML)
3. 다국어 필드(`_native` , `_en`) 함께 저장 | | **출력(JSON)** | `List[MasterResult]` (status: created | existing) | | **언어 처리** | 언어 감지 실패 시 **native = en** 로 저장 | | **검증** | 필수 필드 누락 → HTTP 400 + `error_detail` |

2-6. 다국어 OCR Parser 모듈 상세 요구사항 (추가)

구분	내용
모듈 위치	`parser_service/ocr_parser.py`
엔드포인트	`POST /api/parse/` – Form-Data: 파일 PDF
처리 파이프라인	1. **OCR**: Tesseract or Google Vision

1. **언어 감지**: langdetect
2. **번역**: (비-영어인 경우) DeepL API or Gemini Translate → `_en` 필드 생성
3. **InvoiceDict 조립** | | **출력 모델** | `InvoiceDict` (2-4 스키마) | | **검증 및 예외** | • 필수 키 누락 → HTTP 400
   • 번역 API 실패 시 `*_en = *_native` 로 fallback
   • OCR 결과 NULL → 에러 메시지 반환 | | **샘플 코드 (함수)** | ```python from .schemas import InvoiceDict

def parse_invoice(file: UploadFile) → InvoiceDict: """이미지 기반 PDF를 OCR → JSON (다국어 지원)"""

```
### 2-7. JE Build 모듈 상세 요구사항 (추가)
| 구분 | 내용 |
|------|------|
| **모듈 위치** | `parser_service/je_builder.py` |
| **엔드포인트** | `POST /api/je/build/` – Body: `InvoiceDict` |
| **회계 규칙** | • 자산 → `Fixed Asset`<br>• 상품 → `Sales Revenue`<br>• 서비스 → `Service Revenue`<br>• 공급자
```

기준 AR / AP 결정 |
전표번호	`YYYYMMDD-########` 연속 발번
검증	1. 총 Debit = Credit 2. `total_amount` = 분개 합계
출력 모델	`JournalResult`
금액 표현	소수점 둘째 자리 (`round(x, 2)`)

2-8. 전표 검증 모듈 설계 (Validation Module — VAT·중복 인보이스 포함 최종본)
구분	내용
모듈 위치	`parser_service/validator.py`
엔드포인트	`POST /api/validate/` – Body: `{invoice: InvoiceDict, journal: JournalResult, existing_invoices: List[InvoiceDict]}`
입력	• `InvoiceDict` (Parser 결과) • `JournalResult` (JE Build 결과) • `existing_invoices` : 기 업로드 인보이스 리스트
검증 항목
① **회계 기본** – 총 Debit = Credit, 필수 필드(`invoice_id`, `issuance_date`, `supplier.name_en`, `items`) 존재 확인
② **금액** – `InvoiceDict.total_amount` = JE 라인 합, 각 품목 `qty * unit_price` 합과 일치
③ **언어** – `*_native`, `*_en` 모두 존재, 언어 감지 실패 시 fallback(`native = en`) 확인
④ **마스터 연계** – JE 거래처·품목이 Entity/Article 마스터에 존재 여부 → 미존재 시 `warnings`
⑤ **VAT** – 각 품목에 `vat_rate` 필드 필수 (`"0%"|"7%"|...`) • 누락 시 `errors`: "Missing VAT rate for item X" • (선택) 품목 VAT 합 = 예상 VAT 합?
⑥ **중복 인보이스** – `existing_invoices` 중 `supplier.name_en` + `issuance_date` + `total_amount` + `items` 갯수가 모두 동일하면 `warnings`: "This invoice appears to be a duplicate of ..."
출력 모델	`ValidationReport` ```json { "is_valid": false, "errors": ["Invoice total does not match JE total.", "Missing VAT rate for item 2"], "warnings": ["Supplier not found in Entity master", "This invoice appears to be a duplicate ..."] } ```
함수 시그니처	```python from .schemas import InvoiceDict, JournalResult, ValidationReport def validate_entry(invoice: InvoiceDict, journal: JournalResult, existing_invoices: list) -> ValidationReport: """Invoice & JE 정합성 + VAT + 중복 여부 자동 검증""" ```
처리 규칙	• `is_valid` = `errors` 비어있을 때만 True • 오류 발생 시 후속 JE Build·GL Posting 보류 • `warnings` 는 사용자가 무시 후 진행 가능

ValidationReport 스키마
```python
class ValidationReport(BaseModel):
    is_valid: bool
    errors: List[str] = []
    warnings: List[str] = []
```

2-9. GL Posting 모듈 설계 (Trial Balance 생성)

구분	내용
모듈 위치	`parser_service/gl_posting.py`
엔드포인트	`POST /api/gl/post/` – Body: `JournalResult.lines`

입력 예	```json
[	
{"account":"Accounts Receivable","debit":1600000,"credit":0},	
{"account":"Sales Revenue","debit":0,"credit":1600000}	
]	

| **처리 로직** | 1. **계정별 집계**: 동일 `account` 기준 `debit`, `credit` 합산
2. **잔액 계산**: `balance = debit - credit` (소수점 둘째 자리 반올림)
3. **출력 정렬**: 계정 순서 유지, 모든 계정 포함 |
| **출력 모델** | `List[GLBalance]`<br>```json
[
  {"account":"Accounts Receivable","debit":1600000,"credit":0,"balance":1600000},
  {"account":"Sales Revenue","debit":0,"credit":1600000,"balance":-1600000}
]
``` |
| **규칙** | • `debit`/`credit`는 양수 절대값 유지
• 음수는 `balance` 필드로만 표현
• 사용된 모든 계정은 출력에 포함 |
| **검증** | 1. 총 Debit 합 = 총 Credit 합
2. JE 라인의 모든 계정이 결과에 존재 |
| **함수 시그니처** | ```python
from .schemas import JELine, GLBalance

def post_gl(lines: List[JELine]) -> List[GLBalance]:
    """JournalResult.lines → Trial Balance(계정별 잔액)"""
``` |

------|------|
모듈 위치	`parser_service/validator.py`
엔드포인트	`POST /api/validate/` – Body: `{invoice: InvoiceDict, journal: JournalResult}`
입력	`InvoiceDict`, `JournalResult`
검증 항목	**① 회계기본**: 총 Debit = Credit, 필수 필드 누락 체크 **② 금액**: `total_amount` = JE 합계, `qty * unit_price` 일치 **③ 언어 필드**: `*_native`, `*_en` 모두 존재, fallback 사용 여부 **④ 마스터 연계(선택)**: 미등록 거래처·품목 경고 출력
출력 모델	`ValidationReport` *(아래 스키마 참조)*
| **함수 시그니처** | ```python
from .schemas import InvoiceDict, JournalResult, ValidationReport

def validate_entry(invoice: InvoiceDict, journal: JournalResult) -> ValidationReport:
    """Invoice & JE 일관성 자동 검증"""
``` |
| **오류/경고 규칙** | • 오류: 치명적 불일치 → `errors` 배열에 메시지
• 경고: 마스터 미존재·fallback 등 → `warnings` 배열 |

ValidationReport 스키마 (추가)
```python
class ValidationReport(BaseModel):
    is_valid: bool
    errors: List[str] = []
    warnings: List[str] = []
```

- -----|------| | **모듈 위치** | `parser_service/je_builder.py` | | **엔드포인트** | `POST /api/je/build/` – Body: `InvoiceDict` | | **회계 규칙** | • 자산 → `Fixed Asset`
- 상품 → `Sales Revenue`
- 서비스 → `Service Revenue`
```

• 공급자 기준 AR / AP 결정 | | **전표번호** | YYYYMMDD-######## 연속 발번 | | **검증** | 1. 총 Debit = Credit

2. total_amount = 분개 합계 | | **출력 모델** | JournalResult | | **금액 표현** | 소수점 둘째 자리 (round(x, 2)) |

- -----|------| | **모듈 위치** | parser_service/master_sync.py (FastAPI 레이어) | | **엔드포인트** | POST /api/masters/sync/ – Body: InvoiceDict JSON | | **입력** | OCR Parser 가 반환한 **InvoiceDict** | 예: invoice_sample.json | | **처리 단계** | 1. **거래처(Entity) 등록** • supplier.name_native / name_en 로 조회 → 없으면 ENT###### 코드 생성(6자리 연속)2. **품목(Article) 등록** • items[*] 각각에 대해 description_en 기준 조회 • 고정자산·재고·서비스 자동 분류 룰: - 키워드 매핑 YAML dummy_data/article_mapping.yml (예: desk → FixedAsset)3. **다국어 저장** • 각 마스터 모델에 _native , _en 두 필드 저장4. **결과 집계** → MasterResult 리스트 생성 | | **출력(JSON)** | List[MasterResult] ```json [{"type":"Entity","code":"ENT000123","name_native":"삼성전자","name_en":"Samsung Electronics","status":"created"}, {"type":"Article","code":"ART000412","description_native":"데스크탑 컴퓨터","description_en":"Desktop Computer","status":"existing"}]

| **예외/검증** | • 언어 감지 실패 시 `*_native == *_en` 저장
• 필수 필드 누락 → HTTP 400 과 `error_detail` 반환 |
| **함수 시그니처** | ```python
def sync_masters(invoice: InvoiceDict) → List[MasterResult]:
    """Parser 결과로 Entity·Article 마스터 자동 등록/조회"""
``` |
| **연속 번호 로직** | `ENT` / `ART` prefix + 6자리, 삭제되어도 번호 건너뛰지 않음 |

2-10. FS Aggregate 모듈 설계 (PL·BS·CF 집계)
구분	내용
모듈 위치	`parser_service/fs_aggregate.py`
엔드포인트	`POST /api/fs/aggregate/` – Body: `List[GLBalance]`
| **입력 예** | ```json
[
  {"account": "Accounts Receivable", "debit": 1600000, "credit": 0, "balance": 1600000},
  {"account": "Sales Revenue", "debit": 0, "credit": 1600000, "balance": -1600000}
]
``` |
| **처리 로직** | 1. **계정 매핑**: 사전 정의된 `account_mapping.yml` 을 로드하여
  • `Sales Revenue` → PL, `Accounts Receivable` → BS …
2. **금액 집계**: 동일 항목별 `balance` 절대값을 합산
3. **표기 언어**: 항목명은 영어 기준으로 출력 (`Sales Revenue`, `Current Assets` 등) |
| **출력 모델** | `FinancialStatements`<br>```json
{
  "PL": [{"account": "Sales Revenue", "amount": 1600000}],
  "BS": [{"account": "Accounts Receivable", "amount": 1600000}],
  "CF": []
}
``` |
| **규칙** | • PL 에는 수익·비용만 포함
• BS 는 자산·부채·자본만 포함
• CF 는 계정·분개유형 분석으로 유추 가능 시 포함
• `amount` 는 절대값 사용, 부(-) 표시는 BS·CF Logic 에 따라 결정 |
| **검증** | • (옵션) PL 총계 = JE 매출·비용 합
• BS: 자산 = 부채 + 자본 (가능 시 계산)
• 매핑되지 않은 계정은 `warnings` 리스트에 추가 |
| **함수 시그니처** | ```python
from .schemas import GLBalance, FinancialStatements

def aggregate_fs(trial_balance: List[GLBalance]) → FinancialStatements:
    """Trial Balance → PL / BS / CF 집계"""
``` |
| **경고 처리** | 매핑표에 없는 계정은 `warnings`에 "Unmapped account: XXX" 추가 |

2-11. Closing Runner 모듈 설계 (10-Step 월마감)
| 구분 | 내용 |

|------|------|
| **모듈 위치** | `parser_service/closing_runner.py` |
| **엔드포인트** | `POST /api/closing/run/` – Body: `RunData` |
| **입력 모델** | ```json
{
  "gl_balances": [ ... ],        // 5단계 GL Posting 결과
  "financial_statements": { ... }, // 6단계 FS Aggregate 결과
  "validation_report": { ... }     // 4단계 검증 결과
}
``` |
| **10-Step 프로세스** |
| ① 미결 인보이스 점검
② 전표 검증 상태 확인
③ 고정자산 감가상각
④ 선급·미지급 비용 이연
⑤ 마감 전 분개 조정 가능 여부 확인
⑥ 손익 마감 (PL→Retained Earnings)
⑦ 기말 잔액 이월 준비
⑧ 회계기준(IFRS, K-GAAP 등) 보고서 생성
⑨ 전표 Lock 및 마감 플래그 기록
⑩ 마감 결과 로그 및 보고서 생성 |
| **출력 모델** | `ClosingResult` (List[StepResult])<br>```json
[
  {"step": 1, "name": "Check Open Invoices", "status": "success", "message": "All invoices matched"},
  {"step": 2, "name": "Validation Summary", "status": "warning", "message": "1 invoice missing translation"}
]
``` |
| **처리 특성** | • 각 Step 실패해도 중단 X, `status` = fail & 로그 기록
• 심각 오류는 `status` = error → 사용자 Confirm 필요
• 리얼타임 WebSocket/HTMX SSE 로 로그 append 가능 |
| **결과 저장** | `outputs/closing_log.json` & DB `ClosingLog` 테이블 |
| **함수 시그니처** | ```python
from .schemas import RunData, StepResult

def run_month_end(data: RunData) -> List[StepResult]:
    """10-Step 월마감 자동 실행"""
``` |
| **스케줄링** | Celery Beat 월 1회 예약, 수동 Trigger 가능 |

2-12. DW Loader 모듈 설계 (데이터웨어하우스 적재)
구분	내용
모듈 위치	`parser_service/dw_loader.py`
엔드포인트	`POST /api/dw/load/` – Body: `DWLoadRequest`
| **입력 모델** | ```json
{
  "journal_header": {"doc_no":"20250418-000001","trx_date":"2025-04-18"},
  "journal_lines": [
    {"account":"Accounts Receivable","debit":1600000,"credit":0,"description":"AR from Samsung Electronics"},
    {"account":"Sales Revenue","debit":0,"credit":1600000,"description":"Sale of Desktop Computers"}
  ],
  "financial_statements": {
    "PL": [...],
    "BS": [...],
    "CF": [...]
  }
}
``` |
| **처리 로직** | 1. **JE_Fact 적재**: `journal_header` + `journal_lines` → `je_fact` 테이블
  • 키 컬럼: `doc_no`, `trx_date`, `account`
2. **FS_Fact 적재**: `financial_statements` 각 항목 → `fs_fact` 테이블
  • 키 컬럼: `period`, `statement_type`, `account`
3. **Idempotent**: `ON CONFLICT (doc_no, account)` → UPDATE 금액
4. **로그 기록**: 성공/실패 결과를 `outputs/dw_log.json` 및 DB `DWLoadLog` 저장 |
| **출력 예** | ```json

{
  "success": true,
  "je_records": 2,
  "fs_records": 3,
  "message": "DW Load completed successfully."
}
``` |
| **규칙** | • JE & FS 모두 적재 성공 시 `success=true`
• 하나라도 실패 시 `success=false`, `message` 에 상세 오류
• PostgreSQL 또는 BigQuery 대상 · 커넥터는 `.env` 설정 로드 |
| **함수 시그니처** | ```python
from .schemas import DWLoadRequest, DWLoadResult

def load_dw(data: DWLoadRequest) → DWLoadResult:
    """JE + FS 데이터를 데이터웨어하우스 Fact 테이블로 적재"""
``` |
| **스키마 (추가)** | ```python
class DWLoadRequest(BaseModel):
    journal_header: JournalHeader
    journal_lines: List[JELine]
    financial_statements: FinancialStatements

class DWLoadResult(BaseModel):
    success: bool
    je_records: int
    fs_records: int
    message: str
``` |
| **검증** | • JE 라인 ≥1, FS PL·BS·CF 키 존재
• DB 커넥션 실패 시 500 반환 |

2-13. i18n Lookup 모듈 설계 (다국어 문자열 조회)
구분	내용
모듈 위치	`accounting/i18n_stub.py`
엔드포인트	`GET /api/i18n/lookup/` – Query: `key`, `lang`
입력 예	`/api/i18n/lookup/?key=accounts_receivable&lang=ko`
처리 로직	1. `dummy_data/locales/{lang}.json` 파일 로드 (메모리 캐시)
2. `key` 존재 여부 확인 → 값 반환
3. 파일 없거나 키 누락 시 Fallback 메시지 반환
| **출력 예** | ```json
{"result": "매출채권"}
``` |
| **규칙** | • 키 없으면 `"Translation not found"`
• 언어코드 미지원 시 `"Unsupported language"`
• 번역 파일은 앱 시작 시 캐싱 가능 |
| **함수 시그니처** | ```python
from pathlib import Path
import json, functools

@functools.lru_cache(maxsize=32)
def _load_locale(lang: str) → dict:
    path = Path("dummy_data/locales") / f"{lang}.json"
    if not path.exists():
        raise FileNotFoundError
    return json.loads(path.read_text(encoding="utf-8"))

```
def lookup_i18n(key: str, lang: str) → str:
    """다국어 JSON에서 번역 문자열 조회"""
``` |
| **스키마 (추가)** | ```python
class I18NRequest(BaseModel):
    key: str
    lang: str

class I18NResponse(BaseModel):
    result: str
``` |
| **검증** | • 지원하지 않는 `lang` → 400 + "Unsupported language"
• JSON 파일에 `key` 없으면 `result` = "Translation not found" |
| **확장 고려** | • 사용자별 커스텀 번역을 DB `UserTranslation` 테이블에 저장 후, 기본 번역 위에 오버레이 |

2-14. Dashboard Embed 모듈 설계 (시각화 대시보드)
[dashboard section content preserved as above]

2-15. 추가 개선 Todo 항목 요약 (우선순위 ★ 표시)
No	항목	설명	필요도
1️⃣	**자동분개 엔진 설계**	품목별 자동 회계처리 규칙(계정, VAT 포함 여부 등) 정의	★★★★★
2️⃣	**품목 유형 분류 로직 강화**	`description_en` 기반 고정자산·비용·서비스 자동 분류 룰 구체화	★★★★☆
3️⃣	**부가세(VAT) 항목 자동 분개**	VAT 포함 품목의 VAT Payable / Receivable 라인 자동 생성	★★★★☆
4️⃣	**회계처리 유형 감지 로직**	매출·매입·선급·감가상각 등 거래 유형 인식 → JE 라인 태그	★★★☆☆
5️⃣	**Fallback 플래그 저장**	번역 실패(native → en 복사) 시 필드 `fallback=True` 저장	★★☆☆☆
6️⃣	**모듈간 실행 로그 기록**	각 모듈 결과·에러코드 → `ModuleStatusLog` 저장, Dashboard 표시	★★☆☆☆
7️⃣	**AI 분개 추천 로그**	자동 분개 결과를 `outputs/auto_split_log.json` 저장, 사용자 리뷰	★☆☆☆☆
8️⃣	**품목 단가×수량 불일치 처리**	합계 불일치 시 반올림 or 오류/경고 메시지 설계	★☆☆☆☆

> 위 항목은 **Phase 2 개선 과제**로 스프린트 계획 시 점수와 담당자를 지정해 구체화합니다.

2-16. Auto Split 모듈 설계 (품목 단위 자동 분개)
구분	내용
모듈 위치	`parser_service/auto_split.py`
호출 위치	`je_builder.py` 내부에서 `auto_split(invoice)` 호출
| **함수 시그니처** | ```python
from .schemas import InvoiceDict, JELine
from typing import List

def auto_split(invoice: InvoiceDict) → List[JELine]:
    """품목 유형·VAT·거래 형태를 고려한 JE 라인 자동 생성"""
``` |
| **처리 로직** | 1. **품목 유형 결정**: `description_en` → 고정자산/재고/서비스 키워드 매핑
2. **계정 자동 매핑**: 유형 + 거래 방향(매출/매입) → 계정 코드
3. **VAT 분리**: 품목에 `vat_rate` 존재 시 VAT Payable / Receivable 추가
4. **금액 계산**: `qty * unit_price` → JE 라인 금액, 소수점 2자리
5. **설명**: `description_en` 기반 JE 라인 `description` 필드 구성 |
| **출력** | `List[JELine]` – 품목라인 + VAT 라인(필요시) |
| **검증** | • 생성된 라인의 총 Debit = Credit
• VAT 라인 합계 = 품목 VAT 합계 |

| **로그** | `outputs/auto_split_log.json` 에 품목→계정 변환 결과 저장 |
| **확장 고려** | • ML 기반 품목 ↔ 계정 매핑 추천 결과 저장 및 학습 |

2-17. 파란색 배경 + 밑줄 문장 반영 (최신 수정 사항)
아래 항목은 문서 내 파란색 배경으로 표기 되고 밑줄이 쳐진 변경 지시를 반영한 내용입니다.

No	변경 지시	반영 내역
1	"Entity" → **Counterparty**	- 모든 모듈·스키마·UI 레이블에서 *Entity* 용어를 *Counterparty* 로 교체. - `MasterResult.type` enum, 화면 표기, 예시 JSON 수정.
2	`Supplier` 클래스를 **Counterparty** 로 변경	- `parser_service/schemas.py` 에서 `class Supplier` → `class Counterparty`. - `InvoiceDict` 필드 `supplier: Supplier` → `counterparty: Counterparty`. - 파싱/마스터 Sync·검증 모듈 에서 동일하게 필드명 변경.
3	**JE 라인 key** `account` → `account_no`	- `JELine` 스키마 필드명 변경. - JE Build 결과 JSON·GL Posting 입력/출력 예시 업데이트. - `post_gl()` 집계 key 역시 `account_no` 사용.
4	**JE Build 회계 규칙** 세분화	- 매출: Counterparty 가 *수령자* → `Sales Revenue` / `Service Revenue`. - 매입: Counterparty 가 *발행자* → `Inventory`·`Fixed Asset`·비용계정 추천. - 선수금·선급금·VAT 자동 라인 생성 로직 명시.
5	**중복 인보이스 기준** 수정	`counterparty.name`, `issuance_date`, `invoice_id`, `total_amount` 네 가지가 일치 시 중복 경고.
6	**이상 감지 로직** 추가	- 거래처 정보 변경, 동일 품목 금액 10%↑ 차이, 전년도 인보이스, 내부 관계자 거래 시 `warnings` 알림.
7	**음수 표시 규칙**	모든 보고서에서 음수 금액은 *(괄호)* + **빨간색** 폰트로 시각화.
8	**GL Posting 추가 규칙**	- 계정 내 거래일자순 정렬. - GL 박스 하단에 *Opening Balance / Net Change / Ending Balance* 표시. - 총 Debit = Credit 검증 항목 **삭제**.
9	**Closing Runner 보고서**	월마감 과정에서 신규 생성된 전표 번호(`doc_no`)를 마감 결과 로그에 포함.

> 각 모듈 구현 시 위 변경 사항을 우선 적용한 후 기존 요구사항을 따라 개발하세요.

2-18. 회계 데이터 파일 참고 경로
구분	경로/URL	설명
더미 테스트 데이터	`dummy_data/` (repo 내부)	• `invoices/*.pdf` (**11개 샘플 인보이스**, *회계데이터.zip* 압축 해제) • `invoices/1-11 *.pdf` (샘플 인보이스 일괄 테스트용) • `sample_camt.xml` • `coa_kifrs.yml` (계정 과목표) • `locales/*.json` (다국어 번역)` (다국어 번역)
실제 운영 데이터 마이그레이션	`/mnt/shared/accounting_data/YYYYMM/` (사내 NAS)	• ERP 에서 추출된 CSV / Excel 원장, 거래처·품목 마스터 원본 • 매월 1일 ETL 스크립트로 자동 sync
BI 용 DW 스냅샷	`bigquery://corp-dw.accounting.*`	• `je_fact`, `fs_fact`, `closing_log` 테이블 • Metabase / Superset 대시보드 소스
Git LFS 저장소 *(대용량 PDF)*	`https://git.example.com/ai_accounting/pdf_repo.git`	• 과거 인보이스 PDF 50k건 저장, 파서 성능 개선용

> **개발자는** 로컬 환경에서 `dummy_data/` 를 기본 참조하고, 운영 DW 쿼리는 BigQuery 테스트 계정으로 접근하세요. NAS 원장 파일은 권한 설정 후 마운트 필요.
